Algorithmen der Spracherkennung auf massiv parallelen SIMD–Rechnern

Diplomarbeit von

Tilo Sloboda
(sloboda@ira.uka.de)

am
Institut für Programmstrukturen und Datenorganisation
Universität Karlsruhe

14. Januar 1992

Hauptreferent	:	Prof. Walter F. Tichy
Koreferent	:	Prof. Alex H. Waibel
Betreuer	:	Dipl.-Inform. Lutz Prechelt

Zusammenfassung

Moderne Spracherkenner für kontinuierlich gesprochene Sprache, wie die in JANUS verwendeten, benötigen bei Vokabularen um 500 Wörter minutenlange Rechenzeiten auf konventionellen Arbeitsplatzrechnern. Hierbei eine Annäherung an Echtzeit–Spracherkennung zu erzielen war das Hauptziel dieser Arbeit.

Untersucht wurden dabei die rechenintensiven Algorithmen der Spracherkennung: die digitale Signalvorverarbeitung, die Analyse der Sprachsignale – basierend auf neuronalen Netzen – am Beispiel von *Linked Predictive Neural Networks (LPNNs)*, sowie die Suche der N besten Satzhypothesen für die Spracheingabe.

Die parallelisierten Algorithmen wurden auf einer MasPar MP-1, einem hochparallelen SIMD–Rechner, implementiert. Die parallele Implementation eines *Learning Vector Quantization (LVQ)*–Erkenners wird skizziert.

Die beschriebenen parallelen Algorithmen lassen sich auch auf andere SIMD–Rechner portieren.

Abstract

State-of-the-art speech recognizers for continuously spoken language, like the ones used in JANUS, using vocabulary sizes of 500 words, keep conventional workstations busy for minutes. Therefore it was the main goal to get close to realtime speech recognition.

The examined, most time consuming algorithms are the digital signal processing, the neural network based speech analysis with Linked Predictive Neural Networks (LPNNs), and the N–best search of sentence hypotheses for the input speech.

The parallel algorithms were implemented on a MasPar MP-1, a massively parallel SIMD machine. The parallel implementation of a Learning Vector Quantization (LVQ) recognizer is scetched.

The parallel algorithms presented can be ported to other massively parallel SIMD machines.

Hiermit erkläre ich, daß ich die vorliegende Diplomarbeit selbständig und ohne unzulässige Hilfsmittel angefertigt habe. Alle verwendeten Quellen sind im Literaturverzeichnis aufgeführt.

Karlsruhe, den 14. Januar 1992

(Tilo Sloboda)

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Spracherkennung

Bisher kann der Mensch nur umständlich mit Rechnern kommunizieren: über primitive Schnittstellen wie Tastatur und Maus. Es gibt aber viele Situationen, in denen eine Spracheingabe sehr von Nutzen wäre.

Denkbar wären zum Beispiel: Freisprech-Telefonapparate, bei denen man den Namen des Teilnehmers oder die gewünschte Telefonnummer spricht (Autotelefone), automatische Diktiergeräte, die das Diktierte als maschinenlesbaren Text abspeichern können, freihändige Steuerungen von medizinischen Geräten oder von Meßinstrumenten, bei denen man Meßbereiche und Funktionsmodi durch akustische Befehle umschalten kann, Hilfsmittel für Behinderte, wie Steuerungen von Hilfsgeräten; abschließend sei noch die automatische Sprachübersetzung genannt — sie könnte der Kommunikation zwischen Menschen neue Möglichkeiten eröffnen.

Rechenleistung

Heutige Spracherkenner für kontinuierlich gesprochene Sprache brauchen beachtliche Rechenleistungen: bei Vokabularen um 500 Wörter braucht die Erkennung eines Satzes auf einem konventionellen Arbeitsplatzrechner mehrere Minuten. Eine solche Zeitdauer für die Erkennung wird in einer realen Anwendung nicht toleriert, denn ein Dialog in einer dem Menschen gewohnten Geschwindigkeit ist so nicht möglich.

Zu den zeitaufwendigsten Teilaufgaben eines Spracherkenners gehören: die digitale Signalvorverarbeitung, die Analyse der Sprachsignale sowie die Suche der N besten Satzhypothesen mittels dynamischer Programmierung. Um die Zeitdauer für die Erkennung auf ein erträgliches Maß zu reduzieren, muß man diese Teilaufgaben erheblich beschleunigen. Dies kann man durch Einsatz von Parallelrechnern erreichen.

Parallelrechner

Die zeitaufwendigen Algorithmen im Erkennerteil von JANUS [44], einem sequentiellen Sprach- zu Sprach Übersetzungssystem, beinhalten Iterationen über der Zeit oder über den Wörtern des Vokabulars. Die Parallelisierung dieses Problems scheint sehr aussichtsreich, da es eine hohe Datenparallelität beinhaltet. SIMD–Rechner sind für die Lösung derartig datenparalleler Probleme besonders geeignet.

Ziele

Das unbefriedigende Zeitverhalten des sequentiellen JANUS soll durch Parallelisierung verbessert werden — die Erkennung sollte fast sofort nach der Aussprache des Gesprochenen abgeschlossen werden. Die mögliche Größe des Vokabulars soll dabei vergrößert werden. Hierbei ist auch von Interesse, wie sich der hierbei verwendete SIMD–Rechner, eine MasPar MP-1, bezüglich seiner Anwendbarkeit verhält.

Gliederung

Im Kapitel 2 wird kurz auf Sprache und Spracherkennung im Allgemeinen eingegangen. Es wird auch ein Überblick über JANUS und die MasPar MP-1 gegeben. Die Kapitel 3, 4 und 5 beschäftigen sich mit den drei zeitaufwendigen Teilen der Spracherkennung: der Signalvorverarbeitung, der Analyse der Sprachsignale und der Suche nach den besten Satzhypothesen. In Kapitel 6 werden die Ergebnisse ausgewertet. In Kapitel 7 werden die mit der MasPar MP-1 gesammelten Erfahrungen zusammengefasst. Ein Ausblick wird in Kapitel 8 gegeben. Anhang A enthält Beispiele für die wichtigsten Konfigurationsdateien von JANUS und einige Erklärungen dazu. Der Anhang B enthält eine Liste der Wörter im verwendeten deutschen Vokabular, sortiert nach der Anzahl ihrer Vorgänger. Im Anhang C befindet sich eine Übersicht über die während dieser Diplomarbeit entstandene Software, die sich in Anhang D befindet.

Kapitel 2

Grundlagen

Nach einigen Überlegungen zur Spracherkennung wird eine Übersicht über JANUS gegeben und anschließend die MasPar MP-1 beschrieben.

2.1 Spracherkennung

Die akustische Variabilität der Sprache ist sehr groß: Sprechgeschwindigkeit, Satzmelodie, Tonhöhe, Betonung, Dialekte, Sprechereigenheiten und Prosodie werden vom Menschen weitestgehend unbewusst wahrgenommen. In Spracherkennern können nur die Merkmale ausgewertet werden, die direkte Phoneminformationen beinhalten.

Um die grosse Anzahl an Merkmalen zu verringern, werden außerdem starke Einschränkungen gegenüber der natürlichen Sprache getroffen. Es wird von einer wohldefinierten Sprache, wie Hochdeutsch, ausgegangen. Das Vokabular wird festgelegt und dadurch eingeschränkt. Die Wörter des Vokabulars werden in einer Phonemschreibweise gespeichert. Dabei werden Aussprachevarianten festgelegt und Annahmen über die Anzahl und die Feinstruktur von Phonemen gemacht.

Um trotzdem ein flexibles System zu erhalten, werden die meisten Einschränkungen so getroffen, daß sie einfach verändert werden können. Das bedeutet eine große Anzahl von variierbaren Parametern in einem Spracherkenner.

Als Beispiel eines Spracherkenners wird hier der spracherkennende Teil von JANUS verwendet.

2.2 JANUS

Das Sprach-zu-Sprach Übersetzungssystem JANUS[1] ist in der Lage, kontinuierlich gesprochene Sprache mit einem großen Vokabular von etwa 500 Worten zu erkennen und in andere Sprachen zu übersetzen [44].

JANUS gliedert sich in folgende Teile:

Spracheingabe

A/D – Wandlung

ADC-Daten

Signalvorverarbeitung

FFT-Daten

Analyse der Sprachsignale

Bewertungen

Suche der besten Satzhypothesen

Satzhypothesen

Zerteilung

Zwischensprache (interlingua)

Übersetzung

Zielsprache

Sprachsynthese

Sprachausgabe

Abbildung 2.1: Aufbau von JANUS

Die Eingabe erfolgt entweder in englischer oder in deutscher Sprache. Nach der digitalen Signalvorverarbeitung (Kapitel 3) folgt die Analyse der Sprachsignale (Kapitel 4), die in JANUS auf neuronalen Netzen basiert. Hier kann zum Beispiel ein *Linked Predictive Neural Network (LPNN)*-Erkenner zur Vorhersage von Merkmalsvektoren verwendet werden. Die Differenzen zwischen den damit vorhergesagten und den tatsächlichen Merkmalsvektoren werden als Bewertungen[2] betrachtet, die für die Suche der besten Satzhypothesen verwendet werden (Kapitel 5).

[1] Eine Kooperation der Universität Karlsruhe, der Carnegie Mellon University(USA), ATR International (Japan) und Siemens

[2] Diese Bewertungen können bei anderen Erkennern (zum Beispiel beim *Learning Vector Quantization (LVQ)*-Erkenner) direkt von neuronalen Netzen erzeugt werden.

Es schließen sich Zerteilung und Übersetzung in eine Zwischensprache an. Die Zwischensprache kann dann ins Englische, Japanische oder Deutsche übersetzt werden, und wird schließlich durch ein Sprachausgabegerät hörbar gemacht.

Die verschiedenen Teile von JANUS sind austauschbar — zum Beispiel kann je nach verwendetem Erkenner, sprecherabhängige oder auch sprecherunabhängige Erkennung erfolgen. Sowohl der im folgenden beschriebene sprecherabhängige LPNN–Erkenner, als auch der mittlerweile fertiggestellte sprecherunabhängige LVQ–Erkenner basieren auf neuronalen Netzen.

Bei einer Spracheingabe von 2,5 Sekunden Dauer, brauchte eine weitestgehend unbelastete DEC 5000 folgende Rechenzeiten: für die Signalvorverarbeitung etwa 3,2 Sekunden; für die Vorwärtsberechnungen der LPNNs 9,8 Sekunden; für die Berechnung der besten Satzhypothese bei 115 Wörtern etwa 2 Sekunden; für die Berechnung der 3 besten Satzhypothesen bei 115 Wörtern 2,5 Minuten und bei 447 Wörtern 28 Minuten[3]. Insgesamt braucht die vollständige Erkennung für den genannten Satz, etwa 30 Sekunden für die beste Satzhypothese bis zu 30 Minuten für die 3 besten Satzhypothesen.

Randbedingungen durch JANUS

Parameter	Englisch	Deutsch
verschiedene Phoneme	40	46
verschiedene Modelle	2	3
verschiedene Phonemsegmente[a]	118	136
Wörter im Vokabular	404	464(447)[b]
Phoneme im Vokabular	1838	3081
Phonemsegmente im Vokabular[c]	3512	9241

[a] entspricht der Anzahl der LPNNs.
[b] ohne(mit) Grammatik
[c] Phonemsegmente aller Wörter im Vokabular.

Tabelle 2.1: Die wichtigsten sprachabhängigen JANUS–Parameter

Da JANUS ein Forschungsprojekt ist, beinhaltet es eine Vielzahl von variierbaren Parametern. So werden die Topologie der neuronalen Netze, die Anzahl und die Modellierung der Phoneme, die Zuordnung der neuronalen Netze zu den Phonemmodellen und das Vokabular zur Laufzeit durch Konfigurationsdateien festgelegt. Dadurch wird auch die Quellsprache, die erkannt werden soll, festgelegt. Im Anhang A befindet sich eine Liste mit Beispielen für die Dateien, ihrem Format und den durch sie festgelegten Parametern. In der Tabelle 2.1 sieht man, wie die Anzahl der Phoneme, Phonemsegmente, Phonemzustände und Wörtern im Vokabular von der Eingabesprache abhängen. Falls nicht anders gesagt, werde ich mich im folgenden auf das deutsche JANUS mit 46 Phonemen, 136 Phonemsegmenten und 447 Wörtern im Vokabular beziehen.

[3] Dieser mit der Anzahl der Wörter im Vokabular quadratische Zeitzusammenhang könnte im bisherigen, sequentiellen Algorithmus durch das Verwenden einer Grammatik (siehe Kapitel 5.2.1) und einer Strahlsuche stark abgeschwächt werden. Allerdings stand kein solcher sequentieller Algorithmus für die Zeitmessungen zur Verfügung. Aus dem Fehlen dieser Verbesserungen erklärt sich auch der grosse Zeitunterschied zwischen dem *first best* und dem *N best* Algorithmus.

2.3 MasPar

Die MasPar MP-1 ist ein flexibel ausbaubarer SIMD-Rechner [16]. Er gliedert sich in einen separaten Vorrechner, dem *Front End (FE)*, und den eigentlichen Parallelrechner, dem *Back End (BE)* oder auch *Data Parallel Unit (DPU)*. Der Parallelrechner besteht aus einem sequentiellen Steuerrechner, der *Array Control Unit (ACU)*, sowie den parallelen Prozessorelementen, kurz PEs. Ihre Anzahl kann – je nach Ausbaustufe – zwischen 1024 und 16384 betragen. In der ACU stehen dem Benutzer 128 KByte Speicher zur Verfügung, pro Prozessorelement 16 KByte (optional 64 KByte). Als Vorrechner kann entweder eine VAX 3250 oder eine DEC 5000 angeschlossen werden.

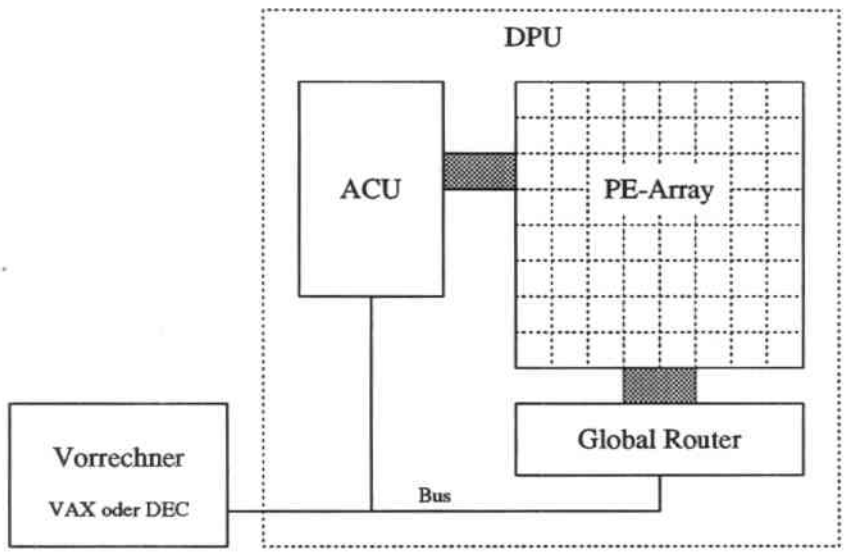

Abbildung 2.2: Aufbau der MasPar MP-1

Die Prozessorelemente sind rechteckig angeordnet. Zur Kommunikation zwischen den Prozessorelementen stehen zwei getrennte Verbindungsnetzwerke zur Verfügung, das *X-Net* und der *Global Router*. Das *X-Net* stellt eine toroide Verbindung zwischen den in einem Rechteck angeordneten Prozessorelementen her. Die Verbindungen eines Prozessorelementes zu seinen acht nächsten Nachbarn stellen dabei die möglichen Kommunikationsrichtungen dar. Bei einer Kommunikation werden alle daran teilnehmenden Prozessorelemente mit jeweils einem Prozessorelement in einer gegebenen, festen Richtung und in einem gegebenen, festen Abstand verbunden [16]. Der *Global Router* erlaubt freie Kommunikation zwischen einzelnen Prozessorelementen. Jeweils 4*4 Prozessorelemente werden, zusätzlich zu ihrer rechteckigen Anordnung, zusammengruppiert. Zwischen diesen Gruppen bestehen Verbindungen. Falls mehrere Prozessorelemente einer Gruppe kommunizieren wollen, werden diese Kommunikationen serialisiert [16]. Optional kann eine parallele Platteneinheit MasPar *Parallel Disk Array (MPDA)* [19] angeschlossen werden, die einen parallelen Zugriff auf Dateien ermöglicht.

An der Universität Karlsruhe stehen 4096 Prozessorelemente mit je 16 KByte RAM, sowie eine parallele Platteneinheit zur Verfügung. Als Vorrechner wird eine VAX 3250 verwendet. In Kürze wird der Rechner auf 16384 Prozessorelemente erweitert und erhält eine

DEC 5000 als Vorrechner. An der Carnegie Mellon University steht eine MasPar mit 4096 Prozessorelementen und einer DEC 5000 als Vorrechner.

Aus dem flexibelen Aufbau der MasPar und dem Wunsch nach Portierbarkeit der Programme, ergeben sich bei der Implementation zwei Forderungen. Die Programme müssen sich auf die jeweilige Konfiguration einstellen und, wenn möglich, voll skalierbar zu sein. Da die verschiedenen Vorrechner verschiedene Bytereihenfolgen in ihren Zahlendarstellungen verwenden, muß die Ein- und Ausgabe von Dateien in einem maschinenunabhängigen Format erfolgen.

Da Modula-2* [42, 25], ein paralleler Modula-2 Dialekt, als Programmiersprache noch nicht zur Verfügung stand wurde MPL [18], ein um parallele Sprachelemente erweiterter C-Dialekt verwendet.

Kapitel 3

Signalvorverarbeitung

Die Vorgehensweise bei der Signalvorverarbeitung wird beschrieben. Danach werden einige Implementationsmöglichkeiten aufgeführt und die erfolgte Implementation begründet.

3.1 Grundlagen

Das Sprachsignal wird in JANUS mit 16 kHz abgetastet und in einem 14 Bit A/D-Wandler in digitale Daten gewandelt. Das bisherige Programm zur digitalen Signalvorverarbeitung lädt die gesamten digitalen Sprachdaten an einem Stück aus dem Pufferbereich des Aufnahmegerätes [7]. Es wird über die gesamte Aufnahme äquidistant alle 5 ms ein Fenster fester Breite[1] geschoben und die in den Fenstern auftretenden minimalen und maximalen Werte berechnet. Diese Aussteuerungsdaten werden normiert und die Teile am Anfang und am Ende der Sprachaufnahme, die unter einem bestimmten Aussteuerungspegel liegen, werden abgeschnitten *(clipping)*. Über die verbleibenden Daten wird äquidistant alle 5 ms ein Hamming-Fenster geschoben und eine schnelle Fourier Transformation mit einer festen Fensterbreite berechnet. Aus den daraus pro Zeitpunkt resultierenden 128 Spektralkoeffizienten werden 16 *melscale*[2] –Koeffizienten [46] für jeden Zeitpunkt berechnet. Man hat nun für alle 5 ms der Spracheingabe je einen Vektor mit 16 *melscale*–Koeffizienten. Diese Vektoren werden über die Zeit paarweise gemittelt. Als Ausgabe erhält man so für alle 10 ms der Spracheingabe einen Merkmalsvektor mit 16 Koeffizienten. Diese werden so normiert, daß der kleinste in allen Merkmalsvektoren vorkommende Koeffizient Null und der größte Eins ist. Diese Ausgabedaten werden nach der Vorverarbeitung an einem Stück in eine Datei geschrieben, um dann vom Erkenner weiterverarbeitet zu werden.

In Abbildung 3.1 auf Seite 20 sieht man die *melscale*–Koeffizienten für den englischen Satz *„could you give me your name and address?“*. Man kann dort die vertikale Unterteilung, entsprechend der 16 Koeffizienten, sowie die zeitliche Ausdehnung der einzelnen Phoneme erkennen. Die unten zu sehende Beschriftung besteht aus der Markierung der Phonemgrenzen und den Phonemnamen. Die Markierungen wurden bei diesem Beispielsatz per Hand erstellt.

[1] Alle hier auftretenden Fensterbreiten umfassen 256 Abtastpunkte. Die Fensterbreiten und die Länge des Merkmalsvektors sind Konstanten des Programms.

[2] Bei den *melscale*–Koeffizienten handelt es sich um eine logarithmische Quantisierung der Frequenzdaten. Man möchte hierbei die frequenzabhängige Empfindlichkeit des menschlichen Ohrs nachahmen und bildet eine Klasseneinteilung der Spektralkoeffizienten in unterschiedlich große Klassen, deren Werte aufsummiert werden.

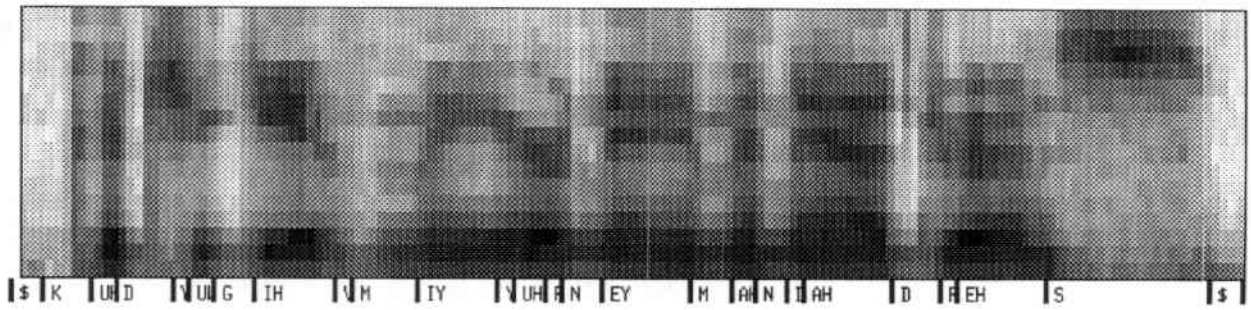

Abbildung 3.1: *melscale*–Koeffizienten für einen englischen Satz

3.2 Implementationsmöglichkeiten

Die Signalvorverarbeitung einer Spracheingabe von 2,44 Sekunden Dauer, brauchte auf einer DEC 5000 bisher etwa 3,2 Sekunden. Sie beginnt erst nachdem die Eingabe zu Ende gesprochen wurde (siehe Abbildung 3.2.a).

3.2.1 Algorithmische Beschleunigung der Vorverarbeitung

Das Ausführungsprofil der Signalvorverarbeitung zeigt, daß die Berechnung der (komplexwertigen) FFT etwa 51% der Zeit in Anspruch nahm (siehe Tabelle 3.1, Seite 23). Da die Eingabedaten aber nur reellwertig sind, führt eine komplexwertige FFT doppelt so viele Multiplikationen aus, wie zur Berechnung des Ergebnisses notwendig sind.

Allein durch die Verwendung einer schnellen Hartley Transformation (FHT) [5] anstelle einer schnellen Fourier Transformation (FFT) [6, 29], kann eine erhebliche Beschleunigung erzielt werden.

Hiebei ist die diskrete Fourier Transformation (DFT) :

$$F(\nu) = \frac{1}{N}\sum_{\tau=0}^{N-1} f(\tau)\exp\left(-i\frac{2\pi\nu\tau}{N}\right) = \frac{1}{N}\sum_{\tau=0}^{N-1} f(\tau)\left(\cos\left(\frac{2\pi\nu\tau}{N}\right) - i\sin\left(\frac{2\pi\nu\tau}{N}\right)\right) \quad (3.1)$$

wobei die Multiplikation, wie auch die Ein- und Ausgaben komplexwertig sind

und die diskrete Hartley Transformation (DHT) :

$$H(\nu) = \frac{1}{N}\sum_{\tau=0}^{N-1} f(\tau)\,\mathrm{cas}\left(\frac{2\pi\nu\tau}{N}\right) = \frac{1}{N}\sum_{\tau=0}^{N-1} f(\tau)\left(\cos\left(\frac{2\pi\nu\tau}{N}\right) + \sin\left(\frac{2\pi\nu\tau}{N}\right)\right) \quad (3.2)$$

wobei die Multiplikation, wie auch die Ein- und Ausgaben reellwertig sind.

Eine weitere Beschleunigung lässt sich, durch die Verwendung von Nachschlagetabellen für die Sinus- und Kosinusfunktion, sowie für die gespiegelte Binärdarstellung der Zahlen $0 \ldots N-1$, bei einer FHT–Fensterbreite von N, erzielen.

Insgesamt sollte in Folge der genannten, algorithmischen Verbesserungen die Signalvorverarbeitung weniger Zeit in Anspruch nehmen, als das Sprechen der Eingabe (siehe Abbildung 3.2.b).

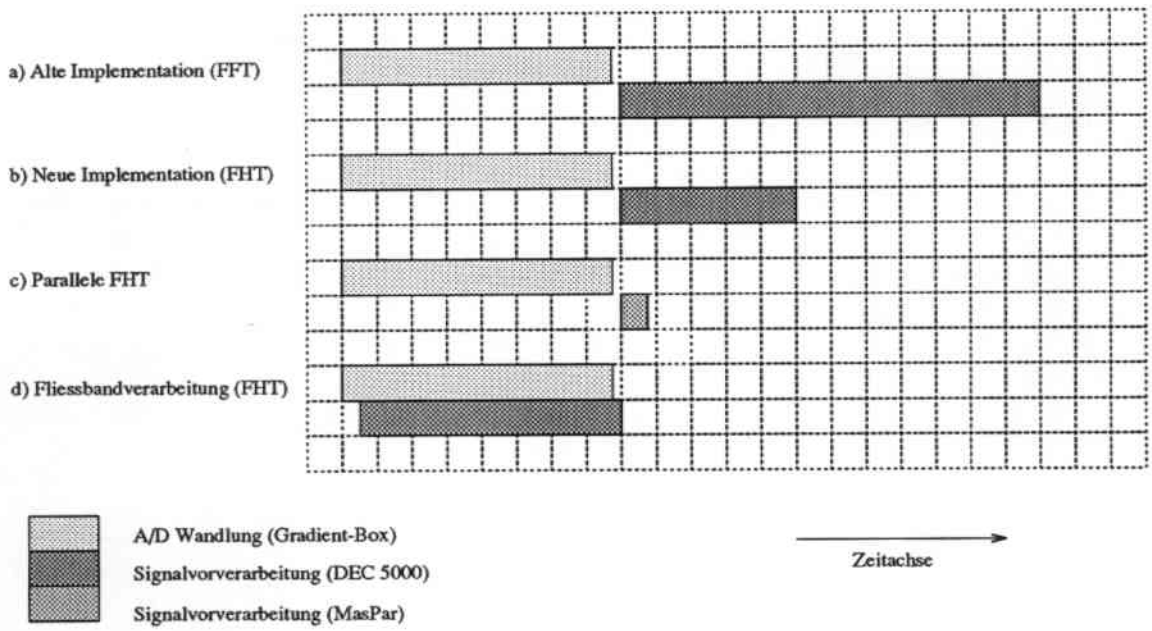

Abbildung 3.2: Abschätzung der verschiedenen Implementationsmöglichkeiten

3.2.2 Weitere Beschleunigung der Vorverarbeitung

Eine weitere Beschleunigung der Signalvorverarbeitung, kann man durch Parallelisierung der Transformationsfunktion[3] erreichen. Hierdurch wird der Zeitaufwand für die Transformation sehr klein (siehe Abbildung 3.2.c).

Eine Alternative zur weiteren Beschleunigung besteht in der Anwendung von Fließbandverarbeitung *(pipelining)*. Da in die digitale Signalvorverarbeitung immer Blöcke von jeweils 256 Werten einfließen, über die jeweils eine Transformation berechnet wird, kann man diese Blöcke unmittelbar nach ihrer Aufnahme durch das Aufnahmegerät an den Arbeitsplatzrechner übertragen und dort transformieren während bereits der nächste Block aufgenommen wird. So kann schon kurze Zeit nach Beenden der Sprachaufnahme die letzte Transformation abgeschlossen werden (siehe Abbildung 3.2.d). Anschließend können dann die restlichen Schritte der Vorverarbeitung erfolgen, die laut Ausführungsprofil weitaus weniger Zeit in Anspruch nehmen (siehe Tabelle 3.1, Seite 23). Auf diese Weise kann man wenige Millisekunden nach dem Ende der Sprachaufnahme die Vorverarbeitung beenden und die Merkmalsvektoren erhalten.

Die Parallelisierung der Transformationsfunktion lässt sich am besten durch Anwendung von *Teile und Herrsche* auf den rekursiven Algorithmus bewerkstelligen. Die Parallelisierung alleine würde vermutlich kaum schneller sein als die Fließbandverarbeitung in Kombination mit einem Arbeitsplatzrechner, da bei der Verwendung eines Parallelrechners die Aufnahmedaten hin und die Ergebnisse zurück übertragen werden müssten.

[3] Wie sich erst gegen Ende der Diplomarbeit herausstellte, ist inzwischen eine Bibliothek von der Firma MasPar erhältlich, die verschiedene Arten von ein- und zweidimensionalen FFTs beinhaltet. Dort ist es möglich, mehrere FFTs gleichzeitig und unabhängig voneinander zu berrechnen, pro Prozessorelement eine (der hochparallele Fall); des weiteren ist es möglich eine FFT auf allen Prozessorelementen zu berrechnen (minimale Wartezeit pro FFT); und schließlich ist es möglich, mehrere FFTs gleichzeitig und jeweils auf mehreren Prozessorelementen zu berechnen, um so den Speicher optimal auszunutzen (maximaler Durchsatz). Diese Bibliothek stand mir allerdings nicht zur Verfügung.

Die Fließbandverarbeitung kann auch mit der parallelen Transformation kombiniert werden. Dies wäre allerdings nicht sinnvoll, da man die Beschleunigung durch die parallele FHT nicht mehr bemerken würde (der sequentielle Algorithmus ist hierfür schnell genug). Außerdem würde man Zeit durch die Dateiübertragung von und zum Parallelrechner verschwenden.

Die Kombination zwischen der Fließbandverarbeitung und dem Berechnen der Signalvorverarbeitung auf einem Arbeitsplatzrechner ist hier die sinnvollste Lösung. Sie ist kostengünstiger da nur ein herkömmlicher Arbeitsplatzrechner zur Vorverarbeitung benötigt wird. Sie ist auch praktischer da sie das gleichzeitige Sammeln von Sprachdaten (inklusive der Vorverarbeitung) an mehreren Arbeitsplatzrechnern ermöglicht. Dieser Fall kommt häufig vor.

3.3 Implementation

Durch die Verwendung einer schnellen Hartley Transformation (FHT) anstelle einer schnellen Fourier Transformation (FFT) sowie durch die Einführung der genannten Nachschlagetabellen konnte eine Beschleunigung um den Faktor 2 erzielt werden. Die gesamte Signalvorverarbeitung für eine Spracheingabe der Länge 2,44 Sekunden braucht jetzt nur noch 1,6 Sekunden – also weniger Zeit als das Sprechen erfordert.

Damit wird bei zusätzlicher Fließbandverarbeitung die verbleibende Rechenzeit nach Beendigung der Spracheingabe sehr kurz. Die Fließbandverarbeitung zur Signalvorverarbeitung wird momentan im Rahmen einer Studienarbeit [38] implementiert und zeigt schon das erwartete Zeitverhalten.

Da die Kombination der Fließbandverarbeitung und der Vorverarbeitung auf einem Arbeitsplatzrechner praktischer als die parallele Implementation ist, wurde auf letztere verzichtet.

Ausführungsprofil makeFFT Revision 1.1 (insgesamt 8.0900 Sekunden)				
%time	seconds	cum %	cum sec	procedure (file)
50.7	4.10	50.7	4.10	fft (makeFFT.c)
31.0	2.51	81.7	6.61	sin (sincos.s)
4.9	0.40	86.7	7.01	cos (sincos.s)
4.4	0.36	91.1	7.37	main (makeFFT.c)
3.1	0.25	94.2	7.62	ham (makeFFT.c)
2.2	0.18	96.4	7.80	ptp_amp (makeFFT.c)
0.9	0.07	97.3	7.87	mkcoeff (makeFFT.c)
0.6	0.05	97.9	7.92	log (log.s)
0.6	0.05	98.5	7.97	read (../read.s)
0.5	0.04	99.0	8.01	fwrite (../fwrite.c)
0.2	0.02	99.3	8.03	fflush (../fclose.c)
0.2	0.02	99.5	8.05	write (../write.s)
0.2	0.02	99.8	8.07	bcopy (../bcopy.s)
0.1	0.01	99.9	8.08	write_float (makeFFT.c)
0.1	0.01	100.0	8.09	_flsbuf (../flsbuf.c)

Tabelle 3.1: Ausführungsprofil der ursprünglichen Signalvorverarbeitung

Ausführungsprofil makeFFT Revision 1.4 (insgesamt 4.4900 Sekunden)				
%time	seconds	cum %	cum sec	procedure (file)
72.4	3.25	72.4	3.25	fht (makeFFT.c)
8.0	0.36	80.4	3.61	fht_pow_spec (makeFFT.c)
4.5	0.20	84.9	3.81	ham (makeFFT.c)
3.6	0.16	88.4	3.97	ptp_amp (makeFFT.c)
3.6	0.16	92.0	4.13	main (makeFFT.c)
3.1	0.14	95.1	4.27	mkcoeff (makeFFT.c)
1.8	0.08	96.9	4.35	log (log.s)
1.1	0.05	98.0	4.40	read (../read.s)
0.7	0.03	98.7	4.43	fwrite (../fwrite.c)
0.4	0.02	99.1	4.45	bcopy (../bcopy.s)
0.4	0.02	99.6	4.47	write_float (makeFFT.c)
0.2	0.01	99.8	4.48	init_fht (makeFFT.c)
0.2	0.01	100.0	4.49	open (../open.s)

Tabelle 3.2: Ausführungsprofil der beschleunigten Signalvorverarbeitung

Kapitel 4

Analyse der Sprachsignale

Ziel der Sprachanalyse sind Bewertungen für jedes Phonem oder Phonemsegment[1], zu jedem Zeitpunkt. Diese Bewertungen werden als Kosten für die Phoneme zum jeweiligen Zeitpunkt betrachtet. Mit diesen Bewertungen wird später die DP–Matrix, die für die Suche verwendet wird, initialisiert (siehe Kapitel 5).

Zur Analyse der Sprachsignale können in Spracherkennern verschiedene Ansätze verfolgt werden:

- probabilistische Verfahren wie *Hidden Markov Models (HMMs)* [26, 11, 3, 15]
- neuronale Ansätze wie:
  - ▷ *Linked Predictive Neural Networks (LPNNs)* [39, 40, 41]
  - ▷ *Time Delayed Neural Networks (TDNNs)* [43]
  - ▷ *Multi Stage Time Delayed Neural Networks (MS-TDNNs)* [9, 10]
- gemischte Verfahren wie *Learning Vector Quanitization (LVQ)* [33, 20, 14]

Zu Beginn dieser Arbeit stand an der Universität Karlsruhe nur ein sprecherabhängiger LPNN–Erkenner (sowohl für englische, wie auch für deutsche Sprache) zur Verfügung. Mittlerweile existiert in Karlsruhe ein sprecherunabhängiger LVQ–Erkenner (ebenfalls für englische und deutsche Sprache), der darüber hinaus auch eine wesentlich bessere Erkennungsleistung hat [47]. Implementiert wurde hier ein paralleler LPNN–Erkenner.

Die Grundlagen zu LPNNs, die Einbettung und die Besonderheiten in JANUS, die Implementationsmöglichkeiten, sowie die Implementation werden im folgenden beschrieben. Möglichkeiten zur Implementation eines parallelen LVQ–Erkenners werden aufgezeigt.

[1] In JANUS wird jedes Phonem in mehrere Phonemsegmente unterteilt. Bei deutscher Sprache treten 46 Phoneme mit zusammen 136 Phonemsegmenten auf.

4.1 *Linked Predictive Neural Networks (LPNNs)*

4.1.1 Grundlagen

Linked Predictive Neural Networks (LPNNs) werden zur Vorhersage von Merkmalsvektoren verwendet. Für jedes Phonemsegment wird versucht für jeden Zeitpunkt der Spracheingabe den aktuellen Merkmalsverktor vorherzusagen. Man erhält so 136 vorhergesagte Vektoren pro Zeitpunkt. Für jeden Zeitpunkt sagt dabei jedes LPNN für genau ein Phonemsegment einen Vektor voraus. Die gesuchten Bewertungen für die Phonemsegmente für einen Zeitpunkt erhält man durch Berechnung der euklidischen Abstände zwischen den vorhergesagten Vektoren und dem tatsächlich aufgetretenen Merkmalsverktor.

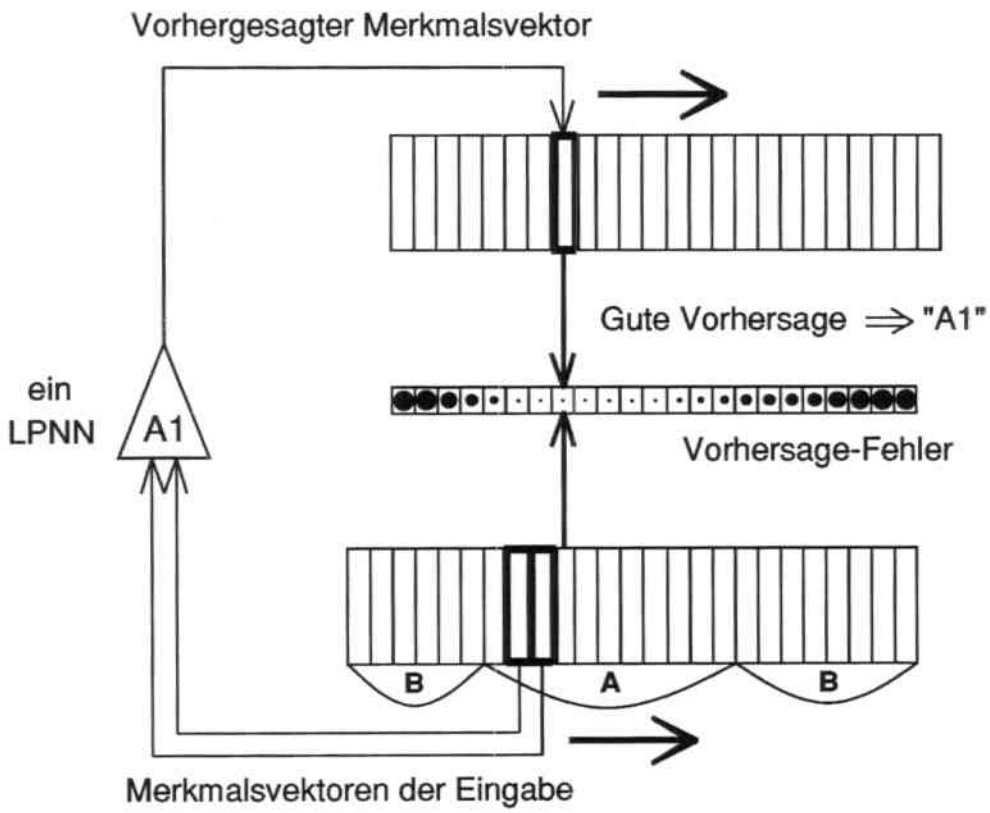

Abbildung 4.1: Berechnung der LPNN–Bewertungen

Die Abbildung 4.1 zeigt eines der LPNNs (symbolisiert durch ein Dreieck), das aus zwei zeitlich zurückliegenden Merkmalsvektoren den Merkmalsvektor für den aktuellen Zeitpunkt vorhersagt. Der euklidische Abstand ist der Vorhersage–Fehler für dieses LPNN "A1" in der Abbildung 4.1 durch die Größe der Punkte angedeutet. Die Größe der Eingabeschicht beträgt in diesem Beispiel 32 Neuronen (zwei zurückliegende Zeitpunkte mit jeweils 16 Koeffizienten der Merkmalsvektoren). Die Zwischenschicht hat 12 Neuronen und die Ausgabeschicht, die einen Merkmalsvektor erzeugt, enthält 16 Neuronen.

In JANUS ist die Topologie der neuronalen Netze variierbar. Sie wird durch eine Konfigurationsdatei festgelegt (*Netzwerk–Datei*, siehe Anhang A.2). Normalerweise werden neuronale Netze mit 64 Neuronen in der Eingabeschicht, 12 Neuronen in der Zwischenschicht und 16 Neuronen in der Ausgabeschicht verwendet. Dabei werden vier Merkmals-

vektoren als Eingabe für die erste Schicht von Neuronen verwendet, nämlich von zwei zeitlich zurückliegenden Zeitpunkten und von zwei zukünftigen Zeitpunkten. Der mittlere Zeitpunkt ist dabei der Vorherzusagende. Im Prinzip interpoliert solch ein LPNN. Diese Eingabekonfiguration ergab sich durch Experimente im Laufe der Entwicklung von JANUS.

4.1.2 Implementationsmöglichkeiten

Um die Vorwärtsberechnungen der neuronalen Netze zu parallelisieren, könnte man die Neuronen eines Netzes auf mehrere Prozessorelemente verteilen. Dies würde allerdings zu einem Kommunikationsaufwand an den zerschnittenen Kanten der neuronalen Netze führen. Eine solche Implementation wurde für vergleichsweise grosse neuronale Netze in [8] und [48] durchgeführt[2].

Andererseits kann man die neuronalen Netze so auf die Prozessorelemente verteilen, daß jeweils ein neuronales Netz auf einem Prozessorelement berechnet wird. Dabei werden keine Kanten der neuronalen Netze zerschnitten — es entsteht keine zusätzliche Kommunikation.

Allerdings treten bei deutscher Sprache 136, bei englischer Sprache nur 118 LPNNs auf. Um die MasPar trotzdem zu füllen, kann man diese parallelen LPNN–Berechnungen zusätzlich für mehrere Zeitabschnitte der Spracheingabe gleichzeitig ausführen.

Die Länge einer Spracheingabe beträgt zwischen einer halben Sekunde und zehn Sekunden. Wie in Kapitel 3 beschrieben, besteht die Eingabe aus Merkmalsvektoren mit einem zeitlichen Abstand von 10 Millisekunden. Die Eingabe der Sprachanalyse besteht also zwischen 50 und 1000 Merkmalsvektoren, für die von den 136 LPNNs Vorhersagen getroffen werden müssen. Insgesamt werden zwischen 6800 und 136000 Vorwärtsberechnungen und euklidische Abstände berechnet.

Ein hochparalleler Rechner kann im Fall, daß ein neuronales Netz pro Prozessorelement berechnet wird, bereits durch einen sehr kurzen Satz voll ausgelastet werden. Bei der MasPar MP-1 mit 16383 Prozessorelementen beträgt die entsprechende Satzlänge 1,2 Sekunden.

4.1.3 Implementation

Die wichtigsten, der Implementation zugrunde liegenden Parameter sind :

- Die Netzwerktopologie der LPNNs
- Die Anzahl der Phonemsegmente (LPNNs)
- Die Anzahl der Prozessorelemente

Für die Implementation wurde der zweite Ansatz gewählt, weil der Zeitaufwand für die Kommunikation bei Parallelrechnern erheblich ist und die Auslastung auch bei einem neuronalen Netz pro Prozessorelement gewährleistet ist.

[2]Die in [48] erreichte Leistung betrug dabei maximal 80 MCPS auf einer Connection Machine CM–2 mit 65536 Prozessorelementen.

Besonders berücksichtigt wurden bei der Implementation die effiziente Berechnung der neuronalen Netze und die Aufteilung der Prozessorelemente.

LPNN–Vorwärtsberechnungen

Abbildung 4.2 auf Seite 29 zeigt ein LPNN mit seinen Neuronen und deren Aktivierungen a, sowie den Verknüpfungen und den zugehörigen Gewichten w. Im folgenden bezeichnen hochgestellte Indizes den laufenden Index, so zum Beispiel die Nummer eines Neurons in einer der drei Schichten. Tiefgestellte Indizes bezeichnen die Schicht im Neuronalen Netz. (siehe Fußnote auf Seite 29).

Auf Seite 29 sieht man die Vortwärts–Berechnungen für ein LPNN. Die beiden Summenformeln (4.1) und (4.2) entsprechen den Berechnungen für die Verknüpfungen zwischen der Eingangs- und der Mittelschicht, beziehungsweise zwischen der Mittel- und der Ausgangsschicht eines dreischichtigen neuronalen Netzes. Die Summenformeln lassen sich mühelos als Vektor- und Matrix–Operationen interpretieren (Gleichungen (4.3) und (4.4)). Erstere als eine Multiplikation des Eingabevektors $\vec{a}_1$ mit der Gewichtsmatrix $\boldsymbol{W}_{12}$ und der folgenden Addition des Schwellwertvektors der Zwischenschicht $\vec{\theta}_2$ — auf den resultierenden Vektor wird komponentenweise die Sigmoid–Funktion angewendet; entsprechend auch die zweite Summenformel.

Die Neuronen der LPNNs und ihre Aktivierungen wurden im bisherigen sequentiellen Programmteil von JANUS in verketteten Listen abgespeichert. Für die parallele Berechnung der neuronalen Netze wird jetzt eine Vektor×Matrix Multiplikation verwendet, bei der Multiplikationen mit Null und mit Eins vermieden werden. Dies führt bei nur spärlich verbundenen neuronalen Netzen zu einer deutlichen Beschleunigung.

Da für den nächsten Bearbeitungschritt, die Suche der Satzhypothesen, nur noch Bewertungen benötigt werden, wurden zusätzlich zu den parallelen Vorwärtsberechnungen auch die euklidischen Abstände parallel berechnet.

Das Programm erhält folglich die FFT–Daten für den gesprochenen Satz und liefert die Bewertungen, entsprechend dem euklidischen Abstand zwischen Vorhersagen und Merkmalsvektoren für alle LPNNs und für alle Zeitpunkte der Spracheingabe.

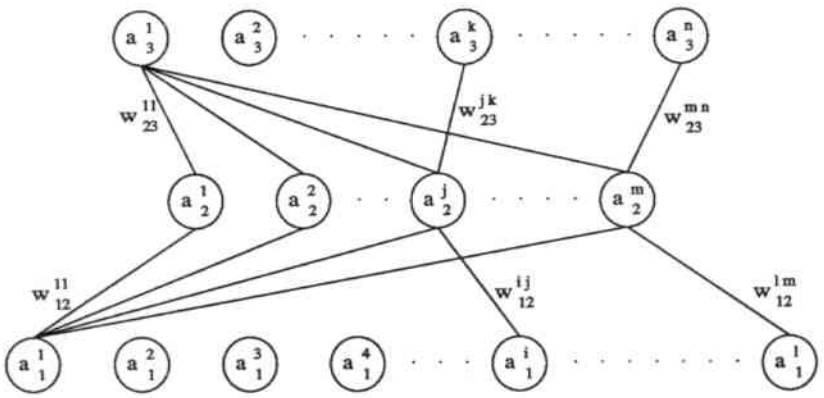

Abbildung 4.2: Ein LPNN mit Aktivierungen und Gewichten

Berechnet wird[3] :

$$a_2^j = sig(\sum_{i=1}^{m} a_1^i * w_{12}^{ij} + \theta_2^j) \tag{4.1}$$

und

$$a_3^k = sig(\sum_{j=1}^{n} a_2^j * w_{23}^{jk} + \theta_3^k) \tag{4.2}$$

mit der Sigmoid–Funktion :

$$sig(x) = \frac{1}{1 + e^{-x}}$$

Das entspricht :

$$\vec{a}_2^T = sig(\vec{a}_1^T * \boldsymbol{W}_{12} + \vec{\theta}_2^T) \tag{4.3}$$

und

$$\vec{a}_3^T = sig(\vec{a}_2^T * \boldsymbol{W}_{23} + \vec{\theta}_3^T) \tag{4.4}$$

wobei

$$\vec{a}_1^T * \boldsymbol{W}_{12} + \vec{\theta}_2^T = (a_1^1, \ldots, a_1^l) * \begin{bmatrix} w_{12}^{11} & w_{12}^{12} & w_{12}^{13} & \cdots & w_{12}^{1m} \\ w_{12}^{21} & w_{12}^{22} & w_{12}^{23} & \cdots & w_{12}^{2m} \\ w_{12}^{31} & w_{12}^{32} & w_{12}^{33} & \cdots & w_{12}^{3m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{12}^{l1} & w_{12}^{l2} & w_{12}^{l3} & \cdots & w_{12}^{lm} \end{bmatrix} + (\theta_2^1, \ldots, \theta_2^m)$$

und

$$\vec{a}_2^T * \boldsymbol{W}_{23} + \vec{\theta}_3^T = (a_2^1, \ldots, a_2^m) * \begin{bmatrix} w_{23}^{11} & w_{23}^{12} & w_{23}^{13} & \cdots & w_{23}^{1n} \\ w_{23}^{21} & w_{23}^{22} & w_{23}^{23} & \cdots & w_{23}^{2n} \\ w_{23}^{31} & w_{23}^{32} & w_{23}^{33} & \cdots & w_{23}^{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{23}^{m1} & w_{23}^{m2} & w_{23}^{m3} & \cdots & w_{23}^{mn} \end{bmatrix} + (\theta_3^1, \ldots, \theta_3^n)$$

[3] Im folgenden bezeichnen hochgestellte Indizes den laufenden Index. Tiefgestellte Indizes bezeichnen die Ebene im Neuronalen Netz; a_1^3 ist die Aktivierung des dritten Neurons in der Eingabeschicht N_1^3. w_{12}^{34} ist das Gewicht zwischen dem dritten Neuron in der Eingabeschicht N_1^3 und dem vierten Neuron in der in der Zwischenschicht N_2^4. W_{12} bezeichnet die Matrix mit den Gewichten zwischen der Eingabe- und der Zwischenschicht. $\vec{\theta}_2$ bezeichnet den Vektor der Schwellwerte der Zwischenschicht. Analog hierzu die Vektoren der Aktivierungen : $\vec{a}_1$, $\vec{a}_2$, $\vec{a}_3$.

Aufteilung der Prozessorelemente

Die Aufteilung der L LPNNs auf die P Prozessorelemente erfolgt bei der Programminitialisierung und ist sprachabhängig. Pro Prozessorelement wird ein LPNN berechnet. Bei deutscher Sprache werden 136 LPNNs ($\mathrm{LPNN}_0,\ldots,\mathrm{LPNN}_{135}$) so oft wie möglich auf die vorhandenen Prozessorelemente repliziert. Bei einer MasPar MP-1 mit 4096 Prozessorelementen werden 4080 Prozessorelemente verwendet. Das entspricht den kompletten Bewertungen für 30 Zeitpunkte der Spracheingabe, die gleichzeitig berechnet werden.

Von den 4096 Prozessorelementen $\mathrm{PE}_0,\ldots,\mathrm{PE}_{4095}$ berechnen die Prozessorelemente:

$$\begin{array}{lllll}
\mathrm{PE}_0, & \mathrm{PE}_{136}, & \mathrm{PE}_{272}, \ldots, \mathrm{PE}_{3944} & \text{die Zeitpunkte } \widetilde{t_0}, \widetilde{t_1}, \widetilde{t_2}, \ldots, \widetilde{t_{29}} & \text{für } \mathrm{LPNN}_0\ ; \\
\mathrm{PE}_1, & \mathrm{PE}_{137}, & \mathrm{PE}_{273}, \ldots, \mathrm{PE}_{3945} & \text{die Zeitpunkte } \widetilde{t_0}, \widetilde{t_1}, \widetilde{t_2}, \ldots, \widetilde{t_{29}} & \text{für } \mathrm{LPNN}_1\ ; \\
\vdots & \vdots & \vdots \quad \ldots, \quad \vdots & \vdots & \vdots \\
\mathrm{PE}_{135}, & \mathrm{PE}_{271}, & \mathrm{PE}_{407}, \ldots, \mathrm{PE}_{4079} & \text{die Zeitpunkte } \widetilde{t_0}, \widetilde{t_1}, \widetilde{t_2}, \ldots, \widetilde{t_{29}} & \text{für } \mathrm{LPNN}_{135}\ .
\end{array}$$

Hierbei sind $\widetilde{t_0}, \widetilde{t_1}, \widetilde{t_2}, \ldots, \widetilde{t_{29}}$ 30 Mengen von Zeitabschnitten, die folgende Zeitabschnitte der Sprachengabe enthalten:

$$\widetilde{t_0} = t_0, t_{30}, t_{60}, \ldots\ ;\ \widetilde{t_1} = t_1, t_{31}, t_{61}, \ldots\ ;\ \cdots\ ;\ \widetilde{t_{29}} = t_{29}, t_{59}, t_{79}, \ldots\ .$$

Die Prozessorelemente $\mathrm{PE}_{4080},\ldots,\mathrm{PE}_{4095}$ sind deaktiviert (das entspricht weniger als 0,4% der Prozessorelemente).

Ist dann eine Spracheingabe länger als (P div L) Zeitpunkte[4], dann werden die Ergebnisse pro Prozessorelement (pro LPNN) lokal gespeichert. Am Ende der Berechnungen für die momentane Spracheingabe werden die Ergebnisse parallel in eine Datei geschrieben. So wird auf die Ausgabedatei nur einmal pro Prozessorelement zugegriffen.

Bei dieser Implementation werden nur 8 KBytes Speicher pro Prozessorelement benötigt. Der überwiegende Teil wird dabei für die Vektoren und die Gewichtsmatrizen benötigt. Der vergleichsweise geringe Speicherplatz pro Prozessorelement bei der MasPar MP-1 stellte hier kein Problem dar.

4.1.4 Ergebnisse

Insgesamt wurde eine Beschleunigung um den Faktor 22 gegenüber der Berechnung auf einer DEC 5000 erreicht. Bei vollständig verbundenen neuronalen Netzen wurde eine maximale Leistung von 50 MCPS bei einer MasPar MP-1 mit 4096 Prozessorelementen erreicht. Da das Programm voll skalierbar ist, wird auf einer MasPar MP-1 mit 16384 Prozessorelementen eine maximale Leistung von etwa 200 MCPS möglich sein. Diese Spitzenleistung wird nur bei Sätzen erreicht, deren Länge durch die Anzahl der gleichzeitig berechneten Zeitpunkte teilbar ist. Im ungünstigsten Fall werden im letzten Durchlauf nur die Berechnungen für einen einzigen Zeitpunkt durchgeführt. Die erreichte mittlere Leistung von 41,4 MCPS liegt nur knapp unter dem Spitzenwert.

Ein Vergleich zu einer MIMD–Implementation der LPNNs befindet sich im Kapitel 6.

[4] Bei deutscher Sprache 30 Zeitpunkte.

4.2 *Learning Vector Quantization (LVQ)*

Hier wird kurz auf einen neuen, sprecherunabhängigen Ansatz zur Spracherkennung eingegangen, der erst gegen Ende dieser Arbeit vorgestellt wurde und bessere Erkennungsleistungen als der LPNN–Erkenner erreicht.

Der auf *Learning Vector Quantization (LVQ)* [14, 21] basierende Erkenner [33] verwendet ebenfalls ein Phonemmodell mit 6 Zuständen und 3 Phonemsegmenten. Die Besonderheit bei diesem gemischten LVQ-HMM–Ansatz ist, daß man ihn sowohl als neuronalen wie auch als statistischen Ansatz erklären und berechnen kann (siehe [34, 33]). Das Phonemmodell wird dabei als *Hidden Markov Model (HMM)* [26] interpretiert. Es werden die gleichen Merkmalsvektoren mit 16 Koeffizienten wie beim LPNN–Erkenner verwendet.

4.2.1 Grundlagen

Im folgenden werden der Aufbau und die Berechnungen für den LVQ–Erkenner skizziert[5]:

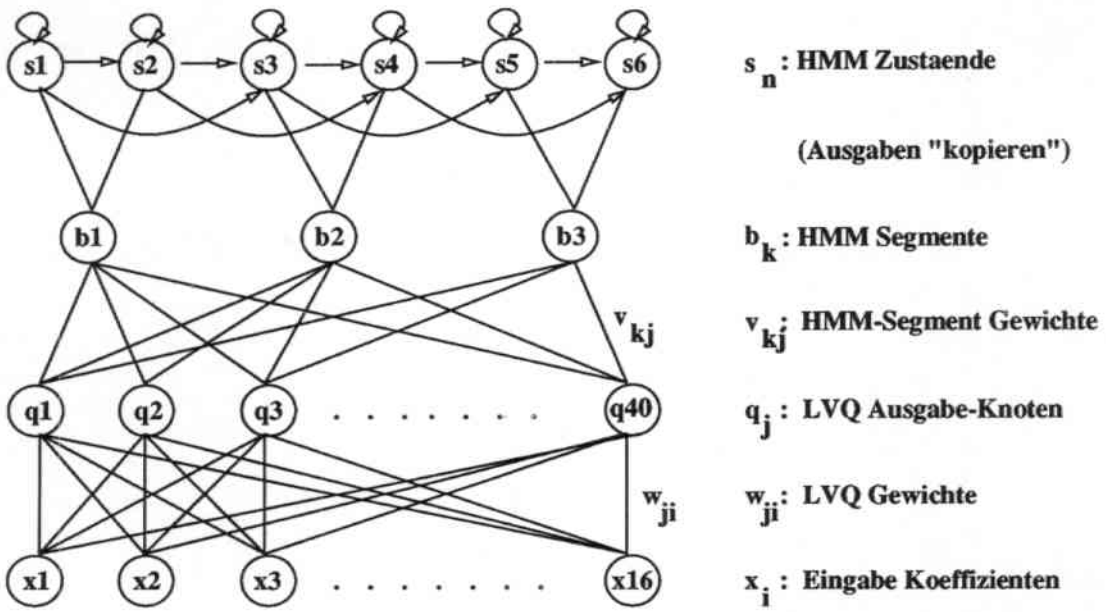

Abbildung 4.3: Ein LVQ–Netz mit Aktivierungen, Gewichten und HMM

Pro Phonem existieren zwei voneinander unabhängige LVQ–Netze:

Für jeden Zeitpunkt t der Spracheingabe bekommt das erste LVQ–Netz den Merkmalsvektor des Zeitpunkts t, das zweite bekommt die Differenz zwischen den zwei Merkmalsvektoren der Zeitpunkte t–2 und t+2 als Eingabe. Das zweite LVQ–Netz erhält so Informationen über die Dynamik der Spracheingabe.

Ein LVQ–Netz umfaßt drei Schichten und ein an die Ausgabeschicht angekoppeltes HMM. Die Eingabeschicht umfaßt 16 Knoten, die mittlere Schicht umfaßt etwa 40 Knoten und die Ausgabeschicht umfaßt 3 Knoten.

[5] Die oberste Schicht in der Abbildung 4.3 wird für die folgende Erklärung der Vorwärtsberechnungen **nicht** benötigt! (Diese Schicht zeigt das Phonemmodell, daß hier als HMM bezeichnet wird.)

Die Vorwärtsberechnungen erfolgen entsprechend dem LVQ2–Algorithmus [20, 22]: Zwischen der Eingabe- und der Zwischenschicht werden keine Gewichte benutzt, sondern 40 Referenzvektoren. Zwischen diesen 40 Referenzvektoren und dem Eingabevektor werden die Abstände berechnet. Es wird nur der Knoten mit dem kleinsten Abstand aktiviert. Zwischen der mittleren und der letzten Schicht gibt es Gewichte. Der aktivierte Knoten wird mit den drei der zu ihm gehörenden Gewichte multipliziert — man erhält so je 3 Zwischenergebnisse pro LVQ–Netz.

Da pro Phonem zwei LVQ–Netze berechnet werden, stehen zwei mal drei Zwischenergebnisse zur Verfügung. Je zwei davon werden gewichtet summiert, so daß insgesamt drei Ausgaben entstehen. Sie stellen die gesuchten Bewertungen für die Phonemsegmente dar.

Größenvergleich zwischen LVQs und LPNNs

Beim LVQ–Erkenner werden drei Bewertungen pro Phonem durch ein Paar von LVQ–Netzen errechnet. Bei deutscher Sprache gibt es insgesamt 92 LVQ–Netze. Beim LPNN–Erkenner werden drei Berwertungen pro Phonem durch drei getrennte LPNNs errechnet. Hier gibt es insgesamt 136 LPNNs.

Ein LVQ–Netz ist etwas größer als ein LPNN. Dafür sind allerdings die Vorwärtsberechnungen beim LVQ–Erkenner weniger umfangreich.

4.2.2 Implementationsmöglichkeiten

Ein Verteilen der Knoten der LVQ–Netze auf verschiedene Prozessorelemente würde, wie auch bei LPNNs, zu starker Kommunikation führen.

Man kann auch die LVQ–Netze entweder einzeln oder paarweise auf Prozessorelemente verteilen und dort parallel berechnen. Um den SIMD–Rechner zu füllen kann man (wie bei den LPNNs) identische Netze replizieren und so die Berechnungen für verschiedene Zeitpunkte der Spracheingabe parallelisieren.

Verteilt man immer zwei der 92 LVQ–Netze auf ein Prozessorelement kann man auf einer MasPar MP-1 mit 4096 Prozessorelementen 89 Zeitabschnitte gleichzeitig berechnen. Verteilt man je ein LVQ–Netz auf ein Prozessorelement kann man 44 Zeitabschnitte gleichzeitig berechnen. Das hat in erster Linie Auswirkungen auf die Kurve der Rechenzeit über der Dauer der Spracheingabe. Diese verläuft bei beiden Varianten treppenförmig. Bei einem LVQ–Netz pro Prozessorelement fallen die Stufen allerdings kleiner aus — es liegen im Mittel weniger Prozessorelemente brach.

Die abschließenden Verknüpfungen der Paare von Zwischenergebnissen der LVQ–Netze kann man durch eine X–Net Kommunikation bewerkstelligen. Dabei greifen alle Prozessorelemente gleichzeitig auf den Nachbarprozessor zu. Die Hälfte der Prozessorelemente berechnet dann die gewichtete Summe. Diese Kommunikation ist etwa viermal so schnell wie eine Gleitkommamultiplikation. Diese eine Kommunikationsoperation pro Vorwärtsberechnung fällt nicht stark in's Gewicht.

Es ist zu erwarten, daß durch ein Verteilen von je einem LVQ–Netz auf ein Prozessorelement die größte Beschleunigung zu erzielen ist. Dabei werden die Prozessorelemente für fast alle Zeitabschnitte der Spracheingabe voll ausgelastet. Die Beschleunigung wird daher vermutlich ähnlich wie bei den parallelen LPNN–Vorwärtsberechnungen ausfallen.

Kapitel 5

Suche der Satzhypothesen

Im folgenden wird eine Einführung in die Suche nach Satzhypothesen bei der Spracherkennung gegeben. Anschließend werden Besonderheiten von JANUS erläutert, die zu starken Einschränkungen der Implementationsmöglichkeiten führen. Im Anschluß daran werden die Probleme und Überlegungen bei der Implementation besprochen und die tatsächliche Implementation beschrieben.

5.1 *Dynamic Time Warping (DTW)*

5.1.1 DTW für Einzelworterkennung

Unter *Dynamic Time Warping (DTW)* oder auch *Dynamic Programming (DP)*, dynamischem Programmieren, versteht man im allgemeinen die Längenanpassung zweier unterschiedlich langer Muster. Dabei wird der optimale Pfad zwischen einer Eingabe und einem Muster gesucht. Solch eine Anpassung ist notwendig, wenn man zeitlich verzerrte Muster miteinander vergleichen möchte. Ziel dabei ist, für jedes Referenzmuster eine Bewertung zu finden, die angibt wie gut es zur Eingabe paßt.

In früheren Ansätzen für die Spracherkennung [27] bediente man sich dieser Technik, um die gesprochene Sprache mit festen Referenzmustern zu vergleichen.

In Abbildung 5.1 auf Seite 34 sieht man die Spracheingabe entlang der x–Achse (Zeitachse) sowie ein Referenzmuster entlang der y–Achse (Vokabularachse). Die dazwischen aufgespannte Matrix nennt man *DP–Matrix*. Sie wird Punkt für Punkt mit den euklidischen Abständen zwischen dem Muster und der Eingabe gefüllt. Dann wird für alle Zeitpunkte der Eingabe nacheinander, also Spaltenweise von links nach rechts und in jeder Spalte von unten nach oben, für jeden Punkt der billigste Vorgänger gesucht. Dabei sucht man an (vorher fest vorgegebenen) relativen Koordinaten und addiert den Wert des billigsten Vorgängers zum Wert des aktuellen Punktes. Da versucht wird die Eingabe an ein Referenzmuster anzupassen, muß der optimale Pfad in der DP–Matrix links unten beginnen und rechts oben enden. Man erhält so in der rechten oberen Ecke eine Gesamtbewertung, die um so kleiner ist, je besser die Eingabe zum Referenzmuster paßt. Die relativen Koordinaten werden durch die sogenannte DP–Funktion festgelegt.

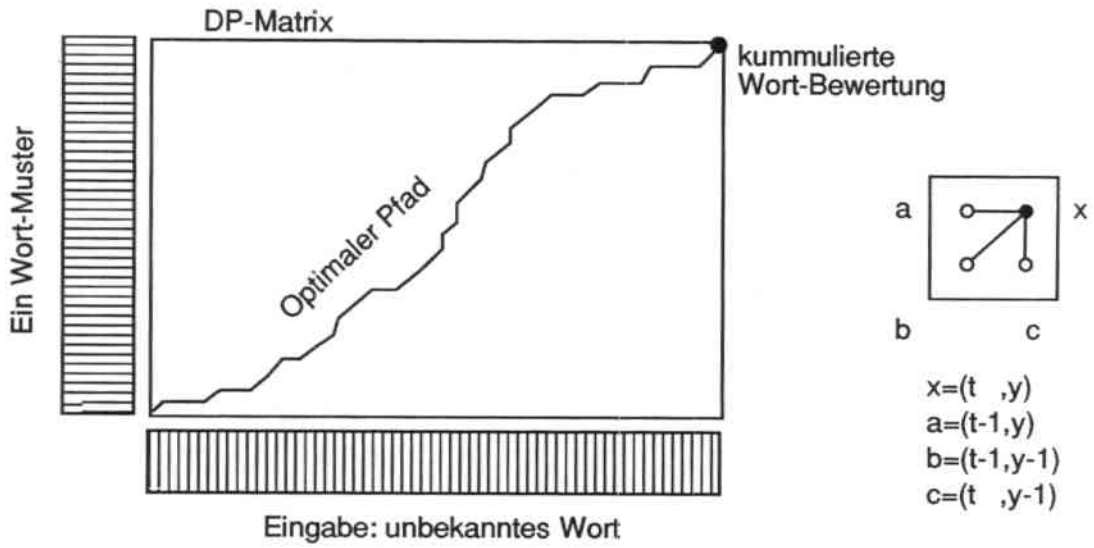

Abbildung 5.1: Einfaches DTW – Zeitanpassung zweier Muster

Die in Abbildung 5.1 gezeige DP–Funktion lautet zum Beispiel :

$$S(x) = B(x) + \min\{S(a), S(b), S(c)\}$$

Hierbei ist $B(x)$ die Bewertung für den Punkt x und $S(x)$ ist die kumulierte Summe am Punkt x. Diese DP–Funktion greift symmetrisch um die Diagonale auf benachbarte Werte zu — man nennt solche DP–Funktionen daher *symmetrisch.*

Der wiederholte Zugriff auf den Ast zu Punkt c kann allerdings zu einem Pfad führen, der senkrecht in der DP–Matrix verläuft. Dies bedeutet aber, daß ein kleiner Zeitabschnitt der Eingabe auf das gesamte Muster abgebildet werden kann. Folglich kann das Muster in diesem Fall nicht richtig angepaßt werden. Aus diesem Grund verwendet man oft keine symmetrische DP–Funktion, sondern eine *asymmetrische.* Würde man im Beispiel den Ast zum Punkt c weglassen, erhielte man die neue, asymmetrische DP–Funktion:

$$S(x) = B(x) + \min\{S(a), S(b)\}$$

DP–Funktionen können darüber hinaus auch Strafen für die einzelnen Äste definieren:

$$S(x) = B(x) + \min\{S(a) + p_{a\to x}\,,\, S(b) + p_{b\to x}\}$$

Wobei $p_{a\to x}$ die Übergangsstrafe von Punkt a zum Punkt x ist; entsprechend ist $p_{b\to x}$ die Übergangsstrafe von b zu x. Hierdurch kann eine Richtung in der DP–Matrix bevorzugt werden.

In [31] finden sich weitere Überlegungen zu möglichen DP–Funktionen.

In einem musterbasierten Einzelworterkenner würde man Referenzmuster für alle Worte im Vokabular speichern und mittels dynamischen Programmierens eine Bewertung für jedes dieser Muster ermitteln. Die so gefundenen Bewertungen könnte man dann miteinander vergleichen und dasjenige mit den geringsten Kosten als erkannte Worthypothese für die Spracheingabe betrachten.

Solche musterbasierten Ansätze haben unter anderem den Nachteil, daß man viele lange Referenzmuster einsprechen und speichern muß. Andere Ansätze wie HMM-Erkenner und

auf neuronalen Netzen basierte Erkenner verwenden Bewertungen für Phoneme oder Teile von Phonemen. Die Anzahl der verschiedenen Phoneme ist im allgemeinen weit geringer als die Anzahl der Wörter des Vokabulars. Es werden also weniger und kürzere Einheiten zur Erkennung herangezogen. Im folgenden werden nur noch solche Ansätze beschrieben.

Der bisher beschriebene einfache DTW–Algorithmus hat außerdem den Nachteil, daß das Eingabe–Wort zuerst aus dem gesprochenen Satz isoliert werden muß. Das gelingt allerdings nur zuverlässig bei abgehackt gesprochener Sprache.

5.1.2 DTW für kontinuierlich gesprochene Sprache

Bei kontinuierlich gesprochener Sprache treten keine künstlichen Pausen zwischen den Wörtern auf. Hier ist es nicht möglich Teile der Spracheingabe zu isolieren um sie dann mit Referenzmustern zu vergleichen.

Um kontinuierlich gesprochene Sprache verarbeiten zu können muß eine Möglichkeit gefunden werden, die einzelnen Wörter der Spracheingabe trotzdem zu erkennen und zu trennen. Hierzu wurden bereits einige Algorithmen vorgestellt [30, 23].

Zur Verdeutlichung der Suche wird hier der *one–stage* Algorithmus [23] detaillierter beschrieben (siehe Abbildung 5.2 auf Seite 36). Mit diesem Algorithmus wird die Satzhypothese[1] mit der besten Gesamtbewertung für den kontinuierlich gesprochenen Eingabe-Satz gefunden.

Zur Vereinfachung werden in diesem Beispiel die Buchstaben der Wörter als Phoneme betrachtet und es wird eine einfache DP–Funktion ohne Strafen verwendet.

Die DP–Matrix enthält auf der y–Achse alle Wörter des Vokabulars in ihrer Phonemdarstellung (hier Buchstaben). Für jeden Zeitpunkt der Spracheingabe wurden während der Sprachanalyse (Kapitel 4) Bewertungen für die einzelnen Phoneme berechnet. Mit diesen Bewertungen wird die DP–Matrix initialisiert.

Die Wörter des Vokabulars können nur mit ihrem ersten Phonem anfangen und mit ihrem letzten Phonem enden. Daher werden alle anderen Phonem–Bewertungen in der DP–Spalte $t = 0$ mit einem Wert nahe Unendlich initialisiert. Da der optimale Pfad derjenige mit der minimalen kumulierten Summe ist, wird so verhindert, daß ein Pfad woanders als am Anfang eines Wortes anfängt.

Analog hierzu muß am Ende der DTW–Berechnungen auch nur bei jeweils dem letzten Phonem eines Wortes nach Kandidaten für den optimalen Pfad gesucht werden.

Es wird nun, ähnlich wie beim musterbasierten DTW, spaltenweise ab der Spalte $t = 1$ Phonem für Phonem nach dem günstigsten Vorgänger gesucht — entsprechend der gegebenen DP–Funktion. Zum Beispiel für das Phonem N im Wort EBEN (vgl. Abbildung 5.2) kommen N und E als Vorgänger in Betracht. Anders beim Phonem E am Anfang des Wortes: hier muß bei der Suche nach dem kostengünstigsten Vorgänger–Phonem über eine Wortgrenze hinweg gesucht werden (vgl. Abbildung 5.2). Anstelle jetzt nur beim Phonem E des eigenen Wortes zu suchen, müssen zusätzlich alle End–Phoneme aller Wörter des Vokabulars (DAS, IST, SO, EBEN) in Betracht gezogen werden. An einem Wortübergang

[1] Eine Folge von Wörtern des Vokabulars.

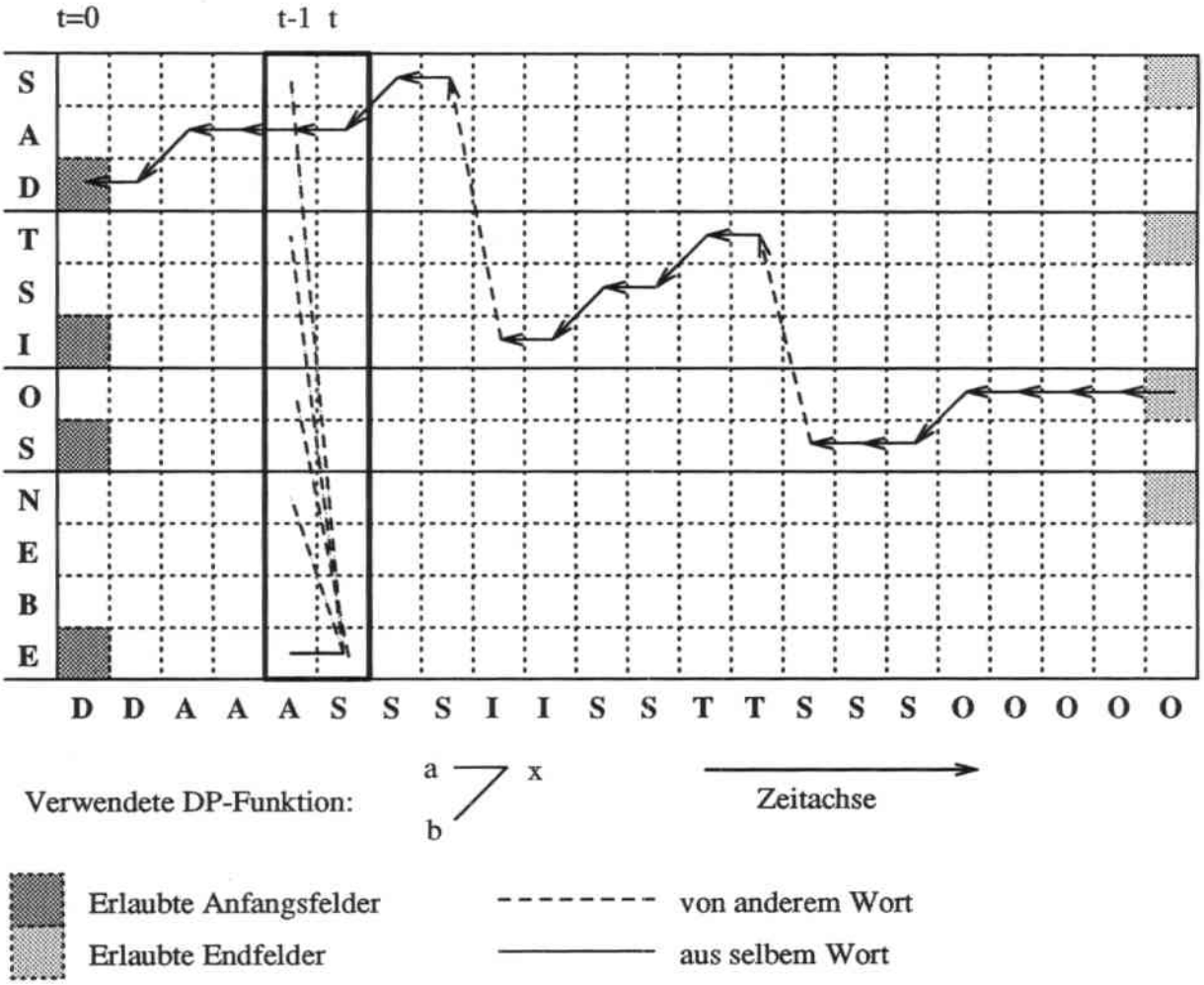

Abbildung 5.2: Der *one–stage* Algorithmus

müssen im Beispiel die vier End–Phoneme S, T, O und N im Vokabular und das Anfangs–Phonem im momentanen Wort (entsprechend des Astes $a \rightarrow x$ der DP–Funktion) betrachtet werden. Solch ein Wortübergang ist hier für eine Stelle der DP–Matrix durch gestrichelte Linien angedeutet (vgl. Abbildung 5.2).

Es treten also zwei Arten von DP–Funktionen auf, eine innerhalb der Wörter und eine an Wortübergängen.

Für jeden Zeitpunkt (jede Spalte in der DP–Matrix) müssen alle wortinternen Übergänge und zusätzlich an den Wortanfängen auch alle Übergänge von allen Wörtern des Vokabulars zu allen Wörtern des Vokabulars untersucht werden. Dieser quadratische Aufwand ist der Grund für die Verwendung von relativ kleinen Vokabularen bei der Erkennung kontinuierlich gesprochener Sprache.

Die Suche nach dem besten Vorgänger erfolgt spaltenweise, also sequentiell entlang der Zeitachse. Die Berechnungen für einen Zeitpunkt t finden also nur im Bereich der Spalten $t-1$ und t statt. Bei der Berechnung der Spalte t muß die Spalte $t-1$ bereits vollständig berechnet sein um Wortübergänge zu ermöglichen[2].

[2]Im *one–stage* Algorithmus können deshalb nur DP–Funktionen verwendet werden, die ausschließlich auf zeitlich zurückliegende Felder zugreifen. Außerdem kann das in der Literatur zu parallelen Algorithmen

In jedem Feld der DP–Matrix wird der Weg zum günstigsten Vorgänger gespeichert. Wenn man so für jeden Punkt der DP–Matrix einen Rückwärtszeiger zum günstigsten Vorgänger berechnet hat, kann man aus diesen Rückwärtszeigern den optimalen Pfad gewinnen.

Das letzte Wort der Satzhypothese findet man, indem man die kumulierten Summen für die End–Phoneme der Wörter des Vokabulars in der hintersten DP–Spalte vergleicht (die hell schraffierten Felder in der Abbildung 5.2). Das Wort mit der geringsten Summe ist das End–Wort der Satzhypothese. Die gesamte Satzhypothese bekommt man, indem man die Rückwärtszeiger bis zur DP–Spalte $t = 0$ zurückverfolgt.

Hier raus ergibt sich automatisch die Trennung der gesprochenen Spracheingabe: Eine Wortgrenze tritt genau dort auf, wo ein Wortübergang der billigste Weg zum vorangehenden Feld in der DP–Matrix ist (siehe Abbildung 5.2).

In Abbildung 5.3 kann man den optimalen Pfad (als helle Linie) erkennen. In der DP–Matrix sind hier die kleinsten Abstände zum Vorgängerfeld als Graustufen eingetragen. Helle Felder entsprechen einem kleinen Abstand. Um die Grafik zu erstellen wird zuerst die normale Erkennung durchgeführt. Der erkannte Satz wird dann nachträglich noch einmal in seiner Phonemdarstellung auf der y–Achse eines musterbasierten DTW aufgetragen. Der gesamte erkannte Satz wird als Referenzmuster aufgefaßt und ein dazu passender optimaler Pfad berechnet. Den hohen Rechenaufwand beim *one–stage* Algorithmus kann man im sequentiellen Fall durch Strahlsuche beschneiden. Auch durch das Einführen von Listen erlaubter Vorgänger- beziehungsweise Nachfolgewörter kann man den Aufwand an den Wortübergängen veringern. Trotzdem bleibt der Algorithmus sehr rechenzeitaufwendig.

Der Speicheraufwand für die DP–Matrix ist beträchtlich, wenn man pro Matrixelement eine kumulierte Summe und den Rückwärtszeiger speichert. Die bisher kumulierte Summe wird aber nur in den zwei jeweils zuletzt berechneten Spalten der DP–Matrix benötigt. Wenn man die kumulierten Summen nur in den beiden für die momentanen Berechnungen nötigen DP–Spalten speichert und außerdem nur in den Wortanfangs–Zeilen die Rückwärtszeiger speichert, kann man den Speicherbedarf erheblich reduzieren.

Den *one–stage* Algorithmus für die beste Satzhypothese nennt man auch *first best*, den für die Suche der N besten Satzhypothesen nennt man auch *N best* Algorithmus. Der einfacheren Erklärung wegen wird auch in der folgenden Beschreibung der Suche der N besten Satzhypothesen von einer mit Bewertungen gefüllten DP–Matrix ausgegangen.

5.1.3 Suche der N besten Satzhypothesen

Es reicht im allgemeinen nicht aus, nur die beste Satzhypothese für eine Spracheingabe zu finden. Wenn die Sprachaufnahme akustische Fehler (zum Beispiel Hintergrundgeräusche) aufweist kann das dazu führen, daß die am besten bewertete Satzhypothese ungrammatikalisch ist. Die Satzhypothese kann dann in späteren Verarbeitungsschritten nicht zerteilt werden. Um solch eine fehlerbehaftete Aufnahme trotzdem bearbeiten zu können, hilft es wenn die N besten Satzhypothesen zur Verfügung stehen. Die richtige Satzhypothese befindet sich dann oft unter den weniger gut bewerteten.

Der beschriebene *one–stage* Algorithmus läßt sich für die Suche von N besten Satzhypothesen ausbauen [37].

---

vorkommende „Diagonalisierungs–Verfahren“ zur DP–Parallelisierung auf SIMD–Rechnern daher beim *one–stage* Algorithmus nicht angewendet werden.

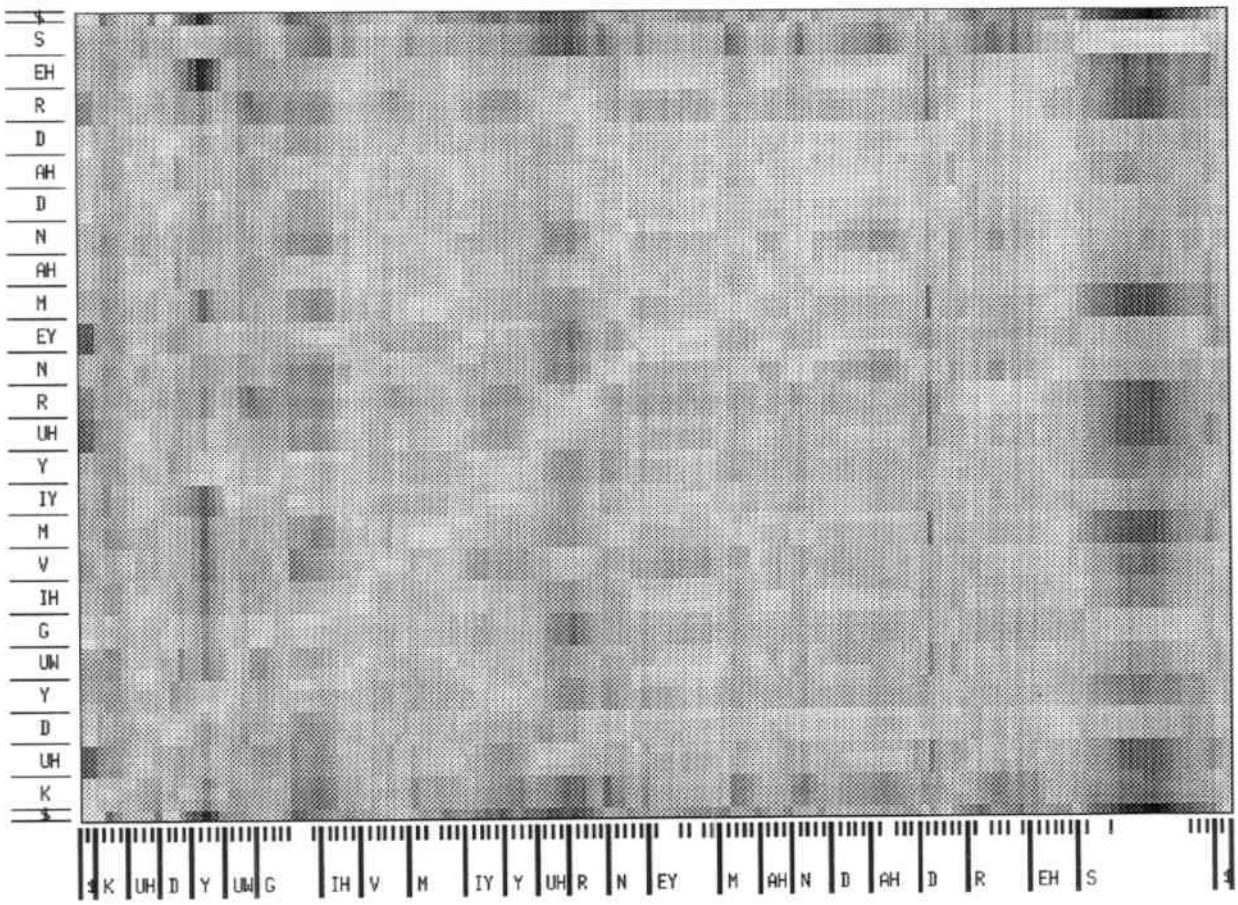

Abbildung 5.3: Optimaler Pfad für einen englischen Satz

An jedem Punkt der DP–Matrix werden N Rückwärtszeiger und N kumulierte Summen gespeichert. Die jeweils erste Summe in jedem Feld wird mit der jeweiligen Phonembewertung für den jeweiligen Zeitpunkt initialisiert; die restlichen $N-1$ Summen pro Feld werden mit einem Wert nahe Unendlich initialisiert.

Wie beim *first best* Algorithmus werden in der DP–Spalte $t = 0$ alle ersten Summen die nicht zu einem Anfangs–Phonem gehören mit einem Wert nahe Unendlich initialisiert.

Die Vorgehensweise ist nun ähnlich wie beim *first best* Algorithmus, nur daß anstelle den günstigsten Vorgänger für ein Feld zu suchen, die N–besten–Listen mit den kumulierten Summen und Rückwärtszeigern in eine N–besten–Liste gemischt werden. Die darin verbleibenden Übergänge sind die N günstigsten für dieses Feld.

Möchte man aus den in der letzten DP–Spalte pro Wort resultierenden N–besten Listen die N besten Satzhypothesen gewinnen, dann müssen dort die insgesamt N besten kumulierten Summen gesucht werden. Durch Rückverfolgen der Rückwärtszeiger erhält man anschließend die N besten Satzhypothesen.

Der Speicheraufwand läßt sich hier wiederum reduzieren, indem man nur in den beiden für die Berechnung momentan notwendigen DP–Spalten die N kumulierten Summen und die N Rückwärtszeiger speichert. Außerdem muß man pro Wortanfangs–Zeile in der DP–Matrix die N Rückwärtszeiger pro Feld speichern. Die kumulierten Summen werden bei der Rückverfolgung der Zeiger nur in der letzten DP–Spalte gebraucht.

Wie beim *first best* Algorithmus, läßt sich auch hier durch Verwendung von Strahlsuche und durch Listen von gültigen Vorgängerwörtern eine Beschleunigung erzielen. Überlegungen zu weiteren Verbesserungen für die N–besten–Suche auf sequentiellen Rechnern sind in [35, 2] beschrieben.

5.2 DTW in JANUS

Es werden einige Besonderheiten von JANUS erläutert, die zu starken Einschränkungen der Implementationsmöglichkeiten führen.

5.2.1 Beschneiden des Suchbaumes

Wortpaar–Grammatik und Bigramme

Oft sind bei einem Spracherkenner ein Vokabular und eine Menge von Beispielsätzen gegeben. Die Menge der gültigen Nachfolgewörter für jede Stelle der Beispielsätze kann man leicht berechnen. Damit kann man im *one–stage* Algorithmus den Aufwand an den Wortübergängen beschneiden. Möchte man nur Sätze erkennen, die den Beispielsätzen ähnlich sind, reicht es aus an Wortübergängen nur noch die gültigen Nachfolgewörter in Betracht zu ziehen.

In JANUS kann die Anzahl der Wortübergänge durch Verwendung von Wortpaaren oder Bigrammen [12] eingeschränkt werden. Das führt zu besseren Erkennungsleistungen und zu geringeren Rechenzeiten. Beim parallelen Algorithmus soll die Möglichkeit offengehalten werden diesen Mechanismus später zu implementieren.

Strahlsuche

Im sequentiellen *first best* DTW–Algorithmus von JANUS kann Strahlsuche verwendet werden. Das heißt, daß Zustandsübergänge nur dann erfolgen, wenn die Differenz zwischen der entstehenden kumulierterten Bewertung und dem bisherigen besten Pfad kleiner als ein vorgegebener fester Wert ist. Bei einer einfach ausgelasteten SIMD–Maschine würde das Beschneiden der Pfade zu einem Abschalten von Prozessorelementen führen. Hierdurch kann man keine Beschleunigung erreichen. Die Strahlsuche bringt in diesem Fall keine Verbesserung.

5.2.2 Phonemmodelle

Eine Besonderheit an JANUS ist die Verwendung von Phonemmodellen, ähnlich *Hidden Markov Models (HMMs)* [26].

Die Phonemmodelle dienen dazu, die Variabilität der Phoneme im Sprachkontext zu modellieren: Anfang und Ende eines Phonems sind stärker von den benachbarten Phonemen abhängig als der mittlere Teil. Es ist deshalb sinnvoll, das Phonem in mehrere Abschnitte mit eigenen Bewertungen zu unterteilen.

Die durchschnittliche Länge eines Phonemes beträgt etwa 60 Millisekunden. Da ein Merkmalsvektor auf der Zeitachse genau einem Zeitabschnitt von 10 Millisekunden entspricht,

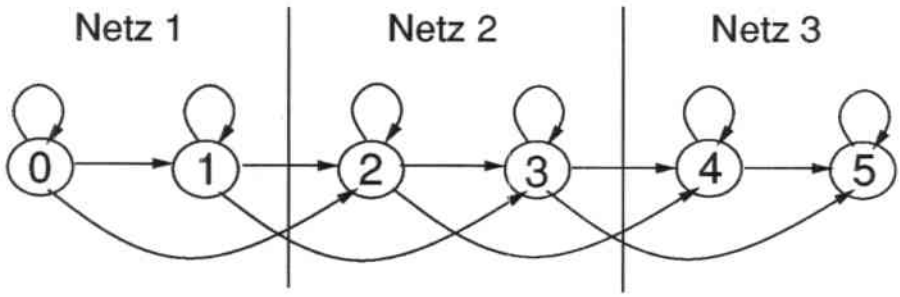

Abbildung 5.4: Phonemmodell mit 6 Zuständen

wird meist eine Unterteilung in sechs Zustände gewählt, so daß ein Phonem im Mittel auf der Zeit- und Vokabularachse gleichviele Zustände hat. Da nicht jedes vorkommende Phonem genau 60 Millisekunden lang ist, muß es möglich sein, Phonemzustände zu überspringen oder in einem Phonemszustand zu verweilen (entsprechend einem steileren oder flacheren Pfad in der DP-Matrix). Dabei kann man die Übergänge auch mit Übergangsstrafen versehen, um zum Beispiel das zu lange Verweilen in einem Phonemzustand zu verhindern.

Im allgemeinen reichen die vorhanden Trainingsdaten nicht, um für die selteneren Phoneme sechs verschiedene neuronale Netze zu trainieren — die Anzahl der Trainingsvektoren pro Phonemabschnitt wäre zu klein. Deshalb werden je zwei Zustände zu einem *Phonemsegment* zusammengefasst, und erhalten beide dieselbe Bewertung.

Die Zuordnung der Modelle zu den Phonemen geschieht, wie auch die gesamte Definition der Modelle und Phoneme, in der JANUS–Konfigurationsdatei *modelfile*. (Siehe auch Anhang A.1). Jedes Phonem kann im Prinzip einzeln modelliert werden.

Das Phonem „Stille“ nimmt eine Sonderrolle unter den Phonemen des Vokabulars ein: Es wird im allgemeinen durch ein Phonemmodell mit nur zwei Zuständen modelliert. Außerdem gibt es bei JANUS das Wort „Stille“. In JANUS müssen gültige Satzhypothesen mit „Stille“ anfangen und enden — dies führt zu besseren Erkennungsleistungen.

5.2.3 Phonemmodelle und Vokabularachse

In JANUS werden auf der Vokabularachse keine Wortmuster verwendet, sondern Wörter die aus aneinandergehängten Phonemmodellen bestehen. Jedem Phonemsegment wird für jeden Zeitpunkt in der Eingabe eine Bewertung zugeordnet werden (Kapitel 4).

Stellt man sich ein einfaches Vokabular mit dem einen Wort *Testwort* und einem Phonemmodell mit sechs Zuständen vor (siehe Abbildung 5.4), dann sind auf der Vokabularachse die 48 Phonemzustände aufgereiht. Je zwei der Zustände entsprechen einem Phonemsegment. In die DP–Matrix werden dann, wie beschrieben, die Bewertungen für jeden Phonemzustand für jeden Zeitpunkt der Spracheingabe eingetragen.

5.2.4 Phonemmodelle und DP–Funktionen

Die DP–Funktion innerhalb eines Phonems hängt direkt mit dem jeweiligen invertierten Phonemmodell zusammen (siehe Abbildung 5.5). Die erlaubten Übergänge von einem

Zustand zu den vorhergehenden entsprechen den Ästen der DP–Funktion für diesen Phonemzustand. Die Übergangsstrafen entsprechen denen der DP–Funktion.

Da jedes Phonem im Vokabular im Prinzip einzeln modelliert werden kann, bedeutet dies, daß innerhalb der verschiedenen Phoneme verschiedene DP–Funktionen definiert sein können. Die so definierten DP–Funktionen unterscheiden sich allerdings meist nur durch die verwendeten Strafen.

Insgesamt bedeutet dies für die tatsächliche DTW–Implementation, daß in jedem Phonem auf der Vokabularachse andere DP–Funktionen verwendet werden können. Da in JANUS nie Phonemmodelle übersprungen werden, greifen die DP–Funktionen dabei immer auf das benachbarte Phonemmodell zu.

Zusätzlich zu den in der DP–Funktion definierten Strafen gibt es in JANUS eine konstante Strafe für Phonemübergänge, sowie entweder konstante oder wortpaarabhängige Strafen für Wortübergänge (siehe Kapitel 5.2.1, Seite 39). Diese beiden Strafen werden gegebenenfalls zusätzlich zu der in der DP–Funktion definierten Strafe addiert.

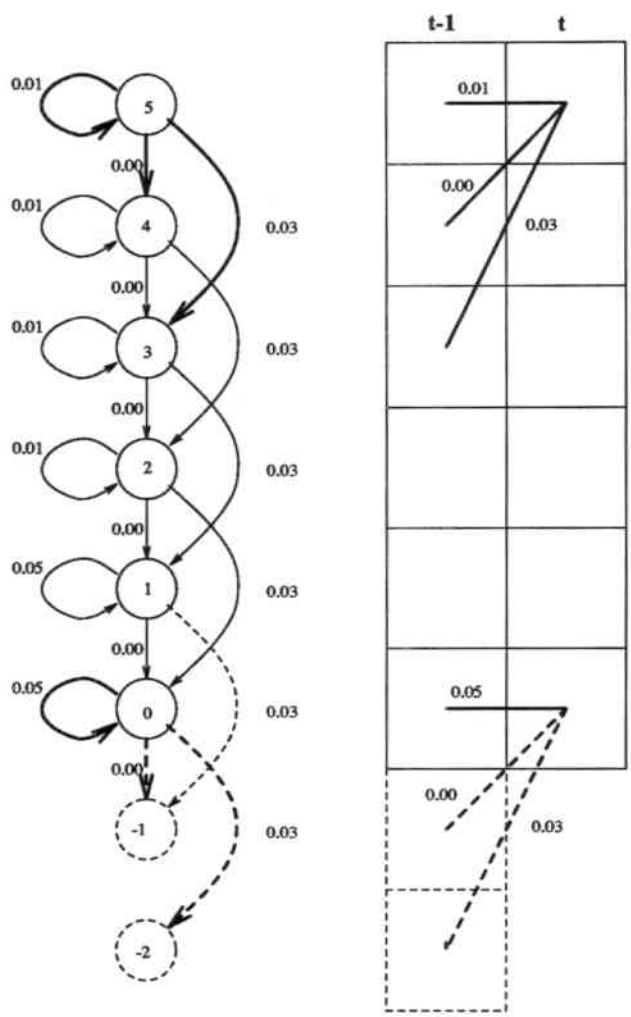

Abbildung 5.5: Zusammenhang zwischen Modell und DP–Funktion

5.3 Implementationsmöglichkeiten

Im folgenden werden die Überlegungen zur SIMD–Implementation aufgeführt.

Verteilung der Wörter auf die Prozessorelemente

Wie in der Erklärung des *one–stage* Algorithmus schon erwähnt, müssen die DP–Spalten nacheinander berechnet werden. Ein paralleles Bearbeiten von verschiedenen Zeitabschnitten ist also nicht möglich. Weil die verwendete DP–Funktion nur auf zeitlich zurückliegende Felder zugreift, ist es möglich alle Punkte der Vokabularachse parallel zu bearbeiten.

Ganze Wörter auf Prozessorelemente zu verteilen führt wegen der unterschiedlichen Länge der Wörter dazu, daß die Rechenzeit pro Prozessorelement unterschiedlich ist. Für eine SIMD–Maschine ist diese Verteilung ungünstig.

Eine Alternative, bei der dieses Problem nicht auftritt ist, die einzelnen Wörter auf mehrere Prozessorelemente zu verteilen, wobei man ganze Phonemmodelle oder Mengen von Zuständen eines Phonems auf jeweils ein Prozessorelement legt. Hierdurch läßt sich auch ein SIMD–Rechner mit vielen Prozessorelementen auslasten.

Beim *one–stage* Algorithmus treten drei Arten von Übergängen auf: Phoneminterne–, Phonem–zu–Phonem- und Wort–zu–Wort–Übergänge. Da die Berechnungen spaltenweise erfolgen, kann man diese Übergänge innerhalb einer Spalte in einer beliebigen Reihenfolge berechnen.

Erlaubt der verwendete SIMD–Rechner die gleichzeitige Kommunikation aller Prozessorelemente mit ihrem Vorgänger[3], dann können die Teile der Wörter des Vokabulars sukzessive auf die Prozessorelement verteilt werden. Die Übergänge innerhalb eines Wortes können dann mit nur einem Kommunikationsschritt erfolgen. Dies ist möglich, weil in das Überspringen von Phonemen nicht erlaubt ist. Die Wort–zu–Wort–Übergänge kann man am besten mittels einer *broadcast*–Kommunikation durchführen: jeder Wortende–Prozessor gibt seine Daten für die DP–Spalte $t-1$ an alle Wortanfangs–Prozessorelemente, die sie dann gleichzeitig in ihre lokale Liste für den Wort–Übergang mischen. Diese Methode der Verteilung der Daten über den Instruktionsbus sollte schneller sein als logarithmisches Verteilen.

Wenn genügend Speicher pro Prozessorelement vorhanden ist, kann die bearbeitbare Anzahl von Wörtern auf annähernd die Anzahl der Prozessorelemente erhöht werden. Verteilt man die Wörter so auf die Prozessorelemente, daß nie zwei Wortanfangs- und nie zwei Wortende–Phoneme auf einem Prozessorelement zu liegen kommen, dann muß beim Wortübergang pro Prozessorelement nur jeweils ein Wortende–Phoneme Daten abschicken und nur ein Wortanfangs–Phoneme Daten empfangen. Diese Verteilung der Wörter führt zu maximaler Parallelität beim Verteilen der Daten für die Wortübergänge. Wenn die Kommunikation am Wortübergang der wesentliche Kostenfaktor ist, dann spielt die zusätzliche lokale Sequentialisierung auf den Prozessorelementen eine geringere Rolle.

[3] Bei linearer Aufzählungsreihenfolge der Prozessorelemente.

Speicheraufteilung

Wie schon bei der Beschreibung des *one-stage* Algorithmus angedeutet, brauchen für jeden Zeitpunkt der DTW-Berechnung nur zwei Spalten der DP-Matrix im Speicher gehalten werden. Außerdem müssen die Wortanfangs-Zeilen mit den Rückwärtszeigern gespeichert werden.

Da der Speicher pro Prozessorelement bei den meisten SIMD-Rechnern knapp ist, wäre dies für eine Implementation ungünstig. Die Rückwärtszeiger könnten immer nur über einen gewissen Zeitabschnitt im PE-Speicher gehalten werden. Die restlichen Zeiger müßten auf eine Datei ausgelagert werden. Bei der Rückwärtssuche nach den besten Pfaden müßten sie wieder geladen werden. Das würde zu einer wesentlichen Zeitverzögerung führen. Außerdem wäre der Speicherbedarf bei den Wortanfangs-Prozessoren wesentlich größer als bei den restlichen Prozessorelementen.

Eine Alternative besteht darin, zusätzlich zu jeder kumulierten Summe, in den Feldern der zwei DP-Spalten auch die gesamte bisherige Satzhypothese abzuspeichern. Hierbei wird nur noch ein Vorwärtsdurchlauf beim *one-stage* Algorithmus benötigt. Die Satzhypothesen stehen am Ende direkt zur Verfügung. Das hat den Vorteil, daß kein Dateizugriff erfolgen muß. Es hat aber auch den Nachteil, daß die Satzhypothesen in ihrer Länge beschränkt werden.

Da bei den meisten SIMD-Rechnern nur wenig Speicherplatz pro Prozessorelement zur Verfügung steht und außerdem eine deutliche Beschleunigung des *one-stage* Algorithmus erzielt werden soll, wird im folgenden von der zweiten Speicherrepräsentation ausgegangen.

Die Bewertungen für die Spracheingabe, mit denen die DP-Matrix initialisiert wird, sind sehr umfangreich. Da mit Spracheingaben bis zu 10 Sekunden gerechnet wird, müssen bis zu 1000*136 Bewertungen (544 KBytes) verarbeitet werden. Je nach verwendetem SIMD-Rechner muß diese Datenmenge blockweise bearbeitet werden. Sowohl das Einlesen auf die Prozessorelemente als auch das Einlesen auf den sequentiellen Steuerrechner ist denkbar. In letzterem Fall müssen die Phonem-Bewertungen noch an die Prozessorelemente verteilt werden, die das jeweilige Phonem berechnen.

5.4 Implementation

Die Implementierung des *one-stage* Algorithmus auf der MasPar MP-1 wird im folgenden beschrieben. Neben den bisherigen Überlegungen zur Implementation des *N best one-stage* Algorithmus, bestand die Schwierigkeit in der Vereinheitlichung des Zugriffs auf die Übergangstabelle für die Phonemmodelle. Das Ziel war einen einheitlichen Aufruf für das Mischen der N-besten-Listen zu erhalten.

Invertierung von Phonemmodellen

Die Zustandsübergänge in den Phonemmodellen sind in der Konfigurationsdatei vorwärtsgerichtet definiert (Anhang A.1). Für jeden Zustand sind die Übergänge zu den Folgezuständen angegeben. Für die parallele Implementation hat es sich als wesentlich praktischer erwiesen, rückwärtsgerrichtete Übergänge zu verwenden. Dies ermöglicht mit einem Zugriff auf die Übergangstabelle alle Übergänge zu möglichen Vorgängern zu erhalten.

In Tabelle 5.1 sieht man links die Übergangstabelle für ein vorwärtsgerichtetes Modell mit den Folgezuständen und Übergangsstrafen. Übergänge zum nachfolgenden Phonemmodell sind durch die Zustandsnummern *+1* und *+2* angedeutet. Dasselbe Modell sieht man auch in Abbildung 5.6. Hierbei deuten gestrichelte Zustände und Übergänge benachbarte Phonemmodelle auf der Vokabularachse an. Rechts in Tabelle 5.1 sieht man das invertierte Phonemmodell. Übergänge zum vorhergehenden Phonemmodell sind mit *-1* und *-2* angedeutet. Abbildung 5.7 zeigt das invertierte Modell.

Vorwärts-Modell			
von	Übergänge		
0	0 0.05	1 0.00	2 0.03
1	1 0.05	2 0.00	3 0.03
2	2 0.01	3 0.00	4 0.03
3	3 0.01	4 0.00	5 0.03
4	4 0.01	5 0.00	+1 0.03
5	5 0.01	+1 0.00	+2 0.03

Rückwärts Modell			
von	Übergänge		
0	0 0.05	−1 0.00	−2 0.03
1	0 0.00	1 0.05	−1 0.03
2	0 0.03	1 0.00	2 0.01
3	1 0.03	2 0.00	3 0.01
4	2 0.03	3 0.00	4 0.01
5	3 0.03	4 0.00	5 0.01

Tabelle 5.1: Invertierung von Phonemmodellen

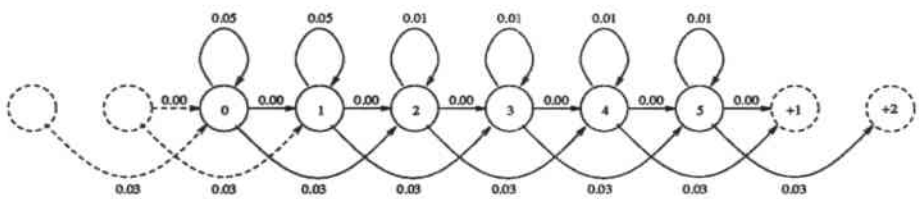

Abbildung 5.6: Vorwärtsgerichtetes Phonemmodell

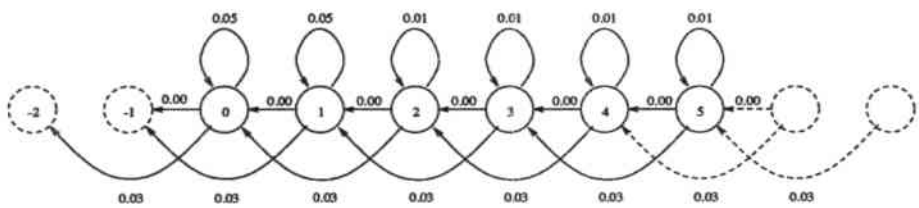

Abbildung 5.7: Rückwärtsgerichtetes Phonemmodell

Wesentlich für das Invertieren der Modelle ist, daß in allen Phonemmodellen alle Übergänge und Übergangsstrafen die aus einem Phonemmodell herausreichen identisch definiert sind. Dies ist in JANUS immer erfüllt.

Verteilung der Wörter auf die Prozessorelement

Beim gegebenen deutschen Vokabular mit 464 Wörtern (siehe Tabelle 2.1 auf Seite 15) und einer MasPar MP-1 mit 4096 Prozessorelementen bietet sich das Verteilen der einzelnen Phoneme im Vokabular auf die einzelnen Prozessorelemente an. Allerdings ist die Kommunikation entlang der linearen Aufzählungsreihenfolge der Prozessorelemente bei der MasPar MP-1 nicht vorgesehen — nur die Kommunikation im zweidimensionalen PE–Gitter (mit dem *X–Net*). Dies kann man für die gleichzeitige Berechnung der Phonem–zu–Phonem–Übergänge ausnutzen, indem man zum Beispiel die Zeilen des PE–Gitters so gut wie möglich mit kompletten Wörtern füllt. Man hätte auch die eindimensionale Kommunikation durch jeweils zwei *X–Net* Kommunikationen nachbilden können — was aber langsamer gewesen wäre. Durch die zweidimensionale Anordnung der Daten braucht man

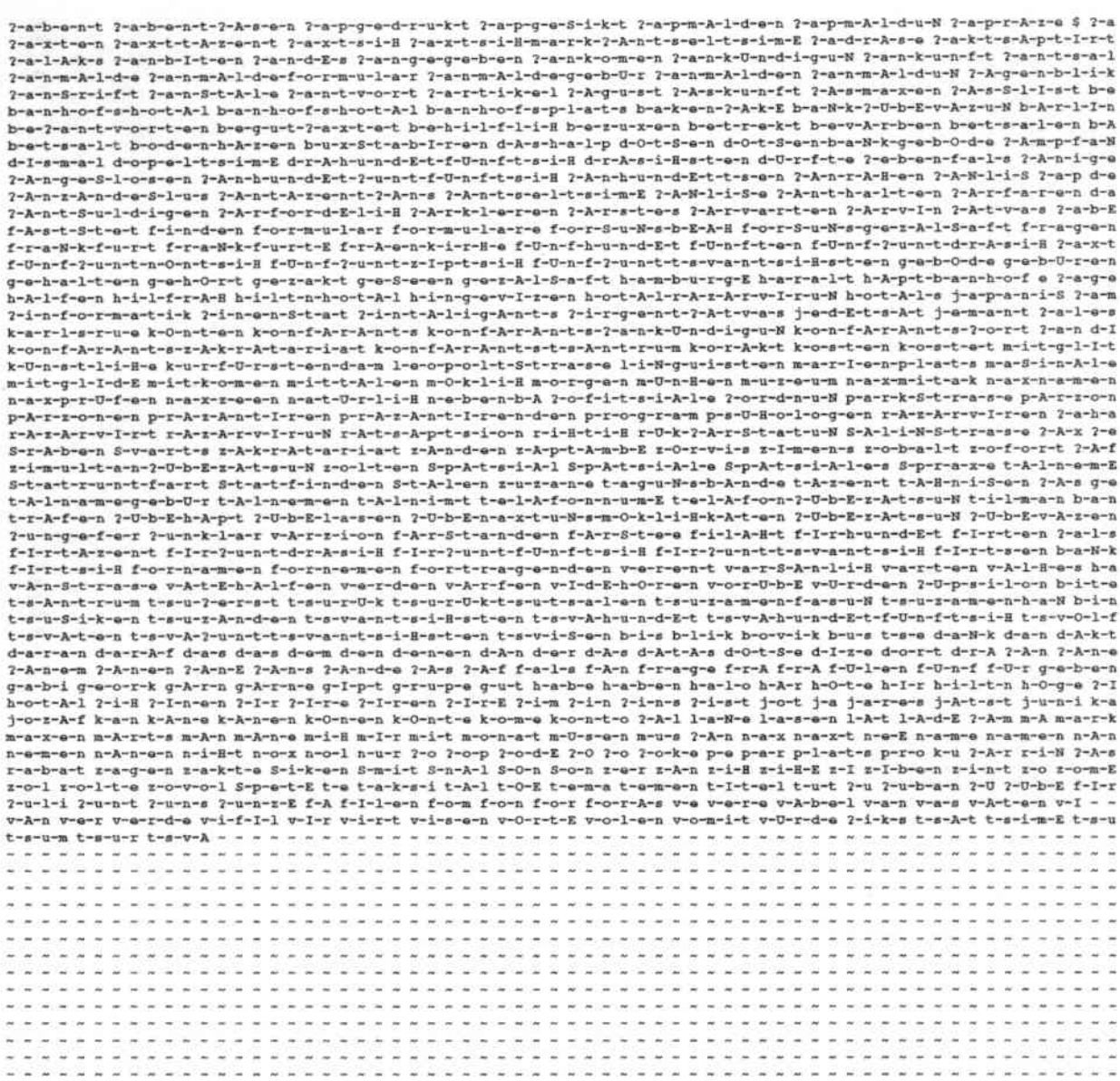

Abbildung 5.8: Verteilung der Wörter auf die Prozessorelemente

einen geeigneten Parkettierungs–Algorithmus[4]. Mittels einer einfachen Heuristik läßt sich eine annehmbare Füllung der Zeilen erreichen (siehe Abbildung 5.8). Hier wurden die Wörter des Vokabulars in zwei Durchläufen auf die Prozessorelemente verteilt: zuerst alle Wörter mit mehr als 5 Phonemen, dann die kürzeren Wörter. Dabei wird außerdem in beiden Durchläufen darauf geachtet, daß am Ende der Zeile kein einzelnes Prozessorelement leer bleibt (ausgenommen im zweiten Durchlauf in der letzten Zeile).

Die MASPAR sieht keine *broadcast*–Kommunikation zur Übertragung ganzer Datenblöcke in den MPL Bibliotheks–Routinen vor. Daher wurde sie durch Kopieren der Daten von einem Prozessorelement an die ACU und dann von dort an alle aktiven Prozessorelemente nachgebildet.

Speicheraufteilung

Wegen des besseren Zeitverhaltens werden pro Prozessorelement ganze Satzhypothesen pro DP–Element gespeichert. In Abbildung 5.9 sieht man ein einzelnes Feld der DP–Matrix. Es besteht aus einer N–besten–Liste mit den kumulierten Summen und den Satzhypothesen. Eine DP–Spalte besteht dabei aus 9 N–besten–Listen[5]: 6 Listen für

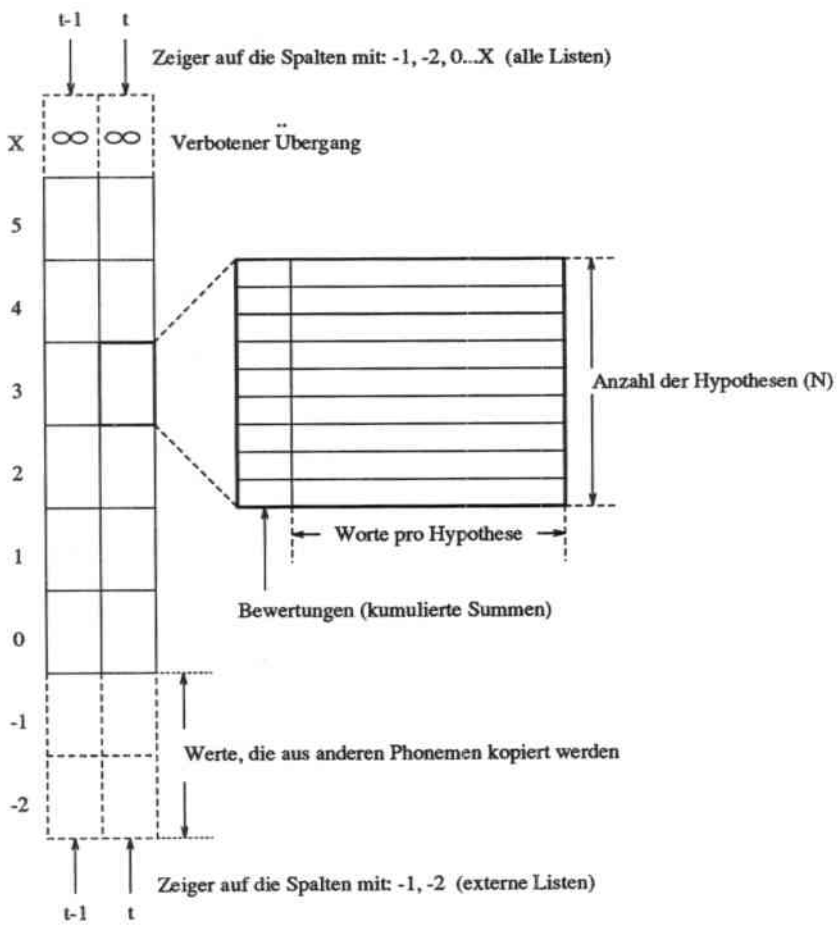

Abbildung 5.9: DTW–Matrix: Repräsentation im PE–Speicher

[4] Ein *knapsack*–Problem.

[5] Die hier auftretenden Zahlen sind Konstanten des Programms und können einfach geändert werden

die maximal 6 phoneminternen Zustände und 2 Listen zur Zwischenspeicherung externer Listen sowie einer Liste „X", die nur Bewertungen nahe Unendlich enthält. Letztere waren nötig um einen einheitlichen Aufruf für die Misch–Routine zu gewährleisten — auch im Fall daß ein Übergang in der Zustandsübergangstabelle nicht definiert ist. Der Wert X ist dabei gleich der Anzahl der maximal erlaubten Zustände.

In der Abbildung 5.9 sind vier Zeiger angedeutet. Diese indizieren die beiden DP–Spalten. Je zwei Zeiger indizieren Spalten für den Zeitpunkt $t-1$ und zwei weitere Zeiger indizieren Spalten für den Zeitpunkt t. Die zwei oberen Zeiger indizieren jeweils eine gesamte Spalte. Diese beiden Zeiger werden während des Fortschreitens entlang der Zeitachse vertauscht. Die beiden unteren Zeiger indizieren jeweils *nur* die DP–Elemente *-1* und *-2*. Die Elemente *-1* und *-2* speichern *externe* N–besten–Listen für Phonem- und Wort–Übergänge. Bei jedem externen Übergang werden die Elemente der finalen Zustände des vorhergehenden Phonems in diese Elemente kopiert und diese werden mit den Elementen für $t-1$ gemischt. Anschließend werden die Zeiger für die externen Listen vertauscht.

Die Berechnung der Übergänge erfolgt in folgender Reihenfolge:

1. alle Wortübergänge
   Alle Wortanfangs- und Wortende–PEs nehmen daran teil.
   Alle Wortende–PEs schicken ihre finalen Listen sequentiell per *broadcast*
   an alle Wortanfangs–PEs. O(Anzahl der Wörter)
   Pro Gruppe von empfangenen Listen mischen die Wortanfangs–PEs diese zu ihren externen Listen.

2. ein Übergang vom Vorgänger–Phonem
   Alle Prozessorelemente außer Wortanfangs–PEs nehmen daran teil.
   Sie holen sich die zwei finalen Listen ihres Vorgängers O(1)
   und mischen sie anschließend zu den eigenen beiden externen Listen.

3. interne Übergänge
   Alle Prozessorelemente mischen *alle* ihre N–besten–Listen —
   inklusive der beiden *externen* Listen.

Durch diese Vorgehensweise werden die Phoneminternen-, die Phonem–zu–Phonem- und die Wort–zu–Wort–Übergänge gleichmäßig behandelt. Die in der Übergangstabelle verbotenen Zustände werden in Zugriffe auf die Liste „X" umgesetzt. Das Mischen einer N–besten–Liste mit der Liste „X" führt zu keiner Veränderung, weil sie nur Werte nahe Unendlich enthält. Hierdurch werden alle Aufrufe der Misch–Routine einheitlich gehandhabt. Durch diese Vereinheitlichungen müssen keine Abfragen gemacht werden — keines der Prozessorelemente der SIMD–Maschine wird hierzu abgeschaltet.

Zusätzlich zu den Teilen der beiden DP–Spalten müssen die Übergangstabelle und die Liste der Vorgängerworte[6] auf den Prozessorelementen gespeichert werden.

Bei der MasPar MP-1 können die bis zu $1000 * 136$ Bewertungen weder komplett auf den Prozessorelementen noch komplett auf der ACU gespeichert werden. Da der parallele Lesebefehl `pp_read()` zu langsam ist (siehe Kapitel 7) werden die Bewertungen blockweise in die ACU gelesen und für jeden Abschnitt der Zeitachse an alle Prozessorelemente mittels *broadcast* verteilt.

[6] Falls eine Wortpaar- oder Bigramm—Grammatik verwendet wird.

Der geringe Speicher der MasPar MP-1 zwingt zu einem Kompromiß zwischen der Wahl von N und der Wahl der Satzhypothesenlänge. Diese beiden Parameter[7] bestimmen in erster Linie den Speicherplatzbedarf für die beiden DP-Spalten.

Stille

Das in JANUS eine Einschränkung im *one-stage* Algorithmus gemacht wird, wurde bereits angemerkt: gültige Satzhypothesen müssen mit dem Wort „Stille" anfangen und enden. Die hat bei der Implementation den Vorteil, daß nur die N-besten-Liste des Worts „Stille" ausgegeben werden muß.

Falls eine Wortpaar- oder Bigramm-Grammatik verwendet wird, müssen die Listen der gültigen Vorgängerworte pro Wortanfangs-Prozessorelement gespeichert werden. Aus der Statistik für das deutsche Vokabular (Anhang B) geht hervor, daß die Anzahl der gültigen Vorgänger von Stille überdurchschnittlich groß ist. Die unterschiedliche Anzahl der möglichen Vorgängerworte bedeutet, daß im allgemeinen das Durchsuchen der Vorgängerlisten pro Prozessorelement unterschiedlich lange dauern wird.

Dieses Verhalten kann man dadurch mildern, daß man das Phonem „Stille" repliziert — allerdings mit kleineren, disjunkten Vorgängerlisten. Dies ist bei „Stille" besonders einfach möglich, weil es ein Wort mit nur einem Phonem ist. Durch das Replizieren kann die mittlere Dauer beim Durchsuchen der Vorgängerlisten gesenkt werden.

5.5 Ergebnisse

Die Implementation des parallelen *N best* Algorithmus benötigt für die Suche nach den 3 besten Satzhypothesen bei 446 Wörtern im Vokabular und bei einem 2,4 Sekunden langen Satz etwa 5,8 Minuten. Das entspricht einer Beschleunigung um den Faktor 5 gegenüber der entsprechenden, gemessenen Zeitdauer des sequentiellen Algorithmus. Natürlich ist dies nur ein punktueller Vergleich — das Zeitverhalten des parallelen Algorithmus ist nur proportional zur Anzahl der Wörter.

[7] Zur Programm-Übersetzungszeit können diese Konstanten des Programms festgelegt werden.

Kapitel 6

Auswertung

6.1 Signalvorverarbeitung

Durch die algorithmische Beschleunigung findet die sequentielle Vorverarbeitung nun in kürzerer Zeit als das Sprechen der Eingabe statt. Erst diese Beschleunigung machte die Implementation einer Fließbandverarbeitung sinnvoll.

Die Fließbandverarbeitung auf einem Arbeitsplatzrechner war hier sinnvoller als die parallele Implementation der Signalvorverarbeitung, da die bei der Fließbandverarbeitung zu erzielende Verarbeitungsgeschwindigkeit nicht wesentlich verbessert werden könnte. Außerdem können durch die Fließbandverarbeitung mehrere Sprachaufnahmen gleichzeitig (inklusive der Vorverarbeitung) an mehreren Arbeitsplatzrechnern gemacht werden.

Erste Ergebnisse der Studienarbeit zur Fließbandverarbeitung [38] zeigen, daß es inzwischen möglich ist, etwa eine halben Sekunde nach dem Beenden der Spracheingabe die Vorverarbeitung abzuschließen. Diese Zeitspanne ist unabhängig von der Länge der Spracheingabe.

Damit ergeben sich folgende Zeiten zwischen Beenden der Spracheingabe und Ende der Vorverarbeitung für einen 2,4 Sekunden langen Satz:

Ursprüngliches System	3,2 Sekunden
Algorithmische Verbesserungen	1,6 Sekunden
Mit Fließbandverarbeitung	0,5 Sekunden

Diese Beschleunigung macht sich besonders bei Vorführungen von JANUS positiv bemerkbar.

6.2 Analyse der Sprachsignale

Vergleich der verschiedenen Versionen

Für die Erkennung einer 2.6 Sekunden langen Eingabe braucht JANUS:

ohne MasPar:

17,6 s =	12,4 s	LPNN Vorwärtsberechnungen auf einer DEC 5000
	+ 2,3 s	DTW auf einer DEC 5000 (3 Dialoge, 115 Wörter)
	+ 2,4 s	Rest (Graphik, kleinere Berechnungen)
	+ 0,5 s	Signalvorverarbeitung mit Fließbandverarbeitung

mit MasPar:

7,9 s =	0,6 s	LPNN Vorwärtsberechnungen auf einer MasPar MP-1
	+ 2,1 s	Kommunikation zwischen DEC 5000 und MasPar (0,3...2,5 s)
	+ 2,3 s	DTW auf einer DEC 5000 (3 Dialoge, 115 Wörter)
	+ 2,4 s	Rest (Graphik, kleinere Berechnungen)
	+ 0,5 s	Signalvorverarbeitung mit Fließbandverarbeitung

Die Kommunikation zwischen MasPar und der DEC 5000, auf der JANUS läuft, erfolgt über ein gemeinsames Dateisystem. Sie braucht je nach Auslastung des Netzes zwischen 0,3 und 2,5 Sekunden pro bearbeitetem Satz. Diese Zeit würde entfallen, wenn JANUS auf einer DEC 5000 als MasPar-Vorrechner laufen könnte. Damit würde der Spracherkenner in JANUS alleine durch die Berechnung der LPNNs auf der MasPar etwa 3 mal schneller als bei der Verwendung einer einzelnen DEC 5000 laufen.

Bei den hier gezeigten Abbildungen (Seite 51) sind jeweils die Geraden zwischen dem Nullpunkt und dem Punkt (5 sek, 5 sek) besonders interessant. Nur Teilaufgaben, deren Rechenzeiten weit unter dieser Gerade liegen, können in einem "Echtzeit"–Spracherkenner sinnvoll eingesetzt werden. Die hier gezeigten Meßergebnisse wurden auf einer ansonsten weitestgehend unbelasteten DEC 5000 und auf einer ansonsten freien MasPar gemessen.

DTW und LPNNs im ursprünglichen JANUS

Abbildung 6.1 zeigt, daß die Gesamtzeiten für die Spracherkennung hier deutlich über der "Echtzeit"–Geraden liegen. Die Teilaufgabe DTW bezieht sich hier auf die Suche der besten Satzhypothese. Diese wird zwar relativ schnell berechnet, müssen aber noch erheblich beschleunigt werden um die Summe der Ausführungszeiten so zu minimieren, daß sie unter der "Echtzeit"–Geraden liegen. Die N–besten Suche ist hier wegen des Maßstabs nicht angegeben (die Auflösung der restlichen Teilaufgaben wäre sonst zu gering).

Analyse der Sprachsignale: LPNNs auf der MasPar

In Abbildung 6.2 wurden die Vorwärtsberechnungen der LPNNs auf der MasPar berechnet; die restlichen Teilaufgaben wurden auf einer DEC 5000 berechnet. Man sieht, daß die Gesamtausführungszeiten in folge der enorm beschleunigten LPNN–Berechnungen stark gesunken sind. Bei einem 2 Sekunden langen Satz werden die LPNN–Bewertungen für 200 Zeitpunkte berechnet. Eine DEC 5000 braucht dafür etwa 9,5 Sekunden, die MasPar MP-1 hingegen nur etwa 0,43 Sekunden. Die Abbildung 6.3 zeigt die zusammengefaßten Meßergebnisse.

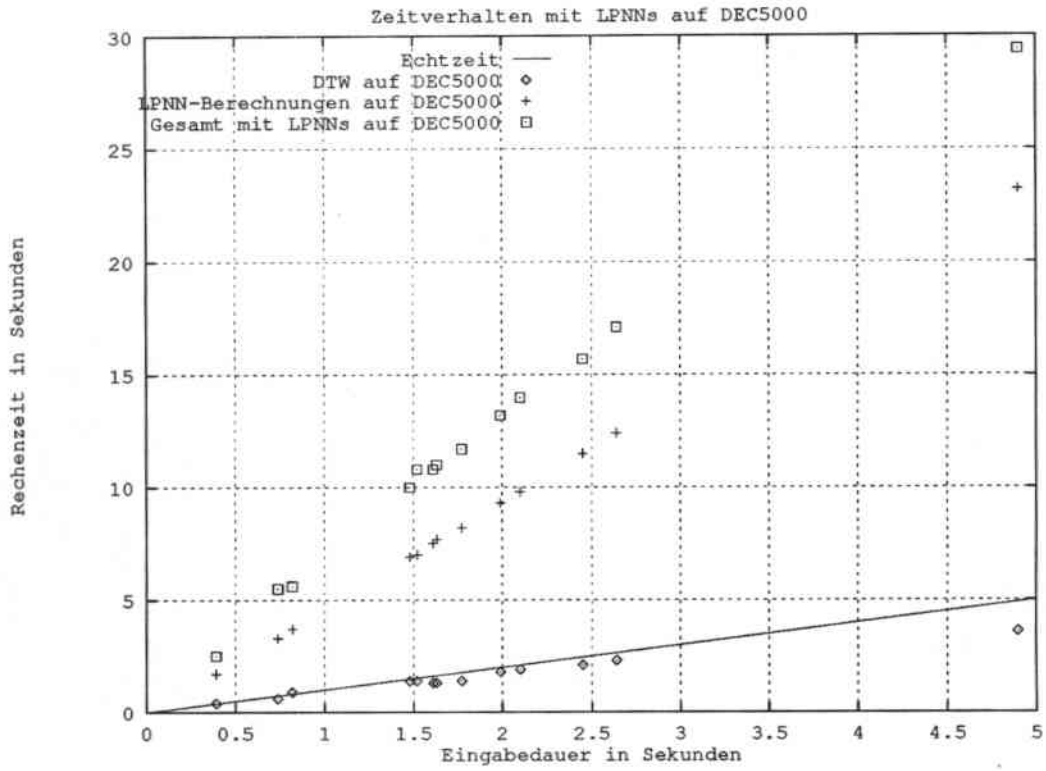

Abbildung 6.1: Meßergebnisse für DEC 5000

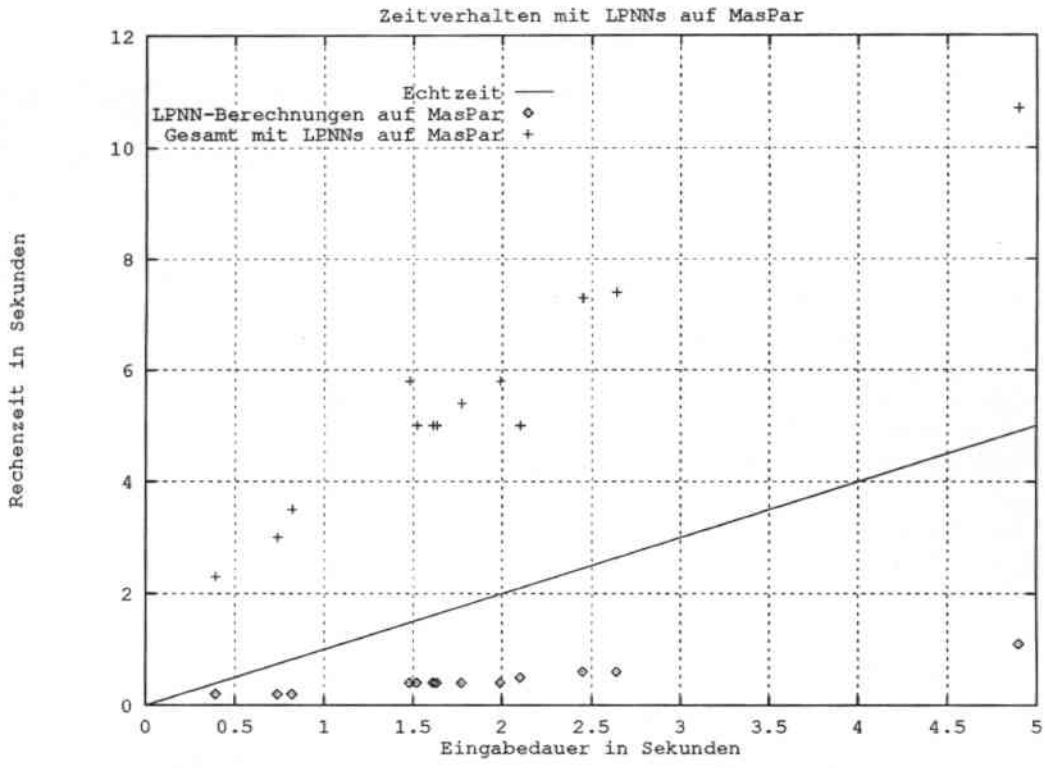

Abbildung 6.2: Meßergebnisse für DEC 5000 und MasPar MP-1

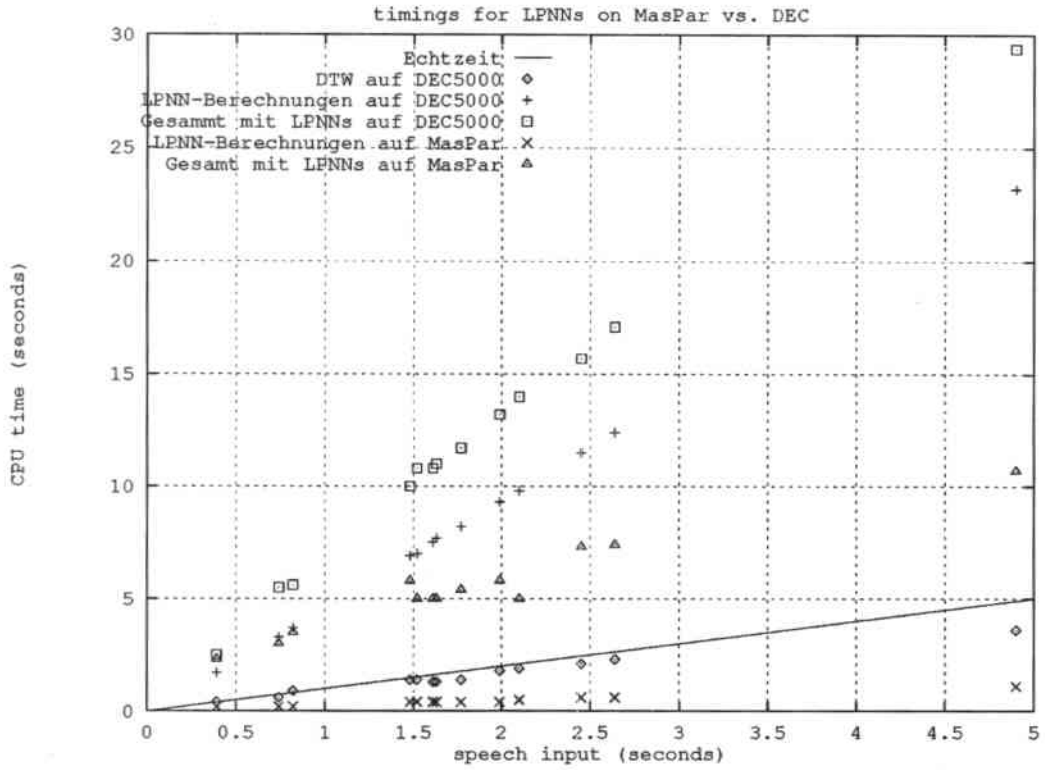

Abbildung 6.3: Zusammengefaßte Meßergebnisse

Vergleich zwischen iWarp und MasPar

Zur gleichen Zeit wie diese Arbeit entstand an der Carnegie Mellon University eine parallele Implementation derselben Teilaufgabe auf einer iWarp, einer MIMD–Maschine mit 64 Prozessorelementen. Die dort erzielte mittlere Leistung beträgt 15,6 MCPS [32].

	mittlere Leistung [MCPS]	Kosten [pro 1000 CPS]
MasPar (SIMD, 4096 PEs)	41,4	\$13,8
iWarp (MIMD, 64 PEs)	15,6	\$29,6

Tabelle 6.1: Vergleich der MasPar- gegenüber der iWarp–Implementation

Da es sich bei der Vorwärtsberechnung der LPNNs um ein stark datenparalleles Problem handelt, entspricht die fast dreimal höhere Geschwindigkeit der MasPar MP-1 gegenüber der iWarp (bei weniger als der Hälfte der Kosten) den Erwartungen die man an SIMD–Maschinen stellt. SIMD–Rechner zeichnen sich bei stark datenparallelen Problemen gegenüber MIMD–Rechnern sowohl durch ihre einfachere Programmierung als auch durch ihre günstigere Kosten–zu–Nutzen Relation aus.

6.3 Suche der N besten Satzhypothesen

Die Zeitdauer (bei einem LPNN–Erkenner), die JANUS für die Erkennung eines kontinuierlich gesprochenen Satzes von 2,4 Sekunden Dauer braucht, beträgt einige Minuten. Die genaue Zeitspanne hängt quadratisch von der Anzahl der Wörter im Vokabular ab. Sie beträgt bei 115 Wörtern 3 Minuten und bei 447 Wörtern 30 Minuten.

Für einen 2,4 Sekunden langen Satz braucht die parallele Suche für 447 Wörter im Vokabular bei der Berechnung der drei besten Satzhypothesen 5,8 Minuten. Das ist etwa 5 mal schneller[1] als der sequentielle *N best* Algorithmus in JANUS, aber immer noch deutlich oberhalb des erwünschten Echtzeitverhaltens.

Immerhin kann man durch die erzielte Beschleunigung in erträglicher Zeit das Verhalten der Erkennungsgenauigkeit bei mehr als drei Hypothesen ausprobieren.

6.4 Schlußfolgerungen

Die untersuchten Algorithmen verhielten sich beim Ansatz sie zu parallelisieren sehr unterschiedlich:

- Im Fall der Signalvorverarbeitung wäre eine Parallelisierung erfolgversprechend gewesen, es war aber möglich das Problem eleganter und billiger durch Fließbandverarbeitung zu lösen.
- Aufgrund der hohen Datenparallelität des Problems konnten die Vorwärtsberechnungen für die neuronalen Netze (LPNNs) erwartungsgemäß erheblich beschleunigt werden. Dazu hat besonders beigetragen, daß hierbei keinerlei Kommunikation zwischen Prozessorelementen nötig war.

  Das trotzdem nur ein Faktor 22 an Geschwindigkeit erreicht wurde, deutete die Grenzen der MasPar MP-1 an. Eine größere Geschwindigkeit wäre bei dieser Anwendung allerdings auch nicht sinnvoll gewesen.
- Vom parallelen N–best DTW wurde erwartet, erheblich schneller als das sequentielle Programm zu sein. Die Erwartungen waren hier viel zu hoch. Der hohe Anteil an Kommunikation führte zu einer wesentlich geringeren Beschleunigung als im Falle der LPNNs.

  Der erreichte Faktor 5 zwischen dem sequentiellen und dem parallelen Algorithmus war eher dürftig, in Anbetracht der Tatsache, daß die Ausführung nochmal um ein Faktor 170 schneller werden müsste, damit die Ausführungszeit unter der Dauer der Spracheingabe liegt.

Eine genaue Untersuchung der gegebenen Algorithmen und Erfahrungen mit dem zu verwendenden Parallelrechner sind nötig um eine aussagekräftige Abschätzung über die erzielbare Beschleunigung geben zu können.

Nicht alle Rechenzeitprobleme lassen sich durch die Verwendung heutiger SIMD–Rechner lösen.

[1] Natürlich ist dies nur ein punktueller Vergleich — das Zeitverhalten des parallelen Algorithmus ist nur proportional zur Anzahl der Wörter.

Kapitel 7

MasPar–Erfahrungen

7.1 Software

MPL

MPL hat sich als eine benutzbare Programmiersprache erwiesen. Zwar unterstützt die momentane Version noch nicht ANSI–C, ein entsprechender Übersetzer wird jedoch bald zur Verfügung stehen. Das Fehlen eines auf MPL angepaßten „lint“ fiel störend auf.

Portierbarkeit

Die während diese Arbeit entstandenen Programme werden an der MasPar MP-1 der Carnegie Mellon University verwendet. Diese MasPar MP-1 hat einen DEC 5000 Vorrechner und kein MPDA.

Die Programme waren problemlos Portierbar. Das Fehlen des MPDA war beim Einlesen der Gewichtsdateien nicht spürbar (siehe unten).

MasPar Programming Environment (MPPE)

Der MPPE ist ein Debugger auf Quellcodeebene, er stellt dem Benutzer eine sehr hohe Funktionalität zur Verfügung. Gerade bei datenparallelen Programmen kann es sehr hilfreich sein die Daten zu inspizieren.

Leider ist der MPPE für die Benutzung zu langsam. Schon bei den einfachsten Operationen bei der Benutzung der Maus–Oberfläche sind Reaktionszeiten von etwa 10 Sekunden die Regel. Eine neue Version des MPPE wird in kürze installiert — sie soll auf einem externen Arbeitsplatzrechner lauffähig und dadurch schneller sein.

Prozeßablaufsteuerung

Die Prozess-Ablaufplanung der MasPar MP-1 stellt nur Zeitscheiben einer gegebenen, festen Länge (zwischen einer und zehn Sekunden) zur Verfügung. Es werden dabei nur Prozesse verwaltet, die zusammen in den vorhandenen PE–Speicher passen. Normalerweise ist es nicht vorgesehen, daß ein Prozeß zwischenzeitlich einen anderen Prozeß rechnen läßt.

Nach Absprache mit der Firma MasPar konnte jedoch eine Möglichkeit gefunden werden, den eigenen Prozeß zeitweise von der DPU zu verdrängen (Siehe Anhang D, `GiveUp_fe.c`). So kann ein anderer Prozess vorgezogen werden, obwohl das die eigentliche Ablaufplanung nicht vorsieht.

7.2 Hardware

Speicher

Bei der Implementation des parallelen DTW wurden mehrfach die Grenzen des zur Verfügung stehenden Speichers, sowohl auf den Prozessorelementen als auch auf der ACU, erreicht. Ein Speicher von 64 KBytes pro Prozessorelement wäre hier hilfreich gewesen.

Leider ist der Speicherausbau der Prozessorelemente sehr kostenintensiv und ein Ausbau des ACU–Speichers nicht vorgesehen.

Kommunikation

Beim DTW trat das Problem auf, einen Datenblock von einem Prozessorelement an eine Menge von Prozessorelementen zu verteilen. Die schnellste Möglichkeit dafür wäre ein *broadcast* gewesen (siehe Kapitel 5.3). Der Broadcast ganzer Datenblöcke wird nicht durch die MPL–Bibliothek unterstützt — die einzelnen Elemente der Datenblöcke mussten daher einzeln kopiert werden. Es ist anzunehmen, daß Bibliotheksroutinen für Datenblöcke hier eine Beschleunigung bringen würden.

Beim DTW wäre auch eine Kommunikation entlang der eindimensionalen Reihung der Prozessorelemente wünschenswert gewesen. Die Grundoperationen für die *X-Net* Kommunikation sind allerdings nur für das zweidimensionale PE–Gitter vorgesehen. Man kann die eindimensionale Kommunikation durch jeweils zwei *X-Net* Kommunikationen nachbilden — was aber um den Faktor 2 langsamer ist.

Die *Global Router* Kommunikation wurde hier nicht verwendet.

MasPar Parallel Disk Array (MPDA)

Beim Laden der Gewichtsdatei wurde zuerst die parallele Lese–Operation `pp_read()` verwendet [18, Kapitel 4]. Da aber eine für die parallele Speichereinheit wesentliche VME–Bus Karte fehlte, war das Einlesen der Gewichte sehr langsam — es dauerte etwa 50 Sekunden.

Durch sequentielles Einlesen der Gewichtsdatei auf der ACU und durch gruppenweise Verteilung der Gewichte jedes LPNNs an die dafür zuständige Gruppe von Prozessorelementen, konnte das Einlesen auf nur 4,8 bis 5,2 Sekunden reduziert werden.

Rechenleistung

Die bei der Berechnung der LPNN–Bewertungen erzielte Geschwindigkeit war für die Anwendung „Spracherkennung“ beeindruckend. Die Gesamtgeschwindigkeit von JANUS wurde, auch wenn nur die LPNN–Berechnungen ausgetauscht wurden, deutlich erhöht.

Der Faktor 22 zwischen einer MasPar MP-1 und einer DEC 5000 war auf den zweiten Blick allerdings nicht sehr groß — zumal beim parallelen Algorithmus keinerlei Kommunikation verwendet wurde. Das ist im wesentlichen auf die geringe Rechenleistung der einzelnen Prozessorelemente zurückzuführen.

•

Kapitel 8

Ausblick

Die folgenden Möglichkeiten zur Erweiterung beziehungsweise zur Verbesserung bieten sich an:

- Der sprecherunabhängigen LVQ–Erkenners kann weitestgehend analog zu den LPNNs implementiert werden.
- Man kann unter Absprache mit MasPar versuchen, die Kommunikation an den Wortübergängen des parallelen DTW zu beschleunigen (das scheint allerdings nicht sehr aussichtsreich).

  Falls das gelingt kann :

  - ▷ die Anzahl der Wörter im Vokabular erweitert werden.
  - ▷ das parallele DTW um die Wortpaar–Grammatik beziehungsweise um Bigramme erweitert werden.
- Sobald die DEC 5000 als Vorrechner für die MasPar verfügbar ist, kann man die sequentiellen Teile von JANUS dort laufen lassen und so die Zeit für den Dateitransfer bei den LPNN- beziehungsweise LVQ–Berechnungen verkürzen.
- Zur Kommunikation zwischen JANUS und der MasPar MP-1 können auch *sockets* verwendet werden.

Anhang A

Konfigurationsdateien

A.1 Die Modell–Datei

In der Modell–Datei *(modelfile)* werden die zur Phonemmodellierung wichtigen Parameter festgelegt, die sowohl in der Analyse der Sprachsignale, wie auch in der Suche der besten Satzhypothesen verwendet werden.

Im Vorspann der Datei stehen außer der Versionsnummer des Dateiformates drei wichtige Grössen: die Anzahl der hier definierten Phonemmodelle, Phoneme und der neuronalen Netze. Diese Informationen sind im Rest der Datei nocheinmal implizit enthalten. Um bei der Initialisierung der LPNN–Berechnungen nicht die gesamte Datei auswerten zu müssen, wurden diese drei Größen in das Dateiformat aufgenommen. Bei den LPNN–Berechnungen erfolgt keine Konsistenzprüfung, aber in JANUS.

Die im folgenden angegebene Modelldatei für die deutsche Sprache enthält 3 Modellbeschreibungen. Diese Phonemmodelle haben die Namen: 2-state, 6-state, r-model. Sie unterscheiden sich in der Anzahl ihrer Zustände und in den Übergangsstrafen.

Jedes Phonemmodell wird einzeln definiert. Man erkennt, daß in jede Modell doppelt so viele Zustände, wie neuronale Netze definiert werden. In der darauf folgenden Zuordnungstabelle wird fest gelegt, durch welche neuronalen Netze, die einzelnen Zustände berechnet werden. Abschliessend folgt die Zustandsübergangstabelle mit den Übergangsstrafen. Es werden nur die erlaubten Übergänge definiert.

Nach den Definitionen der Phonemmodelle folgt eine Liste der erlaubten Phoneme. Jedes Phonem hat einen Namen, eine einbuchstabige Abkürzung, sowie ein ihm zugeordnetes Phonemmodell. Zum Beispiel hat das „Ä“ den Namen `AE`, die Namensabkürzung `A` und wird durch das *6–state* Modell modelliert.

Auffällig sind die beiden Phoneme `SIL` und `?`, das das Phonem für Stille *(silence)* und der sogenannte *glottal stop*. Dabei handelt es sich um ein Geräusch, das vom Vokaltrakt des Menschen erzeugt wird, wenn ein Vokal am Anfang eines Wortes auftritt. Wenn man zum Beispiel ein „A“ ausspricht baut sich noch bevor der Laut ausgesprochen wird, sich im Bereich des Kehlkopfes ein Druck auf. Dieser Druck wird dann plötzlich freigelassen und der Laut „A“ wird geformt. Dieser freiwerdende Druck führt zu einem charakteristischen Geräusch, das *glottal stop* genannt wird.

Beispiel einer Modelldatei *(modelfile)*:

```
Version 2 model file
3 models
46 phonemes
136 nets
----------------------------------------
model name = 2-state
2 states, 1 net

state   net
0       0
1       0

from    to   penalty
0       0       .01
0       1       0.0
1       1       .01
1       2       0.0
----------------------------------------
model name = 6-state
6 states, 3 nets

state   net
0       0
1       0
2       1
3       1
4       2
5       2

from    to   penalty
0       0       .05
0       1       .00
0       2       .03

1       1       .05
1       2       .00
1       3       .03

2       2       .01
2       3       .00
2       4       .03

3       3       .01
3       4       .00
3       5       .03

4       4       .01
4       5       .00

5       5       .01
5       6       .00
----------------------------------------
model name = r-model
6 states, 3 nets

state   net
0       0
1       0
2       1
3       1
4       2
5       2

from    to   penalty
0       0       .10
0       1       .00
0       2       .03

1       1       .10
1       2       .00
1       3       .03

2       2       .10
2       3       .00
2       4       .03

3       3       .10
3       4       .00
3       5       .00

4       4       .10
4       5       .00

5       5       .10
5       6       .00
----------------------------------------
SIL     $       2-state
?       ?       6-state
B       b       6-state
P       p       6-state
PH      P       6-state
D       d       6-state
T       t       6-state
TH      T       6-state
G       g       6-state
K       k       6-state
KH      K       6-state
CH      H       6-state
X       x       6-state
J       j       6-state
F       f       6-state
V       v       6-state
H       h       6-state
L       l       6-state
M       m       6-state
N       n       6-state
NG      N       6-state
R       r       r-model
RJ      R       r-model
RX      R       r-model
S       s       6-state
Z       z       6-state
SCH     S       6-state
A       a       6-state
AH      a       6-state
E       e       6-state
EH      e       6-state
AE      A       6-state
ER      E       6-state
I       i       6-state
IE      I       6-state
O       o       6-state
OH      o       6-state
U       u       6-state
UH      u       6-state
OE      O       6-state
OEH     O       6-state
UE      U       6-state
UEH     U       6-state
AI      A       6-state
AU      A       6-state
EU      O       6-state
```

A.2 Die Netzwerk–Datei

Die Netzwerkdatei *(network file)* wird für die LPNN–Berechnungen benötigt. Aus ihr werden die Informationen über die Größe der Gewichtsmatrizen entnommen. Sie beschreibt die Struktur und Topologie der neuronalen Netze und ihre Eingabeschicht. Zuerst wird definiert, welche Merkmalsvektoren der Spracheingabe den einzelnen neuronalen Netzen als Eingabe dienen (hier 2 vergangene und 2 zukünftige Merkmalsvektoren). Dann wird die Anzahl der Neuronen in der Eingabeschicht, der Zwischenschicht und in der Ausgabeschicht. Darauf folgen die Nummern der Neuronen in der Eingabe- und in der Ausgabeschicht. Die einzelnen Schwellwerte aller Neuronen werden durch Verbindungen zu einem Neuron mit der Nummer „Null", mit konstanter Aktivierung „Eins", dargestellt. Neuron „Null" gehört also nicht wirklich zur Eingabeschicht. Es folgen die Listen der gültigen Verbindungen — hierbei bedeutet `a - b : c - d`, daß die Neuronen `a` bis `b` vollständig mit den Neuronen `c` bis `d` verbunden sein sollen. Die unten definierte Topologie ist zum Beispiel dünn vernetzt.

Beispiel einer Netzwerkdatei *(networkfile)*:

```
Version 1 network file.

4 input frames: -2 -1 +1 +2

3 network layers: 64 12 16

in: 0 - 64
out: 77 - 92

0 - 0 : 65 - 92

1 - 5 : 65 - 65
2 - 6 : 66 - 66
3 - 7 : 67 - 67
4 - 8 : 68 - 68
5 - 9 : 69 - 69
6 - 10 : 70 - 70
7 - 11 : 71 - 71
8 - 12 : 72 - 72
9 - 13 : 73 - 73
10 - 14 : 74 - 74
11 - 15 : 75 - 75
12 - 16 : 76 - 76

17 - 21 : 65 - 65
18 - 22 : 66 - 66
19 - 23 : 67 - 67
20 - 24 : 68 - 68
21 - 25 : 69 - 69
22 - 26 : 70 - 70
23 - 27 : 71 - 71
24 - 28 : 72 - 72
25 - 29 : 73 - 73
26 - 30 : 74 - 74
27 - 31 : 75 - 75
28 - 32 : 76 - 76

33 - 37 : 65 - 65
34 - 38 : 66 - 66
35 - 39 : 67 - 67
36 - 40 : 68 - 68
37 - 41 : 69 - 69
38 - 42 : 70 - 70
39 - 43 : 71 - 71
40 - 44 : 72 - 72
41 - 45 : 73 - 73
42 - 46 : 74 - 74
43 - 47 : 75 - 75
44 - 48 : 76 - 76

49 - 53 : 65 - 65
50 - 54 : 66 - 66
51 - 55 : 67 - 67
52 - 56 : 68 - 68
53 - 57 : 69 - 69
54 - 58 : 70 - 70
55 - 59 : 71 - 71
56 - 60 : 72 - 72
57 - 61 : 73 - 73
58 - 62 : 74 - 74
59 - 63 : 75 - 75
60 - 64 : 76 - 76

65 - 69 : 77 - 77
65 - 69 : 78 - 78
65 - 69 : 79 - 79
65 - 69 : 80 - 80
65 - 69 : 81 - 81
66 - 70 : 82 - 82
67 - 71 : 83 - 83
68 - 72 : 84 - 84
69 - 73 : 85 - 85
70 - 74 : 86 - 86
71 - 75 : 87 - 87
72 - 76 : 88 - 88
72 - 76 : 89 - 89
72 - 76 : 90 - 90
72 - 76 : 91 - 91
72 - 76 : 92 - 92
```

A.3 Die Vokabular–Datei

Diese Konfigurationsdatei wird für die DTW–Berechnungen gebraucht. In ihr wird das gültige Vokabular in seiner Phonemumschrift definiert. Die Vokabulardatei *(dictfile)* ist zeilenweise aufgebaut. Jede Zeile enthält eine Aussprachevariante eines Wortes. Links steht das jeweilige Wort, danach folgen die zugehörigen Phonemnamen.

Der Anfang der Vokabular–Datei *(dictfile)*:

```
A                ? AH
AB               ? A P
ABEND            ? AH B E N T
ABENDESSEN       ? AH B E N T ? AE S E N
ABER             ? AH B ER
ABGEDRUCKT       ? A P G E D R U K T
ABGESCHICKT      ? A P G E SCH I K T
ABMELDEN         ? A P M AE L D E N
ABMELDUNG        ? A P M AE L D U NG
ABREISE          ? A P R AI Z E
ACHT             ? A X T
ACHTEN           ? A X T E N
ACHTTAUSEND      ? A X T T AU Z E N T
ACHTZIG          ? A X T S I CH
ADRESSE          ? A D R AE S E
AE               EH
AG               ? AH G EH
AHA              ? A H AH
AKZEPTIERT       ? A K T S AE P T IE R T
ALEX             ? A L AE K S
ALLES            ? A L E S
ALS              ? A L S
AM               ? A M
AN               ? A N
ANBIETEN         ? A N B IE T E N
ANDERS           ? A N D ER S
ANGEGEBEN        ? A N G E G EH B E N
ANKOMMEN         ? A N K O M E N
ANKUENDIGUNG     ? A N K UE N D I G U NG
ANKUNFT          ? A N K U N F T
ANMELDE          ? A N M AE L D E
ANMELDEFORMULAR  ? A N M AE L D E F O R M U L AH R
ANMELDEGEBUEHR   ? A N M AE L D E G E B UEH R
ANMELDEN         ? A N M AE L D E N
ANMELDUNG        ? A N M AE L D U NG
ANSCHRIFT        ? A N SCH R I F T
ANSTELLE         ? A N SCH T AE L E
ANTWORT          ? A N T V O R T
ANZAHL           ? A N T S AH L
ARTIKEL          ? A R T I K E L
AUCH             ? AU X
AUF              ? AU F
AUGENBLICK       ? AU G E N B L I K
AUGUST           ? AU G U S T
AUS              ? AU S
AUSKUNFT         ? AU S K U N F T
AUSMACHEN        ? AU S M A X E N
AUSSCHLIESST     ? AU S SCH L IE S T
B                B EH
BAHN             B AH N
BAHNHOFSHOTEL    B AH N H OH F S H O T AE L
BAHNHOFSPLATZ    B AH N H OH F S P L A T S
BAKENECKER       B AH K E N ? AE K ER
BANK             B A NG K
BANKUEBERWEISUNG B A NG K ? UEH B ER V AI Z U NG
BEANTWORTEN      B E ? A N T V O R T E N
BEGUTACHTET      B E G UH T ? A X T E T
BEHILFLICH       B E H I L F L I CH
```

A.4 Die Phonemumschrift–Datei

Die Phonemumschrift–Datei wird für die Initialisierung der DTW–Berechnungen benötigt. Zur Beschreibung der einzelnen Wörter in der Vokabulardatei von JANUS können weitaus mehr Phoneme verwendet werden, als in der Modelldatei angegeben sind. Im JANUS–Vokabular wird das sogenannte Arphabet benutzt. Um die Vielzahl von Phonemen auf die, in der Modelldatei erlaubten zu reduzieren, und um auch fremde Vokabulardateien verwenden zu können, braucht man Regeln zur Umschrift der Phoneme der Vokabulardatei. Die Phonemumschrift–Datei *(rewritesfile)* legt diese Umschrift fest. Diese Regeln werden iterativ so lange auf die Wörter im Vokabular angewendet, bis sich keine weitere Änderung ergibt. Die so umgeschriebenen Phonemdarstellungen der Wörter werden im DTW benutzt.

Beispiel einer Phonemumschrift–Datei *(rewritesfile)*:

```
NIL     =>      SIL
PH      =>      P
TH      =>      T
KH      =>      K
RJ      =>      R
RX      =>      R
```

Anhang B

Statistik für das deutsche Vokabular

Auf der nächsten Seite befindet sich eine Statistik der Anzahl der Vorgängerworte eines Wortes im Vokabular entstand unter Zuhilfenahme der Wortpaar–Grammatik (siehe Kapitel 5.2.1). In der untersten Zeile sieht man das Wort „Stille“, das 125 mögliche Vorgänger hat. Das bedeutet daß das Prozessorelement welches das Phonem „Stille“ berechnet bei der Verwendung einer Wortpaar–Grammatik eine Liste mit 125 Vorgängern speichern müsste. Diese Unausgewogenheit würde sich nachteilig auf die Rechenzeit auswirken (beim Durchsuchen der Listen). Das kann umgangen werden indem man mehrere Prozessorelemente das Phonem „Stille“ berechnen läßt, die zueinander disjunkte und kürzere Vorgängerlisten bekommen. Die Längste Vorgängerliste wäre somit 29 Wörter lang (siehe unten).

Wörter	Anzahl der Vorgänger
20	0
286	1
70	2
26	3
19	4
13	5
7	6
5	7
4	8
4	9
1	10
4	11
1	12
1	14
1	16
1	17
1	18
1	21
1	29
1	125

Tabelle B.1: Anzahl der gültigen Vorgänger pro Wort

Anhang C

Struktur der Software

Im Anhang D befinden sich die während dieser Arbeit entstandenen Software.

Bei den Dateinamen gelten dabei die folgenden Namenskonventionen bezüglich der Endungen:

- `xyz.h` gemeinsame Definitionsdatei
- `xyz_fe.h` front end Definitionsdatei
- `xyz_fe.c` front end Programm Datei
- `xyz_be.h` back end Definitionsdatei
- `xyz_be.m` back end Programm Datei

Die Software ist wie folgt gegliedert:

- Allgemeine Teile
  - ▷ `ts_std.h` eine allgemeine Definitionsdatei
  - ▷ `timing_be.h` und `timing_be.m` zur Zeitmessung der back end Programme
  - ▷ `GiveUp_fe.c` enthält eine Routine die die MasPar MP-1 Prozeßablaufsteuerung dazu veranlaßt, einen aktiven Prozeß von der MasPar MP-1 zu verdrängen (kommt in die Warteschlange)
  - ▷ `fileIO_be.h` und `fileIO_be.m` bilden ein Modul für maschinenunabhängige Ein- und Ausgabe von Binärdaten
- Signalvorverarbeitung
  - ▷ `ad.h`
  - ▷ `adc_rw.h`
  - ▷ `makeFFT.c`

- Analyse der Sprachsignale mittels LPNNs
  - ▷ `std_fe.c` enthält nur den Aufruf des eigentlichen Haupprogramms auf dem *back end*
  - ▷ `lpnn_be.h` und `lpnn_be.m` bilden das eigentliche Hauptprogramm
  - ▷ `MatVec_be.h` und `MatVec_be.m` bilden ein Modul für Matrix- und Vektor–Operationen
- Suche der N besten Satzhypothesen
  - ▷ `definitions.txt` enthält einige Variablen- und Typen–Konventionen die in diesem Programm gelten
  - ▷ `dp.h` ist eine allgemeine Definitionsdatei
  - ▷ `dp_fe.h` und `dp_fe.c` beinhalten die Initialisierung, die Konvertierung von Datenformaten und das Verteilen der Wörter auf die Prozessorelemente.
  - ▷ `dp_be.h` und `dp_be.m` beinhalten die parallelen Programmteile für das DTW.
  - ▷ `BAWLfiles_fe.h` und `BAWLfiles_fe.c` bilden ein Modul in dem die Konfigurationsdateien genau so wie in JANUS eingelesen werden und in den gleichen Datenstrukturen gespeichert werden. In `dp_be.m` befinden sich einige Routinen, die diese in neue Darstellungen Umrechnen umrechnen.

Anhang D

Software

D.1 Allgemeine Teile

ts_std.h

```c
#ifndef _TS_STD_H_
#define _TS_STD_H_

/*
 * things, they left out of the C "language"  :-)
 *
 * A cardinal number is unsigned.
 *
 * Usually you just need cardinal numbers,
 * you hardly ever need integers.
 */

#ifdef _MPL  /*-- for MasPar's MPL language -----------------------------------------------------------------------

typedef unsigned long long  x_card;    /* really long cardinal  -  8 Bytes */
typedef long long           x_int;     /* really long integer   -  8 Bytes */

typedef unsigned long long  X_CARD;    /* really long cardinal  -  8 Bytes */
typedef long long           X_INT;     /* really long integer   -  8 Bytes */

#define singular

#endif _MPL  /*----------------------------------------------------------------------------------------------------

typedef unsigned long       address;

typedef unsigned char       byte;      /* each byte has 8 bits - you know, these tiny bits ... */

typedef unsigned long       l_card;    /* long  cardinal - 4 Bytes */
typedef unsigned int          card;    /*       cardinal - 4 Bytes */
typedef unsigned short      s_card;    /* short cardinal - 2 Bytes */
typedef unsigned char       t_card;    /* tiny  cardinal - 1 Byte  */

typedef long                l_int;     /* long  integer  - 4 Bytes */
                                       /*       integer  - 4 Bytes */
typedef short               s_int;     /* small integer  - 2 Bytes */
typedef char                t_int;     /* tiny  integer  - 1 Byte  */

typedef char                boolean;
typedef char                bool;      /* synonym for boolean */

/*-- nun GROSS : ----------------------------------------------------------------------------------------------------

typedef unsigned long       ADDRESS;

typedef unsigned char       BYTE;      /* each byte has 8 bits - you know, these tiny bits ... */

typedef unsigned long       L_CARD;    /* long  cardinal - 4 Bytes */
typedef unsigned int          CARD;    /*       cardinal - 4 Bytes */
typedef unsigned short      S_CARD;    /* short cardinal - 2 Bytes */
typedef unsigned char       T_CARD;    /* tiny  cardinal - 1 Byte  */

typedef long                L_INT;     /* long  integer  - 4 Bytes */
                                       /*       integer  - 4 Bytes */
typedef short               S_INT;     /* small integer  - 2 Bytes */
typedef char                T_INT;     /* tiny  integer  - 1 Byte  */

typedef char                BOOLEAN;
typedef char                BOOL;      /* synonym for boolean */

/*-----------------------------------------------------------------------------------------------------------------

#define FALSE   0
#define TRUE    1


#define NOT     !
#define AND     &&
#define OR      ||

#define BitAnd  &
#define BitOr   |
#define BitExor ^
#define BitComp ~                                     /* complement */
#define BitNeg  -                                     /* negation   */

#define NIL     0
#define EOS 0 /* denotes end of string */

#define then
#define elif    else if

#define mod     %
#define MOD     %
#define div     /
#define DIV     /


/*-- Macros : -------------------------------------------------------------------------------------------------------

#define EVEN(a)  (((a % 2)==0) ? (TRUE) : (FALSE))
#define ODD(a)   (((a % 2)==1) ? (TRUE) : (FALSE))
```

```
#define MAX(a,b) (((a)>(b)) ?  (a) : (b))
#define MIN(a,b) (((a)<(b)) ?  (a) : (b))
#define ABS(x)   (((x)> 0 ) ? -(x) : (x))

#define streq(s1,s2)    (strcmp (s1,s2) == 0)          /* string equality   */
#define strneq(s1,s2)   (strcmp (s1,s2) != 0)          /* string inequality */

/*-- assertions : ------------------------------------------------------------------------------------------------------

#ifndef NODEBUG

#define Assert(expr,str)\
  if (!(expr))\
       fprintf(stderr,"\n >> ASSERTION FAILED !! %s - in file '%s' at line %d <<\n\n", str, __FILE__, __LINE__);

#define AssertAndExit(expr,str,exit_value)\
  if (!(expr))\
     { fprintf(stderr,"\n >> ASSERTION FAILED !! %s - in file '%s' at line %d <<\n\n", str, __FILE__, __LINE__);\
       fprintf(stderr,"\n ###### EXITING WITH RETURN VALUE %d ######\n", exit_value);\
       exit(exit_value);\
     }

#else
#define Assert(e)
#define AssertAndExit(expr,str,ev)

#endif NODEBUG

#endif _TS_STD_H_
```

timing_be.h

```
/*##########################################################################################################################################
 #
 #  timing.h                created 1991 by Tilo Sloboda  (sloboda@ira.uka.de)
 #
 # Note :       timing.h   and   timing_be.h
 #              timing.c   and   timing_be.m
 #
 #    Are identical copies of each other ! The only possible difference is the include <time.h>
 #
 #
 # Description : universal module for timing   ;   using gettimeofday()
 #
 # usage :
 #
 #  static double   duration = 0.0;   >> duration isn't used in this example <<
 #  static Timer    MyTimer;
 #
 #
 #  reset_timer(&MyTimer);
 #  start_timer(&MyTimer);
 #  doit;
 #  duration = stop_timer(&MyTimer);   >> duration isn't used in this example <<
 #
 #  print_timer("elapsed time: ", &MyTimer);
 #
 #
 # Last changes :
 #  16.Sept 91  Tilo   created
 #  17.Sept 91  Tilo   changed timers to be restartable  (cummulative timing)
 #                     and added reset_timer(), print_timer() routines.
 #  24.Sept 91  Tilo   replaced the Timer-Array and get_timerId ... by a new Timer type
 #                     so now one has just to define as many timers, as he wants ...
 #                     BEWARE of the new VAR-parameters "t" !!!
 #   6.Oct  91  Tilo   minor bugs removed and documentation corrected.
 #
 #-- RCS Info ---------------------------------------------------------------------------------------------------------------------------
 #
 # $RCSfile: timing.h,v $     $Revision: 1.5 $        $State: Exp $
 # $Date: 1991/10/06 18:36:24 $    $Author: sloboda $      $Locker:  $
 #
 # $Log: timing.h,v $
 # Revision 1.5  1991/10/06  18:36:24  sloboda
 # documentation now correct.
 #
 # Revision 1.4  1991/09/24  00:59:09  sloboda
 #  replaced the Timer-Array and get_timerId ... by a new Timer type
 #  so now one has just to define as many timers, as he wants ...
 #  BEWARE of the new VAR-parameters "t" !!!
 #
 # Revision 1.3  91/09/17  22:34:43  sloboda
 # working version ...
 #
 # Revision 1.2  91/09/17  22:12:16  sloboda
 # cummulative timing now possible, print_timer(), reset_timer() routines added.
 #
 # Revision 1.1  91/09/17  21:44:35  sloboda
 # Initial revision
 #
 #
 ##########################################################################################################################################*/

#ifndef _TIMING_H_
#define _TIMING_H_

/*== Includes ============================================================================================================================*/

#include <sys/time.h>           /* um Zeitmessungen zu machen ... */
                                /* an der MasPar evtl nur time.h  */
#include "ts_std.h"


/*== Public Constants ====================================================================================================================*/

/*== Public Types ========================================================================================================================*/

typedef struct
  {
    struct timeval  start;
    struct timeval  stop;
    struct timezone zone;
    double          delta;
  }
    Timer;

/*== Public Variables ====================================================================================================================*/

/*##########################################################################################################################################
 #
 #      PUBLIC ROUTINES
 #
 ##########################################################################################################################################*/

extern void     reset_timer ();
```

timing_be.m

```
/*#########################################################################################################################################
 #
 #  timing_be.m          created 1991 by Tilo Sloboda  (sloboda@ira.uka.de)
 #
 # Note :       timing.h   and   timing_be.h
 #              timing.c   and   timing_be.m
 #
 #    Are identical copies of each other ! The only possible difference is the include <time.h> and "timing.h"
 #
 #
 # Description : universal module for timing   ;   using gettimeofday()
 #
 # usage :
 #
 #  static double   duration = 0.0;   >> duration isn't used in this example <<
 #  static Timer    MyTimer;
 #
 #
 #  reset_timer(&MyTimer);
 #  start_timer(&MyTimer);
 #  doit;
 #  duration = stop_timer(&MyTimer);   >> duration isn't used in this example <<
 #
 #  print_timer("elapsed time: ", &MyTimer);
 #
 #
 # Last changes :
 #  16.Sept 91  Tilo   created
 #  17.Sept 91  Tilo   changed timers to be restartable  (cummulative timing),
 #                     and added reset_timer(), print_timer() routines.
 #  24.Sept 91  Tilo   replaced the Timer-Array and get_timerId ... by a new Timer type
 #                     so now one has just to define as many timers, as he wants ...
 #                     BEWARE of the new VAR-parameters "t" !!!
 #   6.Oct  91  Tilo   minor bugs removed and documentation corrected.
 #
 #-- RCS Info ---------------------------------------------------------------------------------------------------------------------------
 #
 # $RCSfile: timing.c,v $     $Revision: 1.5 $        $State: Exp $
 # $Date: 1991/10/06 18:36:58 $    $Author: sloboda $      $Locker:  $
 #
 # $Log: timing.c,v $
 # Revision 1.5  1991/10/06  18:36:58  sloboda
 # minor bugs removed, documentation now correct
 #
 # Revision 1.4  1991/09/24  00:58:32  sloboda
 #  replaced the Timer-Array and get_timerId ... by a new Timer type
 #  so now one has just to define as many timers, as he wants ...
 #  BEWARE of the new VAR-parameters "t" !!!
 #
 # Revision 1.3  91/09/17  22:34:57  sloboda
 # working version ...
 #
 # Revision 1.2  91/09/17  22:13:23  sloboda
 #  cummulative timing now possible, print_timer(), reset_timer() routines added.
 #
 # Revision 1.1  91/09/17  21:44:30  sloboda
 # Initial revision
 #
 #
 #########################################################################################################################################*

/*== Includes ============================================================================================================================*

#include <sys/time.h>                       /* um Zeitmessungen zu machen ... */
                                            /* an der MasPar evtl nur time.h  */
#include <stdio.h>                          /* NUR FUER DIE printf's */
#include "ts_std.h"
#include "timing_be.h"                      /* "timing_be.h"  fuer die MasPar */

/*== Private Constants ===================================================================================================================*
/*== Private Types =======================================================================================================================*
/*== Private Variables ===================================================================================================================*

static char    RCSid[] = "$header$";

/*== Public Variables ====================================================================================================================*


/*#########################################################################################################################################
 #
 #      PUBLIC ROUTINES
 #
 #########################################################################################################################################*

/*=========================================================================================================================================
  |
  | reset_timer
  |
  | Parameters :
  |   IN : &t     Timer
  |
  | Description :
  |   resets the timer t to zero.
```

```
|
| History :
|   17.Sept.91  Tilo     created
|   24.Sept 91  Tilo     changed it for Timer type
|
+=============================================================================================================================

void reset_timer (t)
    Timer  *t;
{
  t->start.tv_sec = 0L;
  t->start.tv_usec= 0L;
  t->stop.tv_sec  = 0L;
  t->stop.tv_usec = 0L;
  t->delta        = 0.0;
}

/*============================================================================================================================
|
| start_timer
|
| Parameters :
|   IN : &t     Timer
|
| Description :
|   starts the timer t
|
| History :
|   16.Sept.91  Tilo     created
|   24.Sept 91  Tilo     changed it for Timer type
|
+=============================================================================================================================

void start_timer(t)
   Timer  *t;
{
   struct timeval   * tp;
   struct timezone  * tz;

 tp = &(t->start);  tz = &(t->zone);

 if (-1 == gettimeofday(tp, tz))
    then printf(" ERROR 1 IN start_timer !\n ");
    else if (tp == NULL)
            then printf(" ERROR 2 IN start_timer !\n ");
}

/*============================================================================================================================
|
| stop_timer
|
| Parameters :
|   IN : &t     Timer
|
| Returns :
|   time elapsed in seconds between start_timer() and stop_timer() call.
|
| History :
|   16.Sept.91  Tilo     created
|   17.Sept 91  Tilo     changed, to enable cummulative timing
|   24.Sept 91  Tilo     changed it for Timer type
|
+=============================================================================================================================

double stop_timer(t)
   Timer  *t;
{
   long             s, us;
   struct timeval   * tp;
   struct timezone  * tz;
   double           result;     /* for MasPar ... */

 tp = &(t->stop); tz = &(t->zone);

 if (-1 == gettimeofday(tp, tz))
    then printf(" ERROR 1 IN stop_timer !\n");
    else if (tp == NULL)
            then printf(" ERROR 2 IN stop_timer !\n");

            else {
#ifdef _DEBUG_TIMING_
                  printf("\nstart.s =%ld\t start.us=%ld\n", t->start.tv_sec, t->start.tv_usec);
                  printf("stop.s =%ld\t stop.us=%ld\n\n",   t->stop.tv_sec,  t->stop.tv_usec);
#endif
                  s  = (t->stop.tv_sec)  - (t->start.tv_sec);
                  us = (t->stop.tv_usec) - (t->start.tv_usec);

                  while (us < 0L)
                      {us += 1000000L; s -= 1L;}
                  t->delta += ((double)s+((double)us/1E6));         /*  += for cummulative timing */
#ifdef _DEBUG_TIMING_
                  printf("        %ld sec\t%ld usec\t==> %ld usec\t%10.6f sec\n",
                         s,us, (us + 1000000*s), t->delta);
#endif
                }
 result = t->delta;     /* for MasPar ... */
```

```
 return (result);
}

/*+=============================================================================================================================
  |
  | print_timer
  |
  | Parameters :
  |   IN : &t     timerId
  |        str   comment string
  |
  | Returns :
  |   prints a comment and the time elapsed in seconds, which was accumulated by this timer.
  |
  | History :
  |   17.Sept 91  Tilo     created
  |   24.Sept 91  Tilo     changed it for Timer type
  |
  +=============================================================================================================================*/
void print_timer (message, t)
    char    * message;
    Timer   * t;
{
  printf ("%s %10.6f seconds\n", message, t->delta);
}

/*-----------------------------------------------------------------------------------------------------------------------------*/
```

GiveUp_fe.c

```
/*#############################################################################################################################
 #
 # GiveUp_fe.c          help for the MasPar job scheduler
 #
 #-- RCS Info ---------------------------------------------------------------------------------------------------------------
 #
 # $RCSfile: GiveUp_fe.c,v $     $Revision: 1.1 $        $State: Exp $
 # $Date: 1991/11/29 12:36:44 $    $Author: sloboda $      $Locker:  $
 #
 # $Log: GiveUp_fe.c,v $
 * Revision 1.1  1991/11/29  12:36:44  sloboda
 * Initial revision
 *
 #
 #############################################################################################################################*/

#include <sys/types.h>
#include <signal.h>

/*   see man pages :
 *        man 2 kill
 *        man 2 getppid
 *        man 2 sigvec
 */

int GiveUp()  /* send a "give up" signal to the parent process */
{
  kill(getppid(), SIGUSR2);
}
```

fileIO_be.h

```
/*#####################################################################################################################
 #
 # fileIO_be.h                created 1991 by Tilo Sloboda  (sloboda@ira.uka.de)
 #
 #   This software is part of the parallel JANUS software, based on Joe Tebelskis "routines.c" for the LPNN JANUS system.
 #
 # Description :
 #   Support for reading/writing canonical binary files on any type of machine
 #
 #   This stuff is based on Joe Tebelskis file IO routines, he used in "routines.c" for JANUS.
 #   The basic idea is to use some convertion routines between the assumed ideal dataformat and the actual,
 #   machine dependent dataformat. So everything is stored like on a SUN (highest bit and byte first).
 #   Strings are stored a little bit strange ... after a leading short whith the number of bytes in the string
 #   (incl. the null-char at the end) the string is stored.
 #
 #   Three different formats of byte representations are known (so far) :
 #
 #   SUN  :   position :  0   1   2   3
 #            bytes    : 'a' 'b' 'c' 'd'
 #
 #   DEC  :   position :  0   1   2   3
 #            bytes    : 'd' 'c' 'b' 'a'
 #
 #   VAX  :   position :  0   1   2   3
 #            bytes    : 'b' 'a' 'd' 'c'
 #
 #   The VAXes have (additional) a different floating representation. So some extra effort is necessary to
 #   transfer the float formats.
 #
 #   For the MasPar-fileIO routines, just DEC like and VAX like io is interessting.
 #
 # Note :
 #   check_machine_type() should be called first in any program, that uses this module
 #
 # Last changes :
 #   26.Sept 91  14:40  tilo    now you have to define one of DEC, VAX to the compiler by mpl_cc-option -D
 #                              to compile code for this machine ( i.e.  cc -DVAX ... ) , default is -DDEC.
 #   10.Oct  91  20:45  Tilo    new macros FLOAT_SWAP, p_FLOAT_SWAP , SUN option available , no default anymore.
 #
 #-- RCS Info ---------------------------------------------------------------------------------------------------------
 #
 # $RCSfile: fileIO_be.h,v $     $Revision: 1.3 $        $State: Exp $
 # $Date: 1991/11/29 12:30:57 $    $Author: sloboda $      $Locker:  $
 #
 # $Log: fileIO_be.h,v $
 # Revision 1.3  1991/11/29  12:30:57  sloboda
 # the byte swapping for floats is now handled by the macro FloatSwap.
 # DO NOT use the read- and write-Float routines.
 #
 # Revision 1.2  1991/10/10  12:40:26  sloboda
 # now you have to define one of DEC, VAX to the compiler by mpl_cc-option -D
 # to compile code for this machine ( i.e.  cc -DVAX ... )
 #
 # Revision 1.1  91/07/15  20:54:08  sloboda
 # Initial revision
 #
 #####################################################################################################################*/

#ifndef _FILEIO_BE_H_
#define _FILEIO_BE_H_

/*== Includes ======================================================================================================*/

#include <errno.h>

/*== Public Constants ==============================================================================================*/

/*== Public Types ==================================================================================================*/

/*== Public Variables ==============================================================================================*/

/*== Private Types =================================================================================================*/

/* this type is just for internal use in the following macros ...
   I'll hide this type later !
 */

typedef
union                           /* union for type conversions */
  { float fval;                         /*    4-byte float  */
    int   ival;                         /*    4-byte int    */
    short sval;                         /*    2-byte short  */
    unsigned char cval [4];             /*    4-byte string */
  }
    unionT;

/*#####################################################################################################################
 #
 #      PUBLIC MACROS
 #
 #####################################################################################################################*/

#ifdef VAX
```

```
# define READ_FLOAT_SWAP(f) \
 {   unionT u,v;\
   u.fval = (float)f;\
   v.cval[0] = u.cval[1];                                  /* VAX: shuffle the bytes       */\
   v.cval[1] = (u.cval[0]==0 ? 0 : u.cval[0] + 1);         /* adjust to weird'o VAX format */\
   v.cval[2] = u.cval[3];\
   v.cval[3] = u.cval[2];\
   f = v.fval;\
 }
# define p_READ_FLOAT_SWAP(f) \
 {   plural unionT u,v;\
   u.fval = (plural float)f;\
   v.cval[0] = u.cval[1];                                  /* VAX: shuffle the bytes       */\
   v.cval[1] = (u.cval[0]==0 ? 0 : u.cval[0] + 1);         /* adjust to weird'o VAX format */\
   v.cval[2] = u.cval[3];\
   v.cval[3] = u.cval[2];\
   f = v.fval;\
 }
# define WRITE_FLOAT_SWAP(f) \
 {   unionT u,v;\
   u.fval = (float)f;\
   v.cval[0] = (u.cval[1]==0 ? 0 : u.cval[1]-1); /* VAX:  shuffle the bytes */\
   v.cval[1] = u.cval[0];                        /* adjust to weird'o VAX format */\
   v.cval[2] = u.cval[3];\
   v.cval[3] = u.cval[2];\
   f = v.fval;\
 }
# define p_WRITE_FLOAT_SWAP(f) \
 {   plural unionT u,v;\
   u.fval = (plural float)f;\
   v.cval[0] = (u.cval[1]==0 ? 0 : u.cval[1]-1); /* VAX:  shuffle the bytes */\
   v.cval[1] = u.cval[0];                        /* adjust to weird'o VAX format */\
   v.cval[2] = u.cval[3];\
   v.cval[3] = u.cval[2];\
   f = v.fval;\
 }
#else
#ifdef DEC
# define READ_FLOAT_SWAP(f) \
 {   unionT u,v;\
   u.fval = (float)f;\
   v.cval[0] = u.cval[3];          /* DEC:  reverse the bytes */\
   v.cval[1] = u.cval[2];\
   v.cval[2] = u.cval[1];\
   v.cval[3] = u.cval[0];\
   f = v.fval;\
 }
# define p_READ_FLOAT_SWAP(f) \
 {   plural unionT u,v;\
   u.fval = (plural float)f;\
   v.cval[0] = u.cval[3];          /* DEC:  reverse the bytes */\
   v.cval[1] = u.cval[2];\
   v.cval[2] = u.cval[1];\
   v.cval[3] = u.cval[0];\
   f = v.fval;\
 }
# define WRITE_FLOAT_SWAP(f) \
 {   unionT u,v;\
   u.fval = (float)f;\
   v.cval[0] = u.cval[3];                    /* DEC:  reverse the bytes */\
   v.cval[1] = u.cval[2];\
   v.cval[2] = u.cval[1];\
   v.cval[3] = u.cval[0];\
   f = v.fval;\
 }
# define p_WRITE_FLOAT_SWAP(f) \
 {   plural unionT u,v;\
   u.fval = (plural float)f;\
   v.cval[0] = u.cval[3];                    /* DEC:  reverse the bytes */\
   v.cval[1] = u.cval[2];\
   v.cval[2] = u.cval[1];\
   v.cval[3] = u.cval[0];\
   f = v.fval;\
 }
#else
#ifdef SUN
# define READ_FLOAT_SWAP(f)
# define p_READ_FLOAT_SWAP(f)
# define WRITE_FLOAT_SWAP(f)
# define p_WRITE_FLOAT_SWAP(f)
#else
# define READ_FLOAT_SWAP(f)    printf(stderr, " U N K N O W N   M A C H I N E  ! \n");
# define p_READ_FLOAT_SWAP(f)  printf(stderr, " U N K N O W N   M A C H I N E  ! \n");
# define WRITE_FLOAT_SWAP(f)   printf(stderr, " U N K N O W N   M A C H I N E  ! \n");
# define p_WRITE_FLOAT_SWAP(f) printf(stderr, " U N K N O W N   M A C H I N E  ! \n");
#endif
#endif
#endif

/*#########################################################################################################################################
 #
 #      PUBLIC ROUTINES
 #
 #########################################################################################################################################*/
```

```
extern void  check_machine_type();
extern void  freadOK();
extern void  readOK();

extern void  pp_write_float();
extern void  p_write_float();
extern void  write_float();

extern plural float pp_read_float();
extern plural float p_read_float();
extern float        read_float();

extern short  read_short();
extern void   read_string();
#endif
```

fileIO_be.m

```
/*****************************************************************************************************************************
 #
 # fileIO_be.m                created 1991 by Tilo Sloboda  (sloboda@ira.uka.de)
 #
 #   This software is part of the parallel JANUS software, based on Joe Tebelskis "routines.c" for the LPNN JANUS system.
 #
 # Description :
 #   Support for reading/writing canonical binary files on any type of machine
 #
 #   This stuff is based on Joe Tebelskis file IO routines, he used in "routines.c" for JANUS.
 #   The basic idea is to use some convertion routines between the assumed ideal dataformat and the actual,
 #   machine dependent dataformat. So everything is stored like on a SUN (highest bit and byte first).
 #   Strings are stored a little bit strange ... after a leading short whith the number of bytes in the string
 #   (incl. the null-char at the end) the string is stored.
 #
 #   Three different formats of byte representations are known (so far) :
 #
 #   SUN  :   position :  0   1   2   3
 #            bytes    : 'a' 'b' 'c' 'd'
 #
 #   DEC  :   position :  0   1   2   3
 #            bytes    : 'd' 'c' 'b' 'a'
 #
 #   VAX  :   position :  0   1   2   3
 #            bytes    : 'b' 'a' 'd' 'c'
 #
 #   The VAXes have (additional) a different floating representation. So some extra effort is necessary to
 #   transfer the float formats.
 #
 #   For the MasPar-fileIO routines, just DEC like and VAX like io is interessting.
 #
 # Note :
 #   check_machine_type() should be called first in any program, that uses this module
 #
 # Last changes :
 #   26.Sept 91  14:40  tilo    now you have to define one of DEC, VAX to the compiler by mpl_cc-option -D
 #                              to compile code for this machine ( i.e.  cc -DVAX ... ) , default is -DDEC.
 #   10.Oct  91  21:50  Tilo    no default anymore ; MasPar front end can be a SUN-like machine too (if ever available).
 #
 #-- RCS Info ------------------------------------------------------------------------------------------------------------------
 #
 # $RCSfile: fileIO_be.m,v $     $Revision: 1.3 $        $State: Exp $
 # $Date: 1991/11/29 12:32:10 $    $Author: sloboda $      $Locker:  $
 #
 # $Log: fileIO_be.m,v $
 # Revision 1.3  1991/11/29  12:32:10  sloboda
 # the byte swapping for floats is now handled by the macro FloatSwap.
 # DO NOT use the read- and write-Float routines.
 #
 # Revision 1.2  1991/10/10  12:41:04  sloboda
 # now you have to define one of DEC, VAX to the compiler by mpl_cc-option -D
 # to compile code for this machine ( i.e.  cc -DVAX ... )
 #
 # Revision 1.1  91/07/15  20:54:21  sloboda
 # Initial revision
 #
 *****************************************************************************************************************************/

/*+============================================================================================================================+
  |  MEMO :                                                                                                                    |
  |          The routine p_write_float  and  pp_write_float (maybe  write_float  too)  are   UNSTABLE !!! they crash !         |
  |                                                                                                                            |
  |          DON'T USE THEM !   cause : (plural double) insead of (plural float) is assumed by the mpl_cc for fct.parameters  |
  +============================================================================================================================+*/

/*== Includes ================================================================================================================*/

#include <stdio.h>
#include <string.h>

#include "fileIO_be.h"
#include "ts_std.h"

/*== Private Constants =======================================================================================================*/

#ifdef DEC
#  define COMPILED_AS "DEC"
#else
#ifdef VAX
#  define COMPILED_AS "VAX"
#else
#ifdef SUN
#  define COMPILED_AS "SUN"     /* oooops ??? a SUN-like MasPar front end ??? we'll let them, if they have one ! :-) */
#else
#  define COMPILED_AS "???"     /* no default anymore */
#endif
#endif
#endif

/*== Private Types ===========================================================================================================*/

/*== Private Variables =======================================================================================================*/
```

```
static  char    RCSid[] = "$header$";

static  int     SizeOfFloat = sizeof(float),            /* = sizeof(float) only calculated at the beginning */
                SizeOfShort = sizeof(short);

/*== Public Variables ===========================================================================================================

/*#############################################################################################################################
 #
 #      PUBLIC DEBUGGING ROUTINES
 #
 #############################################################################################################################*

/*=============================================================================================================================
 |
 | check_machine_type()            EXPORTED
 |
 | Description  :  checks, whether the type of machine we're running on is the same, as the program was compiled for.
 |                 if so, it returns the name of the machine in the string "MachType"
 |                 if not, it prints a error message to stderr and exits the program with value -1.
 |
 | Parameters   :  none
 |
 | History :
 |   26.Sept 91 tilo   created, based on jmt's get_machine_type()
 |
 +============================================================================================================================*

void check_machine_type (MachType)
     char    *MachType;
{
     unionT   u;


 if (sizeof(float) != 4) {fprintf (stderr, "\n>>> PROBLEM: sizeof float = %d <<<\n", sizeof(float)); exit(-1);}
 if (sizeof(int)   != 4) {fprintf (stderr, "\n>>> PROBLEM: sizeof int   = %d <<<\n", sizeof(int));   exit(-1);}
 if (sizeof(short) != 2) {fprintf (stderr, "\n>>> PROBLEM: sizeof short = %d <<<\n", sizeof(short)); exit(-1);}

 u.fval = 123.456; /* assign a known float.  How is it represented? */

 if      (u.cval[3] == 121)                      /* if 121 is in byte #3, we're on a RT or SUN    */
         then strcpy(MachType,"SUN");            /* wow, they have a SUN-like front end !!! :-)   */
 else if (u.cval[0] == 121)                      /* if 121 is in byte #0, we're on a DEC          */
         then strcpy(MachType,"DEC");
 else if (u.cval[2] == 121)                      /* if 121 is in byte #2, we're on a VAX          */
         then strcpy(MachType,"VAX");
 else                                            /* unknown kind of machine                       */
       { strcpy(MachType," - U N K N O W N   M A C H I N E - ");
         fprintf(stderr, "\n>>> ERROR : %s in \"__FILE__\"", MachType);
         exit(-1);
       }

 /*
  * printf ("\n%s %s\n", COMPILED_AS, MachType);
  */

 if strneq(COMPILED_AS,MachType) then
    { fprintf(stderr, "\n>>> ERROR : recompile this program with cc-option \"-D%s\"\n\n", MachType);
      exit(-1);
    }
}

/*#############################################################################################################################
 #
 #      PUBLIC ROUTINES   (which should be transformed into macros)
 #
 #############################################################################################################################*

/*=============================================================================================================================
 |
 | freadOK                 PRIVATE   SINGULAR
 |
 | Description :
 |    Checks the result of fscanf; if there was a problem, abort the program.
 |
 | Parameter:
 |   result   result of a prior call to fscanf.
 |
 | History:
 |   3.July 89  jmt   Created.
 |   8.July 91  tilo  minor change      (old name : readOK)
 +============================================================================================================================*

void

freadOK (result)
int result;
{
  if (result == EOF) { fprintf(stderr, "\n >>> unexpected EOF !!\n"); exit (0); }
}

/*=============================================================================================================================
 |
 | readOK                  PRIVATE   SINGULAR
 |
```

```
  | Description :
  |    Checks the result of basic UNIX-i/o routines; if there was a problem, abort the program.
  |
  | Parameter:
  |   result   result of a prior call to UNIX-i/o routine.
  |   str      a string containig some information about the context, is given to perror
  |
  | History:
  |  19.July 91  tilo  created
  +=============================================================================================================================

void

readOK (result,str)
   int    result;
   char * str;
{
  if (result == -1) { perror(str); exit (-1); }
}

/*#############################################################################################################################
 #
 #      PUBLIC ROUTINES
 #
 #############################################################################################################################*/

/*=============================================================================================================================
  | read_short:  Reads a short integer from a binary file.  Works on any machine.
  |
  | Parameters:  fd = binary file pointer.  (Assumed to be already open.)
  |
  | Returns:     The next short integer in the file.
  |
  | History:
  |   20.Oct  89  jmt   Created.
  |   19.July 91  tilo  changed it to low level file i/o, and just for the VAX ...
  +============================================================================================================================*/

short read_short (fd)
    int    fd;
{
    unionT u,v;

  readOK( read (fd, &(u.sval), SizeOfShort) , " in read_short");  /* read the next 2 bytes */
  v.cval[0] = u.cval[1];                                          /* DEC or VAX:  reverse the bytes */
  v.cval[1] = u.cval[0];
  return (v.sval);
}

/*=============================================================================================================================
  | read_string:  Reads a string from a UNIX file, where it is prefixed with its length and terminated with EOS.
  |
  | Parameters:
  |   f   = binary file to read from.  The file must be already open.
  |   str = address of destination string.
  |  &len = number of bytes, that was processed.
  |
  | History:
  |   12.Oct  89  jmt    Created.
  |   19.Jan  90  jmt    Use read_short, not fread, to read the length.
  |   19.July 91  tilo   changed it to low level file i/o
  +============================================================================================================================*/

void read_string (fd, str, bytes)
    int   fd, *bytes;
    char *str;
{
    short len;

  len = read_short (fd);          /* read length of string */
  if (len % 256 == 0)  len = len >> 8;                  /* kludgy bug fix: swap bytes if incorrect format from old days */
  readOK(read (fd, str, len + 1), " in read_string" ); /* read that many characters (plus EOS) into string */
  *bytes = len + 1 + SizeOfShort;
}

/*#############################################################################################################################
 #
 #      PUBLIC ROUTINES   (not used anymore !  24.Oct 91)
 #
 #############################################################################################################################*/

/*=============================================================================================================================
  | pp_write_float:  Writes floats parallel into a binary file, with parallel filepointers.  works only on VAXes.
  |
  | Parameters:      fd = binary file descriptor.  (Assumed to be already open.)
  |
  | History:
  |   19.July 91  tilo   created, for low level file i/o, and just for the VAX ... and parallel
  |   24.Sept 91  tilo   changed it to run on a MasPar with DEC front end.
  |   24.Oct  91  tilo   now the FLOAT_SWAP macros are used, for simplification.
  +============================================================================================================================*/

void pp_write_float (fd, f)
    int          fd;
    plural float f;
{
```

```
  p_WRITE_FLOAT_SWAP(f);

  pp_write (fd, &f, SizeOfFloat);
}

/*=============================================================================================================================
 | p_write_float:  Writes floats parallel into a binary file, one active PE after the other.  works only on VAXes.
 |
 | Parameters:     fd = binary file descriptor.  (Assumed to be already open.)
 |
 | History:
 |   20.July 91  tilo   created, for low level file i/o, and just for the VAX ... and parallel
 |   24.Sept 91  tilo   changed it to run on a MasPar with DEC front end.
 |   24.Oct  91  tilo   now the FLOAT_SWAP macros are used, for simplification.
 +============================================================================================================================*/

void p_write_float (fd, f)
     int           fd;
     plural float f;
{
  p_WRITE_FLOAT_SWAP(f);

  p_write (fd, &f, SizeOfFloat);
}

/*=============================================================================================================================
 | write_float:  Writes a float into a binary file.  works only on VAXes.
 |
 | Parameters:   fd = binary file descriptor.  (Assumed to be already open.)
 |
 | History:
 |   20.Oct  89  jmt    Created.
 |   19.July 91  tilo   changed it to low level file i/o, and just for the VAX ...
 |   24.Sept 91  tilo   changed it to run on a MasPar with DEC front end.
 |   24.Oct  91  tilo   now the FLOAT_SWAP macros are used, for simplification.
 +============================================================================================================================*/

void write_float (fd, f)
     int   fd;
     float f;
{
  WRITE_FLOAT_SWAP(f);

  write (fd, &f, SizeOfFloat);
}

/*=============================================================================================================================
 | pp_read_float:  Read floats parallel from a binary file, with parallel filepointers . works only on VAXes.
 |
 | Parameters:     fd = binary file descriptor.  (Assumed to be already open.)
 |
 | Returns:        The next float in the file.
 |
 | History:
 |   19.July 91  tilo   created, for low level file i/o, and just for the VAX ... and parallel
 |   24.Sept 91  tilo   changed it to run on a MasPar with DEC front end.
 |   24.Oct  91  tilo   now the FLOAT_SWAP macros are used, for simplification.
 +============================================================================================================================*/

plural float pp_read_float (fd)
    int  fd;
{
    plural float f;

  readOK( pp_read (fd, &f, SizeOfFloat) , " in read_float");  /* read the next 4 bytes        */

  p_READ_FLOAT_SWAP(f);
  return(f);
}

/*=============================================================================================================================
 | p_read_float:  Read floats parallel from a binary file, one PE after the other. works only on VAXes.
 |
 | Parameters:    fd = binary file descriptor.  (Assumed to be already open.)
 |
 | Returns:       The next float in the file.
 |
 | History:
 |   21.July 91  tilo   created for low level file i/o, and just for the VAX ... and parallel
 |   24.Sept 91  tilo   changed it to run on a MasPar with DEC front end.
 |   24.Oct  91  tilo   now the FLOAT_SWAP macros are used, for simplification.
 +============================================================================================================================*/

plural float p_read_float (fd)
    int  fd;
{
    plural float f;

  readOK( p_read (fd, &f, SizeOfFloat) , " in read_float");  /* read the next 4 bytes        */

  p_READ_FLOAT_SWAP(f);
  return(f);
}

/*=============================================================================================================================
 | read_float:  Reads a float from a binary file. works only on VAXes.
```

```
|
| Parameters:  fd = binary file descriptor.  (Assumed to be already open.)
|
| Returns:     The next float in the file.
|
| History:
|   20.Oct  89  jmt    Created.
|   19.July 91  tilo   changed it to low level file i/o, and just for the VAX ...
|   24.Sept 91  tilo   changed it to run on a MasPar with DEC front end.
|   24.Oct  91  tilo   now the FLOAT_SWAP macros are used, for simplification.
+-------------------------------------------------------------------------------------------------------------------------------

float read_float (fd)
    int  fd;
{
    float f;

  readOK( read (fd, &f, SizeOfFloat) , " in read_float");  /* read the next 4 bytes        */

  READ_FLOAT_SWAP(f);
  return(f);
}
/*-------------------------------------------------------------------------------------------------------------------------------
```

ad.h

```
/*-------------------------------------------------------------*
 * AD.H - ad header description
 *-------------------------------------------------------------*
 * HISTORY
 */

#ifndef _AD_H_
#define _AD_H_

struct ad_head {
    short ad_hdrsize; /* Size of header, including this
          * this field, in short words */
    short ad_version;
    short ad_channels;
    short ad_rate; /* In quarter usec */
    int   ad_samples;
    int   little_indian; /* True if least significant byte is
 * byte 0, ie. Vax byteorder */
};

#define CURRENT_AD_VERSION 1

#define ADA_RANGE (1<<16)
#define QUS_PER_MS 4000 /* Quarter usec / msec */

#define SAMPS_PER_MS(r) (QUS_PER_MS/(r))

typedef struct ad_head ad_head_t;

#endif _AD_H_
```

adc_rw.h

```
/* AD_READ - read an adcfile
 *------------------------------------------------------------*
 * HISTORY
 * 17-Nov-87  Fil Alleva (faa) at Carnegie-Mellon University
 * Changed so that binaries can read and written with out
 * regard to byte order problems.
 *
 *  6-Nov-86  Fil Alleva (faa) at Carnegie-Mellon University
 * Changed not to allocate mem if *buf is != 0.
 *
 *  9-Mar-83  Fil Alleva (faa) at Carnegie-Mellon University
 * Modified to read new file format and place the buffer on a page
 * boundary.
 *
 */

#include "ad.h"
#include <sys/types.h>
#include <sys/file.h>
#include <sys/stat.h>
#include <stdio.h>

#define TRUE 1
#define FALSE 0

#define SWAPW(x) (((x)<<8) | (0xFF & ((x)>>8)))
#define SWAPL(x) ((((x)<<24)&0xFF000000) | (((x)<<8)&0x00FF0000) | \
 (((x)>>8) & 0x0000FF00) | (((x)>>24)&0x000000FF))

/* LITTLE_INDIAN - returns non 0 if this is a little indian machine.
 *------------------------------------------------------------*
 * DESCRIPTION
 * Returns non-zero when this code is compiled and run on
 * a machine that formats shorts and integers with the least significant
 * byte at address 0. Otherwise it returns 0.
 */
static
little_indian ()
{
    char b[4];
    register long *l = (long *) b;

    *l = 1;
    return ((int) b[0]);

}

/* AD_READ - read an adc file
 *------------------------------------------------------------*
 */
ad_read (dir, file, buf, head)
char *dir;
char *file;
short **buf;
register ad_head_t *head;
{
    register        fd;
    register short *aptr;
    register int    do_byte_swap = FALSE;
    char            fullname[1024];
    struct stat     fstatb;

    if ((fd = open(file, O_RDONLY, 0)) < 0) {
fprintf (stderr, "ad_read: Couldn't open %s\n", file);
return (-1);
    }
    /*
     * read header
     */
    read(fd, head, sizeof(ad_head_t));
    /*
     * Check the header size
     */
    if (head->ad_hdrsize == 0) {
/*
 * There is no header therefore this file must have been written on a
 * little indian machine (vax).
 */
head->little_indian = TRUE;
    } else {
/*
 * There is a header, check the version number.
 */
if (head->ad_version == 0) {
    /*
     * Version 0 files were only written on VAX's and other little
     * indian machines. Later versions of the header have the
     * little_indian field.
     */
    head->little_indian = TRUE;
}
    }
    /*
```

```
     * check head->little_indian and machine type and set do_byte_swap
     * accordingly.
     */
    if ((head->little_indian && little_indian()) ||
(!head->little_indian && !little_indian()))
do_byte_swap = FALSE;
    else
do_byte_swap = TRUE;


    if (do_byte_swap) {
/*
 * Swap the header bytes
 */
head->ad_hdrsize = SWAPW(head->ad_hdrsize);
head->ad_version = SWAPW(head->ad_version);
head->ad_channels = SWAPW(head->ad_channels);
head->ad_rate = SWAPW(head->ad_rate);
head->ad_samples = SWAPL(head->ad_samples);
head->little_indian = SWAPL(head->little_indian);
    }

    if ((head->ad_hdrsize > 1024) || (head->ad_hdrsize < 0)) {
fprintf(stderr, "ad_read: file = %s, header size = [%d], bad value\n",
file, head->ad_hdrsize);
close(fd);
return (-1);
    }

    if (head->ad_hdrsize == 0) {
head->ad_version = -1;
head->ad_channels = 1;
head->ad_rate = 250;
head->ad_samples = 0;
    }


    /*
     * Header or not skip to the begining of the data.
     */
    lseek(fd, head->ad_hdrsize * 2, 0);

    /*
     * Compute the number samples in this file from an fstat if the header is
     * inadequit.
     */
    if (head->ad_samples == 0) {
if (fstat(fd, &fstatb)) {
    fprintf (stderr, "ad_read: could not get file status on [%s]\n",
     file);
    close (fd);
    return (-1);
}
head->ad_samples = (fstatb.st_size / 2) - head->ad_hdrsize;
    }
    /*
     * Use valloc so that the buffer returned can be used by the analog to
     * digital converter.
     */
    if (!*buf) {
aptr = (short *) malloc(head->ad_samples * 2);
if (aptr == 0) {
    fprintf (stderr, "ad_read: Memory allocation failed.\n");
    close (fd);
    return (-1);
}
    } else
aptr = *buf;
    /*
     * Read the data all in swell foop.
     */
    {
  int samples_read;
/*        printf("\ntrying to read data\n file %x buf %x num_samples %d\
dir %s file %s\n\n",
fd,aptr,head->ad_samples,dir,file);
*/
samples_read = read(fd, aptr, head->ad_samples << 1) >> 1;

if (samples_read < 0) { /* some kind o' read error has occurred */
    perror("ad_read");
    return -1;
}

if (samples_read != head->ad_samples) {
    fprintf (stderr, "ad_read: Premature eof on %s [%d %d]\n",
     file, head->ad_samples, samples_read);
    close(fd);
    return (-1);
}

    }
    close(fd);
    *buf = aptr;
    if (do_byte_swap)
swab (aptr, aptr, head->ad_samples*2);
```

```
    return (0);
}


ad_write (filename, ah, ad_buf)
char *filename;
char *ad_buf;
ad_head_t *ah;
{
    int fd;

    fd = open (filename, O_CREAT|O_TRUNC|O_WRONLY, 0644);
    if (fd < 0) {
fprintf (stderr, "ad_write: Couldn't open %s\n", filename);
return (-1);
    }

    ah->ad_hdrsize = sizeof (ad_head_t) >> 1;
    ah->ad_version = CURRENT_AD_VERSION;
    ah->little_indian = little_indian();

    write (fd, ah, sizeof (ad_head_t));
    write (fd, ad_buf, ah->ad_samples*2);
    close (fd);
    return (0);
}
```

makeFFT.c

```
/*##########################################################################################################################
#
# makeFFT:  Makes an FFT file from an ADC file (demo version, NEW NORMALIZATION).  Performs automatic endpoint detection.
#
# Description:
#
# Usage:
#   makeFFT [-i adcfile] [-o fftfile] [-s framesize] [-v verbosity]
#
# Flags:
#   -i  adcfile   = complete path & filename of the input adc sample file.
#   -o  fftfile   = complete path & filename for the output fft file.
#   -c  clip      = clip endpoints manually?  Default = 0 = automatic clipping.
#   -s  framesize = how many coefficients per frame, default = 16.
#   -v  verbosity = how much debugging information to display.
#
# Input ADC file format:
#   The ADC file contains a small header, followed by a whole bunch of shorts (data points).
#   The header is described by the structure "ad_head" in the include file /../blitz/usr/dbs/src/c/src/mk_wave/ad.h.
#
# Output FFT file format:
#   Each FFT file, such as "data_file.adc", is a binary file in the following format:
# <16 floats>  = 16 floats, 4 bytes each, representing the coefficients of frame #0.
# <16 floats>  = 16 floats, 4 bytes each, representing the coefficients of frame #1.  And so on, for all frames.
#
# Example:
#   "makeFFT -i /../thunder/usr/bojan/demo/data_file.adc -o ./data_file.FFT"
#        will:
#            o  read-in an  "data_file.adc"  from the  "/../thunder/usr/bojan/demo/"  directory and
#            o  create an FFT file  "data_file.FFT"  in the current working directory.
#
# Note:
#   The FFT computation yields frames 5 msec apart, but the final frames produced by this program are 10 msec apart.
#   Hence the word "frame" is potentially confusing.  To distinguish these two meanings, this program consistently
#   adheres to the following convention: a "FRAM" is 5 msec wide, and a "FRAME" is 10 msec wide.
#
# History:
#    24.Mar  90  bojan    Created, based on LPNN's file "makeFFTs.c".
#    23.May  90  jmt      Used standardized "ad_read", rather than nonstandard "areadshort". Eliminated "exchange_bytes" flag
#    13.July 91  tilo     about 21% speedup of the FFT. (it would be 30%, if sincos() would be available on the DEC's)
#     5.Sept 91  mw,tilo  introduced sine lookup table for sine and cosine function in the fft routine.
#    13.Sept 91  tilo     introduced itterative routine for fast Hartley Transformation instead of the fft routine
#                         and a routine to compute the powerspectrum of a Hartley Transformation.
#                         The real, imag arrays were replaced by fht_in, fht_out arrays, both hold real values.
#                         Twice as fast as revision 1.1, both compiled with -O option on a DEC 5000.
#
#-- RCS Info ------------------------------------------------------------------------------------------------------------------
#
# $RCSfile: makeFFT.c,v $     $Revision: 1.4 $        $State: working $
# $Date: 91/09/13 00:57:10 $    $Author: sloboda $      $Locker: sloboda $
#
# $Log:  makeFFT.c,v $
# Revision 1.4  91/09/13  00:57:10  sloboda
# introduced Fast Hartley Transformation, instead of FFT. introduced routines to compute the power spectrum
# for fft and for fht.
# This revision is twice as fast as the original revision 1.1, both compiled with -O option.
#
# Revision 1.3  91/09/05  15:46:26  sloboda
# a real speedup, due to a table lookup in a sine table, rather than function calls of sin(), cos()
# renamed several variables in the fft routine, for sake of readability
#
# Revision 1.2  91/07/14  03:17:03  sloboda
# this version is about 21% faster than the original version (if it's running on a DEC 5000).
# it would be about 30% faster, if there would be a sincos() routine available on the DEC's.
#
# Revision 1.1  91/07/13  16:29:30  sloboda
# Initial revision
#
#
##########################################################################################################################*/


#include <stdio.h>
#include <string.h>
#include <math.h>
#include <limits.h>
#include <sys/time.h>
#include <sys/file.h>
#include <sys/types.h>
#include <sys/stat.h>
#include "ad.h"
#include "adc_rw.c"


#define then
#define mod     %

/*---------------------- Constants ---------------------------------------------------------------------*/

#define M_PI 3.14159265358979323846  /* mathematical constants       tilo's change */
#define M_2PI   6.28318530717958647692
```

```
#define MAX_BND 4 /* begin, end boundary array */
#define SIL_MARGIN 130 /* msec of leading silence */
#define MAX_FILENAMELEN    100 /* max length of a filename */
#define MAX_FRMS           10000   /* maximum number of frames */

#define MAX_FRAMESperWORD  1500 /* max number of 10 msec frames in a word sample */
#define MAX_COEFSperFRAME  16 /* max number of spectral coefficients in a frame */
#define MAX_FRAMESperNORM      10      /* max number of 10 msec frames for normalization */

#define SAMPL_RATE 16    /* sampling rate in kHz */
#define FRAM_SIZ 16    /* size of fram in msec */
#define DSP_WND_SIZ 256    /* number of pts in window */
#define FRAM_RAT 5    /* fram rate in msec */
#define MAX_FRAMS (MAX_FRAMESperWORD * 2)    /* max number of frams in any word */
#define FRAM_OFFSET (SAMPL_RATE * FRAM_RAT)    /*  the offset (in terms of samples) between two succesive frams  */
#define FRAM_WIDTH (SAMPL_RATE * FRAM_SIZ)    /*  width of a single DSP fram that is passed over the time series
                                                    NOTE THAT THE FRAME WIDTH ISN'T NECESSARILY A BASE-2 NUMBER!
                                                    The fram width takes FRAM_SIZ milliseconds of data; depending on
                                                    the sampling rate, this may be more than DSP_WIN_SIZ, causing the
                                                    hamming windowing routine and the FFT routine to barf.
                                                    When FRAM_WIDTH < DSP_WIN_SIZ, the data points are padded out to
                                                    DSP_WIN_SIZ after windowing and before FFTing (as it should be)
                                                */
#define CHOPP_TIME 10.0    /* exact time-axis frame time increment after coefficient merging */
#define SAMPL_P (SAMPL_RATE * FRAM_RAT)    /* endsample adjustment constant for the final phoneme */

#define TRUE 1
#define FALSE 0


/*-------------------------------------------------------------------------------------------------------------------
;  Common macros:
;------------------------------------------------------------------------------------------------------------------*/

#define streq(s1,s2)    (strcmp (s1,s2) == 0)          /* string equality */
#define strneq(s1,s2)   (strcmp (s1,s2) != 0)          /* string inequality */

/*-------------------------------------------------------------------------------------------------------------------
;  Stuff for reading/writing canonical binary files on any type of machine:
;------------------------------------------------------------------------------------------------------------------*/

#define RT_SUN             1            /* RT or SUN machine type */
#define PMAX               2            /* PMAX machine type */
#define VAX                3            /* VAX machine type */
int   machine;                          /* what type of machine are we currently running on? */
union unionT {                          /* union for type conversions */
    float fval;                         /*    4-byte float */
    int   ival;                         /*    4-byte int */
    short sval;                         /*    2-byte short */
    unsigned char cval [4];             /*    4-byte string */
};

/*---------------------- Global Variables ---------------------------------------------------------------------*/

char  adcfile     [MAX_FILENAMELEN]; /* name of input ADC files */
char  fftfile     [MAX_FILENAMELEN];    /* name of the output FFT files */

FILE  *a, /* adc input file */
      *f; /* FFT output file */

float sample [MAX_FRAMESperWORD] [MAX_COEFSperFRAME]; /* spectral coefficients for one sentence sample */
float coeff [MAX_FRAMS] [MAX_COEFSperFRAME]; /* melscale coefficients for one word */
int   bound[MAX_BND]; /* start1, start2, end2, end1 boundary */
float normCoeffA [2*MAX_FRAMESperNORM][MAX_COEFSperFRAME], /* normalizing area of coefficients */
      normCoeff [MAX_COEFSperFRAME];                       /* normalizing coefficient */

float fht_in   [DSP_WND_SIZ]; /* input to the Hartley Transformation (real numbers) */
float fht_out  [DSP_WND_SIZ]; /* output of the Hartley Transformation (real numbers) */
float pow_spct [DSP_WND_SIZ]; /* power spectrum */

/*---------- stuff for the lookup tables : ------------------------------------*/
double sin_tab  [DSP_WND_SIZ],          /* lookup table for the sine function   */
       cos_tab  [DSP_WND_SIZ],
       *s_Ptr,                          /* pointer for the sine in the table    */
       *ms_Ptr,                         /* pointer for minus sin in the table   */
       *c_Ptr;                          /* pointer for the cosine in the table  */
int    BitRev   [DSP_WND_SIZ];          /* table with the bit-reverse numbers
                                           to the base DSP_WND_SIZ              */
/*-----------------------------------------------------------------------------*/

short *ad_buf; /* pointer to a malloc'ed buffer which will hold a file's ADC data */
float ptp [MAX_FRMS]; /* maximum point-to-point values per fram */
ad_head_t head; /* adc file header */

int   numsamples; /* number of data samples in a given ADC file */
int   chop_numfrms; /* number of frams that should fit between startsample and endsample */
int   fram_count, /* number of frams being processed */
      fram_count_norm;
int   np = DSP_WND_SIZ; /* number of data points in the DSP window */
int   np_log_two; /* log of the number of data points in the DSP window */

int clip; /* manual clip? */
float starttime,endtime; /* when does word begin/end (msec)? */
int   startsample,endsample; /* index of first/last ADC data value to use */
float maxval, minval; /* statistics for normalizing coefficients */
```

```
float scale; /* scaling factor for normalization */

int   framesize;                        /* number of spectral coefficients in a frame */
int   maxframes; /* maximum number of frames in any sample */
int   machine; /* machine type */
int   verbose;                          /* verbosity flag */
int   frm; /* debug frame counter */
int   frameN; /* frame counter */

char  *strP; /* string pointer */
int   i,j,k; /* temps */

float startTimEntry, /* manual entries for start and end times */
      endTimEntry;


/*----------------------- Functions ------------------------------------------------------------------------*/

void  init_fft ();                      /* initializes the sine_tab, cosine tables for the fft routine */
void  fft (); /* FFT a time series */
void  fft_pow_spec ();                  /* compute the power spectrum from the fft data */

void  init_fht ();                      /* initialize sin, cos lookup tables for the fht routine */
void  fht ();                           /* FHT a time series (Fast Hartley Transformation) */
void  fht_pow_spec ();                  /* compute the power spectrum from the fht data */

float mkcoeff ();                       /* compute Melscale spectral coefficients from power spectrum coefficients */
float round (); /* rounding subroutine */
int   find_log (); /* determine the most-significant non-zero bit of an integer */
int   areadshort (); /* read an ADC (raw 16-bit a/d data) file */
void  write_float(); /* writes float numbers, machine independent */
int   ham (); /* hamming window a time series */
int   begend (); /* begin-end detection routine */
int   ptp_amp (); /* peak to peak amplitude within a fram */
float normalize (); /* array normalization routine */


/*-------------------------------------------------------------------------
;  Main program:
;-------------------------------------------------------------------------*/

main (argc, argv)
int argc;
char *argv[];
{

/*    printf ("Usage: makeFFT [-i adcfile] [-o fftfile] [-c clip] [-s framesize] [-v verbosity]\n"); */


  init_fht(np);               /* initialize the sine lookup tables for the fht routine */

  machine = get_machine_type ();
  framesize = 16;
  clip = 0;

  k = 1;
  while (argc > k)
    if (argv[k][0] == '-')
      switch (argv[k++][1]) {
        case 'i':  strcpy (adcfile, argv[k++]);       break;
        case 'o':  strcpy (fftfile, argv[k++]);       break;
        case 'c':  clip = atoi (argv[k++]);       break;
case 's':  framesize = atoi (argv[k++]);      break;
case 'v':  verbose = atoi (argv[k++]);        break;
default: {printf ("Unknown flag %s\n", argv[--k]); exit(0);}
    } else       {printf ("Unknown flag %s\n", argv[k]); exit(0);}

    /*-------------------------------------------------------------------------
    ;  Read ADC data file contents into a malloc'd buffer and return a pointer to the buffer
    ;  along with the number of samples in the file (ie. the effective buffer size).
    ;  Then exchange all bytes in the file if necessary.
    ;-------------------------------------------------------------------------*/

    if (ad_read (NULL, adcfile, &ad_buf, &head) < 0)  return; /* read ADC data */
    if (verbose)
printf ("head: hdrsize=%d, version=%d, channels=%d, rate=%d, samples=%d, littleindian=%d\n",
     head.ad_hdrsize, head.ad_version, head.ad_channels, head.ad_rate, head.ad_samples, head.little_indian);
    if (head.ad_rate == 0) /* if sample rate is undefined, */
        head.ad_rate = SAMPS_PER_MS (SAMPL_RATE); /* assume it's 4000/16 = 250. */
    numsamples = head.ad_samples;

    /*-------------------------------------------------------------------------------------------------------
    ;  Compute power spectra and Melscale coefficients for buffered a/d data in a single pattern, fram-by-fram:
    ;-------------------------------------------------------------------------------------------------------*/

    if ((4000/head.ad_rate) != SAMPL_RATE ) {
      printf ("Sample rate inconsistency, is: %dkHz, should be: %dkHz\n", SAMPL_RATE, 4000/head.ad_rate);
      exit (0);
    }

    /*-------------------------------------------------------------------------------------------------------
    ;  Compute the number of frams you can fit (and process) in this entire DSP window (note that DSP_WND_SIZ is
    ;  not the size of the entire window, rather it is the number of data points in the FFT window).
    ;-------------------------------------------------------------------------------------------------------*/
```

```
    fram_count = maxframes = 0; /* initialization */
    np_log_two = find_log (np);                                    /* compute log-2 width of the DSP window */
    chop_numfrms = ptp_amp (ad_buf, numsamples, /* compute point-to-point amplitudes:  IN  */
        ptp); /*                                  OUT */
    normalize (ptp, chop_numfrms, 0.0); /* normalize ptp by its largest element */
    begend_det (ptp, chop_numfrms, /* detect boundaries of ptp:  IN  */
     bound); /*                          OUT */
    startsample = (bound[0] * FRAM_RAT * SAMPL_RATE - SIL_MARGIN * SAMPL_RATE);
    endsample = (bound[3] * FRAM_RAT * SAMPL_RATE + SIL_MARGIN * SAMPL_RATE);

    if (clip) {
      printf ("Starttime (ms): ");
      scanf ("%f", &startTimEntry);
      printf ("Endtime (ms): ");
      scanf ("%f", &endTimEntry);

      startsample = startTimEntry*SAMPL_RATE;
      endsample = endTimEntry*SAMPL_RATE;
    }

    if (startsample < 0) startsample = 0;
    if (endsample > numsamples)  endsample = numsamples;
    printf ("Performing FHT...  ");  fflush (stdout);
    chop_numfrms = (endsample-startsample-DSP_WND_SIZ)/(FRAM_RAT*SAMPL_RATE)+1;

/* startsample = 5440;
   endsample = 38160;
   chop_numfrms = 406; /* st=3840 stp=36240 chop=402 debugging adjustment for testing on Version2 C1.1.adc */

    for (i=startsample; i<=endsample-FRAM_WIDTH; i+=FRAM_OFFSET) {
      if (verbose)
         printf ("DSP (%3d) from %d(%dms) to %d(%dms)\r",
  fram_count, i, i/SAMPL_RATE, i+FRAM_WIDTH-1, (i+FRAM_WIDTH-1)/SAMPL_RATE);
      fflush (stdout);
      if (verbose > 1) {
  k = 1;
  for (j=i; j<i+FRAM_WIDTH; j++) {
      printf ("%d ", ad_buf[j]);
      if ((k++ % 20) == 0) printf ("\n");
  }
  printf ("\n");
      }

      for (j=0; j<np; j++)  fht_in[j] = fht_out[j] = 0.0; /*  clear the global complex computation array */
      ham (fht_in, &ad_buf[i], FRAM_WIDTH); /*  load a fram into the array and Hamming window it */

      if (verbose > 1) {
          for (j=0; j<FRAM_WIDTH; j++)
      printf ("%f ", fht_in[j]);
          printf ("\n");
      }

      fht (np, np_log_two, fht_in, fht_out); /* FHT the array */

      if (verbose > 1) {
          for (j=0; j<np; j++)
              printf ("%f  %f\n", fht_in[j], fht_out[j]);
          printf ("\n");
      }

      /*-------------------------------------------------------------
      ;  Compute the real power spectrum (linear, not logarithmic)
      ;  from the Hartley Transformation :
      ;-------------------------------------------------------------*/

      fht_pow_spec(np, fht_out, pow_spct);

      if (verbose > 1) {                                    /*  display the power spectrum coefficients on stdout  */
        for (j=0; j<np/2; j++)
          printf ("%d.)%3.1f ",j, pow_spct[j]);
        printf ("\n");
      }

      /*-------------------------------------------------------------
      ;  Convert the power spectrum into Melscale coefficients.
      ;  See Waibel (ATR TR-I-0006) sec. 2.1 and
      ;  Waibel & Yegnanarayana (1981) for details.
      ;-------------------------------------------------------------*/

      coeff[fram_count][0] = mkcoeff(pow_spct,0,2);
      coeff[fram_count][1] = mkcoeff (pow_spct,2,6);
      coeff[fram_count][2] = mkcoeff (pow_spct,6,10);
      coeff[fram_count][3] = mkcoeff (pow_spct,10,14);
      coeff[fram_count][4] = mkcoeff (pow_spct,14,18);
      coeff[fram_count][5] = mkcoeff (pow_spct,18,22);
      coeff[fram_count][6] = mkcoeff (pow_spct,22,26);
      coeff[fram_count][7] = mkcoeff (pow_spct,26,30);
      coeff[fram_count][8] = mkcoeff (pow_spct,30,35);
      coeff[fram_count][9] = mkcoeff (pow_spct,35,41);
      coeff[fram_count][10] = mkcoeff (pow_spct,41,48);
      coeff[fram_count][11] = mkcoeff (pow_spct,48,57);
      coeff[fram_count][12] = mkcoeff (pow_spct,57,68);
      coeff[fram_count][13] = mkcoeff (pow_spct,68,81);
      coeff[fram_count][14] = mkcoeff (pow_spct,81,97);
      coeff[fram_count][15] = mkcoeff (pow_spct,97,116);
```

```
      if (verbose > 0) {                              /*  print melscale coefficients to stdout  */
        printf ("(%d)\n", frm++);
        for (j=0; j<MAX_COEFSperFRAME; j++)
          printf ("%f\n", coeff[fram_count][j]);
        printf ("\n");
      }
      fram_count++; /*  fram completed  */
    }

    if (fram_count != chop_numfrms) {
printf ("error: fram_count = %d, but chop_numfrms = %d.  Not the same.\n", fram_count, chop_numfrms);
exit (-1);
    }

    /*-------------------------------------------------------------------------
    ; Calculate intial 10 sample vectors of SIL in order to compute the average
    ; sample vector for later noise compensation.
    ;-------------------------------------------------------------------------*/

    fram_count_norm = 0;
    for (i=0; i<=MAX_FRAMESperNORM*10*SAMPL_RATE-FRAM_OFFSET; i+=FRAM_OFFSET) {
    if (verbose)
        printf ("DSP-N (%3d) from %d(%dms) to %d(%dms)\n", fram_count_norm, i, i/SAMPL_RATE,
                                                  i+FRAM_WIDTH-1, (i+FRAM_WIDTH)/SAMPL_RATE);

      for (j=0; j<np; j++)  fht_in[j] = fht_out[j] = 0.0;       /*  clear the global complex computation array       */
      ham (fht_in, &ad_buf[i], FRAM_WIDTH);                     /*  load a fram into the array and Hamming window it */

      fht (np, np_log_two, fht_in, fht_out);                    /* FHT the array */

      fht_pow_spec(np, fht_out, pow_spct);

      /*-----------------------------------------------------
      ;  Convert the power spectrum into Melscale coefficients.
      ;  See Waibel (ATR TR-I-0006) sec. 2.1 and
      ;  Waibel & Yegnanarayana (1981) for details.
      ;-----------------------------------------------------*/

      normCoeffA[fram_count_norm][0] = mkcoeff(pow_spct,0,2);
      normCoeffA[fram_count_norm][1] = mkcoeff (pow_spct,2,6);
      normCoeffA[fram_count_norm][2] = mkcoeff (pow_spct,6,10);
      normCoeffA[fram_count_norm][3] = mkcoeff (pow_spct,10,14);
      normCoeffA[fram_count_norm][4] = mkcoeff (pow_spct,14,18);
      normCoeffA[fram_count_norm][5] = mkcoeff (pow_spct,18,22);
      normCoeffA[fram_count_norm][6] = mkcoeff (pow_spct,22,26);
      normCoeffA[fram_count_norm][7] = mkcoeff (pow_spct,26,30);
      normCoeffA[fram_count_norm][8] = mkcoeff (pow_spct,30,35);
      normCoeffA[fram_count_norm][9] = mkcoeff (pow_spct,35,41);
      normCoeffA[fram_count_norm][10] = mkcoeff (pow_spct,41,48);
      normCoeffA[fram_count_norm][11] = mkcoeff (pow_spct,48,57);
      normCoeffA[fram_count_norm][12] = mkcoeff (pow_spct,57,68);
      normCoeffA[fram_count_norm][13] = mkcoeff (pow_spct,68,81);
      normCoeffA[fram_count_norm][14] = mkcoeff (pow_spct,81,97);
      normCoeffA[fram_count_norm][15] = mkcoeff (pow_spct,97,116);

      fram_count_norm++;                                        /*  normalization fram completed  */
    }

    /*-------------------------------------------------------------------
    ; STEP 1: Calculate an average sample vector within initial 10 frames
    ;         and subtract it from speech sample vectors.
    ;-------------------------------------------------------------------*/

    for (j=0; j<framesize; j++)
      normCoeff[j] = 0;                                         /* clear normalization coefficient */

    for (j=0; j<framesize; j++)
      for (i=0; i<fram_count_norm; i++)
        normCoeff[j] += normCoeffA[i][j];                       /* calculate average  coefficient */

    for (j=0; j<framesize; j++)
      normCoeff[j] /= fram_count_norm;                          /* average the normalizing coefficient */

    frameN = fram_count / 2;                                    /* two frams will be collapsed into one frame */
    if (frameN > maxframes)  maxframes = frameN;                /* remember max number of frames in any sample */

    for (i=0; i<frameN; i++)
        for (j=0; j<framesize; j++)
            sample[i][j] = (coeff[2*i][j] + coeff[2*i+1][j])/2;
/*            sample[i][j] = (coeff[2*i][j] + coeff[2*i+1][j])/2 - normCoeff[j];  /* subtract for silence */

    /*-------------------------------------------------------------------------
    ; STEP 2: Find  minval  and  maxval  for normalization.
    ;-------------------------------------------------------------------------*/

    maxval = -HUGE;
    minval = HUGE;
    for (i=0; i<frameN; i++)
        for (j=0; j<MAX_COEFSperFRAME; j++) {
            if (sample[i][j] < minval)  minval = sample[i][j];
            if (sample[i][j] > maxval)  maxval = sample[i][j];
        }

    scale = maxval - minval;                                    /* scaling factor */
```

```
    if (verbose) {
      printf ("minval = %f, maxval = %f, scale = %f\nSilence = ", minval, maxval, scale);
      for (j=0; j<framesize; j++)
        printf ("%f ", normCoeff[j]);
      printf ("\n");
    }

    /*-------------------------------------------------------------------------
    ; STEP 3: Scale the sample vectors.
    ;-------------------------------------------------------------------------*/

    frm = 1;
    for (i=0; i<frameN; i++) {
        if (verbose) printf ("(%d)\n", frm++);
        for (j=0; j<framesize; j++) {
            sample[i][j] = (sample[i][j] - minval) / scale;     /* scale */
            if (verbose) {
              printf ("%6f ", sample[i][j]);
              if (j%8==7) printf ("\n");
            }
        }
    }

    /*-------------------------------------------------------------------------
    ;  Write sample into a binary FFT file:
    ;-------------------------------------------------------------------------*/

    if ((f = fopen (fftfile, "w")) == NULL) {
      printf ("Cannot open FFT file %s.\n", fftfile); exit(0);
    }
    printf ("Writing %s\n", fftfile);
    for (i=0; i<frameN; i++)  /* for each frame: */
        for (j=0; j<framesize; j++) /*     for each coefficient: */
    write_float (f, sample[i][j]); /*         write it into the FFT file */

    fclose (f); /* close the file */
    free (ad_buf);                                  /* free the malloc'ed buffer */

} /* main */


/*=============================================================================================================================
; get_machine_type:  Figures out what type of machine we're running on.
;
; Parameters: none.
;
; Returns:
;   One of: RT_SUN, PMAX, or VAX (values 1, 2, 3).
;
; History:
;   10/20/89  jmt  Created.
;============================================================================================================================*/

int get_machine_type ()
{
union unionT u;

if (sizeof(float) != 4)
    {printf ("Problem: sizeof float = %d\n", sizeof(float)); exit(0);}
if (sizeof(int) != 4)
    {printf ("Problem: sizeof int = %d\n", sizeof(int)); exit(0);}
if (sizeof(short) != 2)
    {printf ("Problem: sizeof short = %d\n", sizeof(short)); exit(0);}

u.fval = 123.456;                               /* assign a known float.  How is it represented? */
if (u.cval[3] == 121)  return (RT_SUN);         /* if 121 is in byte #3, we're on a RT or SUN */
if (u.cval[0] == 121)  return (PMAX);           /* if 121 is in byte #0, we're on a PMAX */
if (u.cval[2] == 121)  return (VAX);            /* if 121 is in byte #2, we're on a VAX */
}


/*=============================================================================================================================
; write_float:  Writes a float into a binary file.  Works on any machine.
;
; Parameters:
;   fp = binary file pointer.  (Assumed to be already open.)
;
; History:
;   10/20/89  jmt  Created.
;============================================================================================================================*/

void write_float (fp, f)
FILE *fp;
float f;
{
union unionT u,v;

u.fval = f;
switch (machine) {
    case RT_SUN: v.fval = u.fval; break;                                /* RT or SUN:  already in correct format */
    case PMAX:   v.cval[0] = u.cval[3];                                 /* PMAX:  reverse the bytes */
                 v.cval[1] = u.cval[2];
                 v.cval[2] = u.cval[1];
                 v.cval[3] = u.cval[0];  break;
```

```
    case VAX:    v.cval[0] = (u.cval[1]==0 ? 0 : u.cval[1]-1);          /* VAX:  shuffle the bytes */
                 v.cval[1] = u.cval[0];
                 v.cval[2] = u.cval[3];
                 v.cval[3] = u.cval[2];  break;
    default: {printf ("Unknown machine type %d", machine); exit(0);}
}
fwrite (&v.fval, sizeof(float), 1, fp);
}


/*=========================================================================
; mkcoeff:  Compute melscale coefficients from power spectrum coefficients.
;     (See A. H. Waibel (ATR-TR-I-0006) for conceptual details.)
;
; Parameters:  ??
;
; Returns:  ??
;
; History:
;   1986  ahw  Created.
;=========================================================================*/

float mkcoeff(samples,startsample,endsample)

float samples[];
int startsample,endsample;
{
  int i;
  float coeff;

  if (startsample == 0) coeff = samples[0];
  else coeff = samples[startsample]/2.0;
  coeff += samples[endsample]/2.0;

  for (i=startsample+1;i<=endsample-1;i++) coeff += samples[i];
  coeff = log10 ((double) coeff);
  return (coeff);
}


/*=========================================================================
; find_log:  Determines the most significant non-zero bit of an integer.
;      (This is equivalent to computing the base-2 order of magnitude of the integer, not equivalent to its
;      base-2 logarithm.  However, in the context it is used (always operating on a radix-2 integer), the
;      function is equivalent to log-2 (np).
;
; Parameters:
;   np = integer??
;
; Returns:
;   Most significant non-zero bit of an integer.
;
; History:
;   1986  ahw  Created.
;=========================================================================*/

int find_log (np)
int np;
{
  int np_log;
  unsigned itemp;

  itemp = np;
  np_log = 0;
  while (itemp > 1) { itemp >>= 1; np_log++;}
  return (np_log);
}


int chckiw=0;
/*=========================================================================
; ham:  Hamming windowing routine.
;
; Parameters:
;   r      = windowed data by hamming window
;   nad    = AD data (16 bit integer)
;   iw     = sample number in hamming window (8 <= iw <= 1024)
;
; Globals assumed:
;   chckiw may be set previously (default = 0).
;
; Returns:
;   0 if everything is okay, 1 if there is an error.
;
; History:
;   "June 22, 19??"  Shikano   Wrote the original version.
;=========================================================================*/

ham(r,nad,iw)
short   nad[];
int iw;
float r[];
{int i;
static float d,wind[1024];
double cos();
 if(chckiw != iw)
```

```
{ chckiw = iw;
  d = M_PI / (iw-1);
  for ( i=0 ; i<iw ; i++ )
wind[i] = (0.54+0.46*cos((double)(d*(2*i-iw+1))));
}
 if(iw < 8 || iw > 1024 )
{ printf(" ham err");
  return(1);
}

 for ( i=0 ; i<iw ;i++ )
r[i] = wind[i]*nad[i];
 return(0);
}


/*============================================================================
; fht:  FHT routine (Fast Hartley Transformation)
;
; Parameters :
;   in :
;         N       number of data points in the DSP window
;   ldN     ld(N) log to base 2
;         input   array with real values, that has to be transformed
;
;   out :
;   output  array with the transformation.
;
; Note :  init_fht() has to be called at the beginning of the program.
;
; History :
;   13.Sept 91  tilo  created. lookup-table for sin(x), cos(x) is used.
;
;============================================================================*/

void fht (N, ldN, input, output)              /* itterative , fast hartley transformation ;
                                                 in the examples I assume N=256, ldN=8 */
        int      N, ldN;
        float  *input, *output;
{
        double   *sine, *cosine;                                /* pointers into the lookup tables      */
        float    temp1[DSP_WND_SIZ],                            /* two arrays for intermediate results  */
                 temp2[DSP_WND_SIZ];
        float    *q, *p2, *p3;                                  /* pointers into the arrays             */
        float    *src, *tar, *t, *src_beg, *tar_beg;            /* src source , tar target , t temp     */
        float    t0, t1, t2, t3, f0, f1, f2, f3;                /* all for intermediate results         */
        int      i, j, k, Nr, offset;
        int      Ndiv4, Ndivn, ldn, n,                          /* n is length of target array          */
                 ndiv2, ndiv2m1, ldNm1;                         /* ndiv2 is length of source array      */
                                                                /* m1 means minus 1                     */

  ldNm1 = ldN - 1;
  Ndiv4 = (N>>2);

  /*--- compute all Ndiv4 quadruples, and do the bit reversal, O(N) ---*/

  tar = temp1;
  offset = 0;
  for (i=0; i<Ndiv4; i++)                                       /* "offset" in the comments is :  offset = i*Ndiv4   */
    {
      f0 = input[ BitRev[offset++] ];                           /* in[offset],in[offset+1],in[offset+2],in[offset+3] */
      f1 = input[ BitRev[offset++] ];
      f2 = input[ BitRev[offset++] ];
      f3 = input[ BitRev[offset++] ];

      t0 = f0 + f1; t1 = f2 + f3; t2 = f0 - f1; t3 = f2 - f3;

      *tar++ = t0 + t1;     /* out[0] = in[0]+in[1]+in[2]+in[3] */
      *tar++ = t2 + t3;     /* out[1] = in[0]-in[1]+in[2]-in[3] */
      *tar++ = t0 - t1;     /* out[2] = in[0]+in[1]-in[2]-in[3] */
      *tar++ = t2 - t3;     /* out[3] = in[0]-in[1]-in[2]+in[3] */
    }

  /*--- merging stages :  O((ldN-2) * N) ---*/

  src = src_beg = temp1;
  tar = tar_beg = temp2;

  n = 8; ndiv2 = 4;                         /* ndiv2 is the length of the source array; n is the length of the target array */
  Ndivn = (Ndiv4 >> 1);

  for (ldn=3; ldn<=ldN; ldn++)              /* count the stages that have to be computed */
    {                                       /* starts with ldn=3 ; Ndivn = (N>>ldn) = 32 ; ndiv2 = 4 ; n = 8 */
      ndiv2m1 = ndiv2 - 1;
      q = src + ndiv2;

      for (Nr=0; Nr<Ndivn; Nr++)              /* count the number of sub-arrays that where put together in this stage */
        {
          /*--- merge two n-tuples to 2n-tuples ,  O(N) ---*/    /*  offset is (Nr * n)    */

          cosine = c_Ptr;
          sine   = s_Ptr;

          k = 0;
          for(j=0; j<2; j++)                            /* two times */
```

```
            {
              p2 = src;
              p3 = p2 + ndiv2;

              for(i=0; i<ndiv2; i++)                          /* ndiv2 times */
                {                                                 /*  idx1 = offset + i;                                  */
                                                                  /*  idx2 = offset + (i mod Ndiv2);                      */
                                                                  /*  idx3 = idx2 + Ndiv2;                                */
               /* p4 = q  + (k & ndiv2m1); */                     /*  idx4 = offset + Ndiv2 + (Ndiv2m1 * i) mod Ndiv2; */

                  *tar++ = (*p2++) + (*cosine) * (*p3++) + (*sine) * q[k & ndiv2m1];

                  k  += ndiv2m1;
                  cosine += Ndivn;
                  sine   += Ndivn;
                }
            }
          q += n;  src += n;                              /* increase src and q by  offset */
        }

      n <<= 1;  ndiv2 <<= 1; Ndivn >>= 1;       /* length of source array, length of target array  gets doubled */

      if (ldn<ldNm1)
         then { tar = src_beg; src = tar_beg;   /* toggle source, target arrays */
                tar_beg = tar; src_beg = src;
              }
         else { tar = output; src = tar_beg;    /* if ldn = ldNm1 : target has to be the output array */
              }
    }
}

/*=========================================================================
; init_fht:   initialization routine for the fht routine
;             to be called just once, when the program starts
; Parameters :
;   in:     N    number of data points in the DSP window
;
; Globals initialized :
;       s_Ptr, c_Ptr , sin_tab, cos_tab, BitRev are initialized.
;
; Comment :
;           initializes the sin, cos lookup tables,
;           and the BitRev lookup table,
;           which are used by the fht routine.
;
; History :
;   12.Sept 91  tilo   created
;
;=========================================================================*/

void init_fht ( N )

   int     N;      /* length of the table */
{
   int     i;      /* temporal table index */
   int     a,b,p;

   double * ptr,   /* temporal table pointer */
          * ptr1,
           scl,   /* scaling factor */
           x;


 scl = M_2PI / N;

 x = 0.0;
 ptr  = sin_tab;                        /* point to the beginning of the sin table */
 ptr1 = cos_tab;

 for (i=0; i<N; i++, x += scl)
  {
   *ptr++  = sin(x);                     /* buildup the sin, cos tables */
   *ptr1++ = cos(x);
  }

 s_Ptr  = sin_tab;                      /* initialize the pointers to the beginning of the tables */
 c_Ptr  = cos_tab;

 for (i=0; i<N; i++)                    /* for all Numbers 0...N-1 */
   { a=i; b=0;
     for (p=(N>>1); p>0; p>>=1)         /* buildup the table with the bit-reversed numbers to the base N */
       {
         b += p*(a mod 2);
         a >>= 1;
       }
     BitRev[i] = b;
   }
}

/*=========================================================================
; fht_pow_spec:   compute the power spectrum of a FHT
;
; Parameters :
;   in:     N       number of data points in the DSP window (has to be N !)
;           in      array, that holds the fht data
;
```

```
;   out:    pow     array with the power spectrum
;
; Comment:  P(f) = 1/2 * ( H(f)^2 + H(-f)^2 )
;
; History :
;    13.Sept 91  tilo   created
;
;============================================================================*/

void fht_pow_spec(N, in, pow)      /*   P(f) = 1/2 * ( H(f)^2 + H(-f)^2 )   */

     int   N;
     float *pow, *in;
{
     float  x, y;
     float *p, *q;
     int    n;


  p = in;
  q = in + N-1;
  y = *p;                  /* y = in[0] */
  for (n=0; n<N; n++)
    {
      x = *p++;
      *pow++ = 0.5 * (x*x + y*y);
      y = *q--;
    }
}


/*============================================================================
; fft:  FFT routine.
;
; Parameters :
;   in :
;         N         number of data points in the DSP window
;   ldN       ld(N) log to base 2
;   mode      -1 for forward fft , 1 for inverse fft
;
;   in / out :
;         real      array of real parts
;   imag      array of imaginary parts
;
; Note :  init_fft() has to be called at the beginning of the program.
;
; History :
;   ??.???? ??  ???       created, without documentation, but with goto's.
;   13.July 91  tilo      used registers and += /= >>= ... for 21% speedup.
;         changed the shape of the mainloop (goto's considered harmful...)
;    3.Sept 91  mw,tilo  introduced lookup-table for sin(x), cos(x)
;
;============================================================================*/

void fft ( real , imag , N , ldN , mode )

  register  float *real , *imag;
            int         N;
            int         ldN , mode;        /*  mode, -1 for forward,  1 for inverse  */
{
   int     lmx , Ndiv2 , Nminus1 ;
   int      step;

   double   scl , arg,
            *sp, *sp_start, *cp;     /* pointers to sine and cosine values */

   real -= 1;
   imag -= 1;
   lmx = N;
   scl = M_2PI / N;

   cp = c_Ptr;
   if (mode == 1)
      then sp_start = s_Ptr;
      else if (mode == -1)
              then sp_start = ms_Ptr;
              else { fprintf(stderr, "\nERROR , wrong argument for mode in fft() !\n"); exit(0); };
   sp = sp_start;
   step = 1;

   { register int    i, j;             /* array indices for merging data */
     register double t1, t2, sine, cosine;
     register int    lo, lm, li, lix;

       for ( lo = 1 ; lo <= ldN ; lo++ )                /* outer loop */
{
lix = lmx;
lmx >>= 1;                                  /* lmx = lmx / 2; */
sp = sp_start; cp = c_Ptr;                  /* reset the pointers to the beginning of the "tables" */

for ( lm = 1 ; lm <= lmx ; lm++ )           /* middle loop */
{
cosine = *cp; sine = *sp;                /* get the sine, cosine values from the table */

                        sp += step; cp += step;                /* increment the pointers to the "tables" */
```

```
for ( li = lix ; li <= N ; li += lix ) /* inner loop */
{
i = li - lix + lm;
j = i + lmx;
t1 = real[i] - real[j];
t2 = imag[i] - imag[j];

real[i] += real[j];
imag[i] += imag[j];

real[j] = (cosine * t1) + (sine * t2);
imag[j] = (cosine * t2) - (sine * t1);
}
}
step *= 2;                                  /* double the step distance */
}
   }

   { register int    i, j, k;
     register double t1, t2;

        j = 1;
        Ndiv2   = N / 2;
        Nminus1 = N - 1;
for ( i = 1 ; i <= Nminus1 ; i++ )
{
            if ( i < j )
    {
  t1 = real[j];            /* swap real[j] and real[i] */
  t2 = imag[j];            /* swap imag[j] and imag[i] */

  real[j] = real[i];
  imag[j] = imag[i];
  real[i] = t1;
  imag[i] = t2;
            }
            k = Ndiv2;
            while (k < j)
            {
  j -= k;                  /* j = j - k */
  k >>= 1;                 /* k = k / 2 */
            }
            j += k;                       /* j = j + k */
}

if ( mode == 1 )
{
for (i = 1 ; i <= N ; i++ )
{
            real[i] /= N;
   imag[i] /= N;
}
}
   }

 return;
}

/*=========================================================================
; init_fft:   initialization routine for the fft routine
;             to be called just once, when the program starts
; Parameters :
;   in:     N    number of data points in the DSP window
;
; Globals initialized :
;       s_Ptr, ms_Ptr, c_Ptr, sin_tab are initialized.
;
; Comment :
;           initializes the sine lookup table,
;           which is used by the fft routine
;           because of the symetry of sine, cosine we're using
;           just one sine-table
;           cos(x) = sin(x+pi/2)  ;  sin(-x) = sin(x+pi)
;
; History :
;    5.Sept 91  tilo   created
;
;=========================================================================*/

void init_fft ( N )

   int     N;      /* length of the table */
{

   int     i;      /* temporal table index */
   double * p,     /* temporal table pointer */
          scl,     /* scaling factor */
          x;


 scl = M_2PI / N;

 x = 0.0;
 p = sin_tab;                       /* point to the beginning of the sine table */

 for (i=0; i<N; i++, x += scl)
```

```
   *p++ = sin(x);

 s_Ptr  = &(sin_tab[0]   );        /* initialize the pointers to the beginning ...          */
 ms_Ptr = &(sin_tab[N/2]);         /* ... of the three functions sin(x), sin(-x), cos(x)    */
 c_Ptr  = &(sin_tab[N/4]);
}

/*============================================================================
; fft_pow_spec:   compute the power spectrum of a FFT
;
; Parameters :
;   in:     N       number of data points in the DSP window (N/2 is sufficient)
;           re,im   two arrays, that hold the fft data
;
;   out:    pow     array with the power spectrum
;
; Comment:  P(f) =  F_re(f)^2 + F_im(f)^2
;
;           it's sufficient to call this routine with N/2 instead of N
;
; History :
;    12.Sept 91  tilo   created
;
;============================================================================*/

void fft_pow_spec(N, re, im, pow)   /*   P(f) =  F_re(f)^2 + F_im(f)^2    */

     int   N;
     float *pow, *re, *im;
{
     register
     float r,i;
     int   n;

  for (n=0; n<N; n++)
    {
      r = *re++;
      i = *im++;
      *pow++ = r*r + i*i;
    }
}


/*============================================================================================================================
; begend_det:  Beginning/endpoint detector.
;
; Parameters:
;   IN:
; ptp     = array of peak-to-peak amplitude values.
; numfrms = number of frames.
;   OUT:
; bound   = boundaries.  [0] and [3] are coarse boundaries, [1] and [2] are fine boundaries.
;
; Algorithm:
;   This routine first finds a big jump (THRESH2) and then tries to find a smaller jump (THRESH1)
;   to determine the precise, fine location of the beginning/end of the ADC data.
;
; History:
;   1990  jmt+bojan  Obtained code from Alex Waibel.
;============================================================================================================================*/

begend_det (ptp, numfrms, bound)
float *ptp;
int numfrms, *bound;
{

#define THRESH1 0.05
#define THRESH2  0.2 /* 0.3 sometimes failed, 0.2 is more sensitive. */
#define TIM_BEG_INT 20 /* begin time interval  in msec */
#define TIM_END_INT 100 /* end time interval in msec */

int i,j,frm_beg_int,frm_end_int;
int start1,start2,end1,end2;
int bnd_num;
float *ptr1,*ptr2;

frm_beg_int = TIM_BEG_INT/FRAM_RAT;
frm_end_int = TIM_END_INT/FRAM_RAT;
for (i=0;i<10;i++) bound[i] = 0;
bnd_num = 0;
start1 = start2 = end1 = end2 = 0;

/* find begin point */
/*------------------*/
ptr1 = ptp;
ptr2 = &ptp[frm_beg_int];
for (i=frm_beg_int;i<numfrms;i++) {
if (*ptr2 > ((*ptr1)+THRESH2)) {start2 = i-1; break;}
ptr1++; ptr2++;
}
ptr1 = ptp;
ptr2 = &ptp[frm_beg_int];
start1= start2;
for (i=frm_beg_int;i<start2;i++) {
if (*ptr2 > ((*ptr1)+THRESH1)) {start1 = i-1; break;}
ptr1++; ptr2++;
```

```
}

/* find end point */
/*------------------*/
ptr1 = &ptp[numfrms-1];
ptr2 = &ptp[numfrms-1-frm_end_int];
for (i=numfrms-1;i>start2;i--) {
if (*ptr2 > ((*ptr1)+THRESH2)) {end2 = i-frm_end_int; break;}
ptr1--; ptr2--;
}
ptr1 = &ptp[numfrms-1];
ptr2 = &ptp[numfrms-1-frm_end_int];
end1 = end2;
for (i=numfrms-1;i>end2;i--) {
if (*ptr2 > ((*ptr1)+THRESH1)) {end1 = i-frm_end_int; break;}
ptr1--; ptr2--;
}

bound[0] = start1;
bound[1] = start2;
bound[2] = end2;
bound[3] = end1;

if (start1 == 0) printf ("No type 1 startpoint found\n");
if (start2 == 0) printf ("No type 2 startpoint found\n");
if (end2 == 0) printf ("No type 2 endpoint found\n");
if (end1 == 0) printf ("No type 1 endpoint found\n");
bnd_num = 4;
return (bnd_num);
}


/*=============================================================================================================
; ptp_amp:  Computes an array of point-to-point amplitudes.
;
; Parameters:
;   IN:
; ad_buf     = array of ADC samples.
; numsamples = size of ad_buf.
;   OUT:
; ptp        = array of point-to-point amplitudes.
;
; Returns:
;   Length of ptp array.
;
; History:
;   1990  jmt+bojan  Obtained code from Alex Waibel.
;=============================================================================================================*/

ptp_amp (ad_buf, numsamples, ptp)
short *ad_buf;
int numsamples;
float ptp[];
{
int i,j, numfrms;
int sampl_shift,window_length;
int start_frame;
short ptpmax,ptpmin;
float *ptr1,ftemp;
short *ptr2,*startptr;

sampl_shift = FRAM_RAT * SAMPL_RATE;
window_length = SAMPL_RATE * FRAM_SIZ;
ftemp = FRAM_SIZ/FRAM_RAT;
start_frame = (int) (ftemp/2.0);

numfrms = (numsamples - window_length + sampl_shift) / sampl_shift;
if (numfrms > MAX_FRMS)
 {printf ("Failure in ptp_ampl: Frame Number exceeds array bounds\n"); return (-1);}

ptr1 = ptp;
for (i=0;i<start_frame;i++) *ptr1++ = 0.0;
ptr2 = ad_buf;
for (i=start_frame;i<numfrms;i++) {
ptpmax = -HUGE;
ptpmin = HUGE;
startptr = ptr2;
for (j=0;j<window_length;j++) {
if (*ptr2 > ptpmax) ptpmax = *ptr2;
if (*ptr2 < ptpmin) ptpmin = *ptr2;
ptr2++;
}
*ptr1++ = (float) (ptpmax - ptpmin);
ptr2 = startptr + sampl_shift;
}
return (numfrms);
}


/*=============================================================================================================
; normalize:  Normalizes an array of values.
;
; Parameters:
;   farray  = array of floats.
;   numfrms = size of farray.
;   factor  = normalization factor.  If factor = 0, farray is normalized by its largest absolute value.
```

```
;
; Returns:
;    Maximum absolute value in farray before normalization.  (Or factor, if factor != 0.)
;
; History:
;    1990  jmt+bojan  Obtained code from Alex Waibel.
;=============================================================================================================*/

float normalize (farray, numfrms, factor)
float *farray,factor;
int numfrms;
{
int i,j;
float *fptr,max;

if (factor == 0.0) {
    fptr = farray;
    max = -HUGE;
    for (i=0;i<numfrms;i++) {
if (*fptr > max) max = *fptr;
if (-(*fptr) > max) max = -(*fptr);
fptr++;
    }
}
else max = factor;
fptr = farray;
for (i=0;i<numfrms;i++)
*fptr++ /= max;
return (max);
}
```

D.3 Analyse der Sprachsignale mittels LPNNs

std_fe.c

```
/*-----------------------------------------------------------------------------
  Standard Front-End Hauptprogramm fuer MasPar-Programme
  Projekt :
  Datei    : Makefile
  Autor    : Tilo Sloboda, Karlsruhe
  Stand    : 11.05.91
  Umgebung: UNIX, DECstation, (MasPar)
  RCS      : $Id: std.fe.c,v 1.1 91/05/11 21:35:15 sloboda Exp $
-----------------------------------------------------------------------------*/
extern mpl_main();

main()
{
  callRequest( mpl_main, 0);
}
```

lpnn_be.h

```
/*#########################################################################################################################################
 #
 # lpnn_be.h            parallel implementation of Linked Predictive Neural Networks on the MasPar
 #                      created 1991 by Tilo Sloboda  (sloboda@ira.uka.de)
 #
 # Description :
 #   In this implementation each PE calculates euclidian distance between the next frame and it's prediction.
 #   The results are stored into a file "PredDistFile" and can be read from a Machine running the DP.
 #
 #-- RCS Info ---------------------------------------------------------------------------------------------------------------------------
 #
 # $RCSfile: lpnn_be.h,v $     $Revision: 1.4 $        $State: Exp $
 # $Date: 1991/11/29 11:20:43 $    $Author: sloboda $      $Locker:  $
 #
 # $Log: lpnn_be.h,v $
 # Revision 1.4  1991/11/29  11:20:43  sloboda
 # *** empty log message ***
 #
 # Revision 1.2  91/07/25  12:58:47  sloboda
 #  running version, which predicts nonsense.
 #
 # Revision 1.1  91/07/15  20:46:26  sloboda
 # Initial revision
 #
 #########################################################################################################################################*/

#ifndef _LPNN_BE_H_
#define _LPNN_BE_H_

/*== Includes ============================================================================================================================*/

#include "ts_std.h"

/*== Public Constants ====================================================================================================================*/

/*== Public Types ========================================================================================================================*/

/*== Public Variables ====================================================================================================================*/


/*#########################################################################################################################################
 #
 #      PUBLIC ROUTINES
 #
 #########################################################################################################################################*/

extern        void   p_predict();

extern        void   p_write_scores();

extern        void   init_MasPar();

/* --- just for testing them : -------------------*/

extern plural void   p_Add_Vectors();

extern plural void   p_Multiply_Vector_by_full_Matrix();

extern plural void   p_Multiply_Vector_by_sparse_Matrix();

extern plural void   p_sigmoid();

extern plural float  p_distance();

extern        bool   ValidConnections();

extern plural float  p_get_MatrixElement();

extern plural void   p_put_MatrixElement();


#endif
```

lpnn_be.m

```
/*##########################################################################################################################################
#
# lpnn_be.m            parallel implementation of Linked Predictive Neural Networks on the MasPar
#
##########################################################################################################################################
#
#   Copyright (C) 1991 by     Tilo Sloboda,   (sloboda@ira.uka.de)
#
#   All rights reserved.      NO CITING BEFORE JAN. 1992
#
#
#   This software was developed at the
#
#             University of Karlsruhe
#
#             Dept. of Informatics
#             Inst. f. Program Structures
#             and Data Organisation
#
#             P.O. Box. 6980
#             7500 Karlsruhe 1
#             WEST GERMANY
#
#
#   This software is part of a parallel JANUS implementation on a
#   MasPar machine, based on Joe Tebelskis LPNN JANUS system.
#
#   It may be used for demo purposes by members of the JANUS project.
#   It's for internal use only.
#
#   It may be copied  only  to members of the JANUS project
#   in accordance with the explicit permission to do so
#   and  with the  inclusion  of  the  copyright  notices.
#
#   This software  or  any  other copies thereof may
#   not be provided or otherwise made available to any other person.
#
#
#   Results accomplished  by this software may not be cited,
#   provided or otherwise made available to any other person
#   before I published my masters thesis (January 1st, 1992).
#   If cited, my Name and the University of Karlsruhe have
#   to be mentioned.
#
#
#   No title to and ownership of the software is hereby transferred.
#
#   DO NOT MAKE CHANGES TO THE SOFTWARE WHITHOUT EXPLICIT PERMISSION.
#
##########################################################################################################################################
#
# Description :
#   In this implementation each PE calculates euclidian distance between the next frame and it's prediction.
#   The results are stored into a file "data.scores" and can be read from a Machine running the DP.
#
# Note :
#   see the file  lpnn_be.h            for a description of the public constants, types, variables.
#   see the file  Data_Structures      for a description of the internal datastructures.
#   see the file  DemoSynchronisation  for a description of the file transfer protocol in the demo.
#
# Implementation Note :
#
#   From now on, we the Weightmatrices are stored transposed (see "MatVec_be.m") ,
#   which means that for A[i][j] index i varies faster than index j.
#   (different from C's standard way to store matrices )
#
# Last change :
#   27.July 91  15:25  Tilo    debugging it.   the order of the weights was wrong ! re-wrote the read_weights routine.
#   28.July 91  17.35  Tilo    changed all filenames to have no path
#    3.Sept 91  21:48  Tilo    changed the file formats to the bawl formats :
#                              weights file version 0; model file version 2; network file version 1
#   24.Sept 91  01:02  Tilo    changed timing routines, to use revision 1.4 of timing.[ch]
#   26.Sept 91  22:46  Tilo    moved the low level file io stuff to "fileIO_be.m" - which is now for DEC and VAX front end.
#   27.Sept 91  18:22  Tilo    minor bug fixed; it caused the program to crash arbitrary during initialization.
#                              introduced the -DVAX option in read_FFT too; included the copyright notice.
#   11.Oct  91  18:15  Tilo    changed the read_Weights routine - now it's much faster !
#   24.Oct  91   4:21  Tilo    moved all vector and matrix operations to "MatVec_be.m" and changed some of the calls to the
#                              matrix and vector operations.
#   24.Oct  91  18:15  Tilo    simplified the read_FFT routine a little.
#   21.Nov  91  16:28  Tilo    tried to implement the GiveUp() routine.
#   25.Nov  91  20:40  Tilo    problem solved - from now on, we can run demos with predictions for two languages !
#                              use cc comand line option -DBRUTE  to build a program that takes all CPU time.
#    2.Dec  91  13:05  Tilo    removed a bug in the routine allocate_PE_mem()
#
#-- RCS Info ------------------------------------------------------------------------------------------------------------------------------
#
# $RCSfile: lpnn_be.m,v $     $Revision: 1.8 $        $State: Exp $
# $Date: 1991/12/02 12:35:27 $    $Author: sloboda $      $Locker:  $
#
# $Log: lpnn_be.m,v $
# Revision 1.8  1991/12/02  12:35:27  sloboda
# removed a bug in allocate_PE_mem().
#
```

```
# Revision 1.7  1991/11/29  11:21:23  sloboda
# The matrix and vector operations were moved to the file MatVec_be.m (new!)
# The read_weights routine is now much faster.
# read_FFT was simplified, due to changes in fileIO_be.m
# The appropriate fileIO_be.m now contains FloatSwapping routines.
# most variables are now static.
# A GiveUp proutine was introduced in the main control loop - so two identical
# incarnations of this program can be run for two different languages, and they
# share the CPU time .
#
# Revision 1.6  1991/10/10  12:45:55  sloboda
# timing routines changed ; the low level file io stuff was moved  to "fileIO_be.m"
#  - which is now for DEC and VAX front end ; introduced the -DVAX option in read_FFT too.
#
# Revision 1.5  91/09/03  22:26:44  sloboda
# BAWL compatible version - all input files have BAWL format
#
# Revision 1.4  91/09/03  19:38:58  sloboda
# First Version which is working, weightsfile format is old-LPNN
#
# Revision 1.3  91/07/25  12:58:05  sloboda
# running version, which predicts nonsense.
#
# Revision 1.2  91/07/19  03:11:45  sloboda
# This version is not running yet. Most of the functions are tested. But some glue is missing.
# p_predict(), p_write_block, p_read_block should be finnished ...
#
# Revision 1.1  91/07/15  20:46:32  sloboda
# Initial revision
#
################################################################################################################################

/*== Includes ===================================================================================================================

#include <mpl.h>
#include <stdio.h>
#include <ppeio.h>              /* the parallel file i/o stuff */
#include <sys/types.h>          /* wird von sys/stat.h gebraucht */
#include <sys/stat.h>           /* stat  , information structure for unix files */
#include <sys/file.h>
#include <errno.h>
#include <string.h>
#include <math.h>

#include "ts_std.h"             /* some things they left out of C */
#include "lpnn_be.h"
#include "timing_be.h"
#include "fileIO_be.h"          /* low level, machine dependent file io routines  */
#include "MatVec_be.h"          /* matrix and vector operations, used for the NNs */

/*== Private Constants ==========================================================================================================

#define FrameSize 16

#define MaxLayers      4                        /* max. number of layers per neural network              */
#define MaxInputConn   16                       /* max. number of input connections to one neural network */
#define MaxMatrices    MaxLayers-1              /* max. number of weight matrices                        */
#define MaxThetas      MaxLayers-1              /* max. number of theta vectors, index starts with 0     */

#define MaxScores      48                       /* max. number of scores that can be calculated per PE   */
#define MaxFrames      1000                     /* max. number of 10 msec input frames of speech         */


#define PMODE          0666                     /* everybody can read and write */

#define READ_ONLY       0                       /* UNIX basic file i/o modi */
#define WRITE_ONLY      1
#define READ_WRITE      2


#define MODEL_FILENAME    "model"
#define NET_FILENAME      "network"
#define WEIGHTS_FILENAME  "weights"

#define SCORES_FILENAME   "data.scores"
#define SCORES_SEM_NAME   "data.scores-ready"
#define FFT_FILENAME      "data.FFT"
#define FFT_SEM_NAME      "data.FFT-ready"

/*== Private Types ==============================================================================================================

/*== Private Variables ==========================================================================================================

static char     RCSid[] = "$header$";

static int      NrNets,                                 /* actual number of NN's (depending on modelfile) */

                NrLayers,                               /* actual number of net layers                    */
                LayerSize       [MaxLayers],            /* size of each of the layers                     */
                FirstNeuron     [MaxLayers],            /* number of first neuron in layer                */
                LastNeuron      [MaxLayers],            /* number of last neuron in layer                 */

                NrConns,                                /* number of connections in the Weightmatrices    */
                NrProcsUsed,                            /* number of processors used for the NN's         */
                FramesPerRun,                           /* number of input frames, that are processed in parallel */
```

```
                NrInputConn,                            /* actual number of input connections to one NN    */
                InputConn       [MaxInputConn],         /* relative indices of the actual NN input frames  */
                PosFrameSkip,                           /* how many frames at the end should be skipped in the fft file */
                NegFrameSkip;                           /* how many frames at the beginning should be skipped in the fft file *

static int      NrWeights,                              /* the number of floats in all Weightsmatrices   */
                NrActivations,                          /* the number of floats in all ActivationVectors */
                NrThetas;                               /* the number of floats in all ThetaVectors      */
static
plural float *  WeightsPtr      [MaxMatrices],          /* pointers to plural weight matrices    */
              * ActivationsPtr  [MaxLayers],            /* pointers to plural activation vectors */
              * ThetasPtr       [MaxThetas],            /* pointers to plural thetas vectors, index starts with 0  */
                NominalFrame    [FrameSize],
                Score           [MaxScores];            /* the final score for each run and each PE is stored here */

static int      NrSpeechFrames,                         /* number of input "speech" frames                 */
                NrPredFrames;                           /* number of frames, that will be predicted        */
static float    SpeechFrame[MaxFrames * FrameSize];     /* array of input "speech" frames (FFT-frames)     */

static int      SizeOfFloat = sizeof(float),            /* = sizeof(float) only calculated at the beginning */
                SizeOfShort = sizeof(short);

static int      SizeOfFrame = FrameSize * sizeof(float);/* size of a frame in bytes */

static Timer    timer0;                                 /* for timing only ... */
static double   time0;

static int      PE = 1;                                 /*  FOR DEBUGGING ONLY !!!
                                                            to watch the processing element with the Nr. PE */

/*== Public Variables ==========================================================================================================

/*##############################################################################################################################
 #
 #      PRIVATE ROUTINES
 #
 ##############################################################################################################################

/*==============================================================================================================================
 |
 | read_Model                  PRIVATE   SINGULAR
 |
 | Description :
 |    Reads a "Version 2" model file, which describes all the phoneme models to be built.
 |
 | Parameter :
 |    modelfile   name of model file
 |
 | Globals initialized :
 |    NrNets      number of NN's to be modeled.
 |
 | Note :
 |    just the overall Number of NN's is interessting.
 |
 | History :
 |    4.Juli.91  Tilo     created
 |    2.Sept 91  Tilo     changed file format to BAWL format version 2
 |   27.Sept 91  Tilo     changed the test for the version number
 |
 +=============================================================================================================================*/

void

read_Model (modelfile)
  char    *modelfile;
{
  FILE    *f; /* modelfile pointer */
  int     Result;                                 /* for printf results*/
  int     models,phonemes;
  char    version[16], str[48];

/*------------------------------------------------------------------------------------------------------------------------------
 |  The model file has two parts.  Just read the number of neural nets.
 +-----------------------------------------------------------------------------------------------------------------------------*/

  f = fopen(modelfile,"r");             /* open modelfile */
  if (f == NULL) then { fprintf(stderr, "\n >>> couldn't open modelfile ! \n\n");
                        exit(-1);               /* ooops ! */
                      }

  printf ("  Reading model file : \"%s\" \n", modelfile);
  fflush(stdout);

  /* read "Version 2 model file" :  **  note : MasPar's fscanf has a bug : doesn't append a null char to strings ! **  */
  Result = fscanf (f, "%s %s %s %s", str, version, str, str);

  if      (Result==EOF) then { fprintf(stderr, "\n >>> unexpected EOF ! \n\n");
                               exit(-1);                /* ooops ! */
                             }
  else if (Result==0)    then                            /* prohibit old file formats */
          { fprintf(stderr, " >>> Result is 0 ; please change this file to use the format \"Version 2\" !\n\n");
            exit(-1);
          }

  if (version[0] != '2')                                /* strneq(version, "2") */
     then { fprintf(stderr, "\n >>> Version \"%s\" ; please change this file to use the format \"Version 2\" !\n",version);
```

```
            fprintf(stderr, " >>> strlen(%s) = %d \n", version, strlen(version) );
            fprintf(stderr, " >>> s[0]=%x s[1]=%x \n", version[0], version[1]);
            fprintf(stderr, " >>> strlen(%s) = %d \n\n", "2", strlen("2") );
            exit(-1);
          }

  printf("  Version %c\n", version[0]);

  freadOK (fscanf (f, " %d %s", &models, str)); /* read "2 models" or "1 model" */
  printf ("  %d models", models);

  freadOK (fscanf (f, " %d %s", &phonemes, str)); /* read "40 phonemes" */
  printf ("  %d phonemes", phonemes);

  freadOK (fscanf (f, " %d %s", &NrNets, str)); /* read "118 nets" */
  printf ("  %d nets\n", NrNets);

  fclose (f); /* close modelfile */
  printf("  finished reading the modelfile and closed it.\n\n");
}

/*=============================================================================================================================
  |
  | read_Net                    PRIVATE   SINGULAR
  |
  | Description :
  |   reads netfile and creates the datastructures for the plural NN's.
  |   In the weightsmatrices, only the used connections are initialized to 1, unused connections to 0.
  |
  | Parameters : netfile    name of the netfile
  |
  | Globals initialized :
  |   NrInputConns, InputConns, LayerSize, FirstNeuron, LastNeuron, NegFrameSkip, PosFrameSkip, NominalIdx,
  |   NrConns, Weightmatrices are pre-initialized
  |
  | Globals assumed :
  |   NrNets must be initialized before read_Net is called. It is necessary for the plural malloc.
  |
  | History :
  |    8.Juli 91  Tilo     created
  |    2.Sept 91  Tilo     changed file format to BAWL format version 1
  |   27.Sept 91  Tilo     changed the test for the version number
  |   24.Oct  91  Tilo     changed the call to p_put_MatrixElement
  |
  +============================================================================================================================*/

void

read_Net(netfile)
   char   * netfile;
{
   FILE   * f;

   char     version[16],
            str[100];
   int      Result;                              /* for printf results*/
   int      i, a, b;

/*-----------------------------------------------------------------------------------------------------------------------------
  |  The net file has two parts. First we read what version, which input frames and how many layers each NN has
  +-----------------------------------------------------------------------------------------------------------------------------*/

  f = fopen(netfile,"r");
  if (f == NULL) then { fprintf(stderr, "\n >>> couldn't open netfile ! \n\n");
                        exit(-1);               /* ooops ! */
                      }

  printf("  Reading net file : \"%s\" \n", netfile);
  fflush(stdout);

/*-----------------------------------------------------------------------------------------------------------------------------
  | read the header "version ..."
  +-----------------------------------------------------------------------------------------------------------------------------*/

  /* read "Version 1 network file" :  **  note : MasPar's fscanf has a bug : doesn't append a null char to strings ! **   */
  Result = fscanf (f, "%s %s %s %s", str, version, str, str);

  if      (Result==EOF) then { fprintf(stderr, "\n >>> unexpected EOF ! \n\n");
                               exit(-1);              /* ooops ! */
                             }
  else if (Result==0)   then                          /* prohibit old file formats */
          { fprintf(stderr, " >>> Result is 0 ; please change this file to use the format \"Version 1\" !\n\n");
            exit(-1);
          }

  if (version[0] != '1')                              /* strneq(version, "1") */
     then { fprintf(stderr, "\n >>> Version %s ; please change this file to use the format \"Version 1\" ! \n\n", version);
            exit(-1);
          }

  printf("  Version %c\n", version[0]);

/*-----------------------------------------------------------------------------------------------------------------------------
  | read the input frames of the NN
```

```
 freadOK (fscanf (f, "%d %s %s", &NrInputConn, str, str)); /* read "4 input frames:" */
 printf("  %d input frames: ", NrInputConn);

 for (i=0; i<NrInputConn; i++)   /* loop... */
  {
   freadOK (fscanf (f, "%d", &InputConn[i])); /*    read "-2 -1  1  2" = list of relative frame offsets */
   printf( "%d ",InputConn[i]);
  }

 if (InputConn[0] < 0)
    then NegFrameSkip = (- InputConn[0]);
    else NegFrameSkip = 0;

 if (InputConn[NrInputConn-1] > 0)
    then PosFrameSkip = InputConn[NrInputConn-1];
    else PosFrameSkip = 0;

/*-------------------------------------------------------------------------------------------------------------------------------
 | read the number and the size of the layers;
 |
 | initialize : LayerSize, FirstNeuron, LastNeuron
 +------------------------------------------------------------------------------------------------------------------------------*/

 freadOK (fscanf (f, "%d %s %s", &NrLayers, str,str)); /* read "3 network layers:" */
 printf( "\n  %d network layers: ", NrLayers);

 for (i=0; i<NrLayers; i++)   /* loop... */
  {
   freadOK (fscanf (f, "%d", &LayerSize[i])); /*    read "64 12 16" = list of layersizes  */
   printf( "%d ",LayerSize[i]);
  }

 /* initialize : FirstNeuron, LastNeuron : */
 FirstNeuron[0] = 1;
 LastNeuron[0]  = LayerSize[0];
 for (i=1; i<NrLayers; i++)
   { FirstNeuron[i] = LastNeuron[i-1] + 1;
     LastNeuron[i]  = LastNeuron[i-1] + LayerSize[i];
   }

 freadOK (fscanf (f, "%s %d - %d", str, &a, &b));          /* read "in:   0 - 64" , and ignore it */
 freadOK (fscanf (f, "%s %d - %d", str, &a, &b));                          /* read "out: 77 - 92" , and ignore it */

/*-------------------------------------------------------------------------------------------------------------------------------
 | calculate the amount of memory for the vectors and matrizes, allocate it in the PE's memory, initialize the pointers
 +------------------------------------------------------------------------------------------------------------------------------*/

 allocate_PE_mem (LayerSize, NrLayers);                          /* all allocated memory is initialized to 0.0 */

/*-------------------------------------------------------------------------------------------------------------------------------
 | read the connections, that are used,  initialize NrConns
 +------------------------------------------------------------------------------------------------------------------------------*/
 {
       int  fromA, fromZ, toA, toZ;
       int  MatIdx, iStart, iStop, jStart, jStop;
       int  i, j, Erg;

  NrConns = 0;                                          /* to count overall number of connections, we have in the NN */

  Erg = fscanf (f, "%d - %d : %d - %d", &a,&b, &toA, &toZ);                    /* read thetas   "0 - 0 : 65 - 92" */
/*
 *  printf("%d - %d : %d - %d\n",a,b,toA,toZ);
 */
  while (4 == fscanf (f, "%d - %d : %d - %d", &fromA, &fromZ, &toA, &toZ))     /* read wheights "1 - 32 : 33 - 42" */
    { /*-------------------------------------------------------------------------------------------
       | test whether fromA, fromZ belong to the same layer L1, whether toA, toZ belong to the same layer L2
       | and whether L1, L2 belong to the same matrix :
       +-------------------------------------------------------------------------------------------*/
    /*
     *  printf("%d - %d : %d - %d\n", fromA,fromZ,toA,toZ);
     */

      if ( ValidConnections( fromA,fromZ, toA,toZ, &MatIdx ,&iStart, &iStop, &jStart, &jStop ) )
         then for (i=iStart; i<=iStop; i++)
                for (j=jStart; j<=jStop; j++)
                 { NrConns++;                                                  /* count the connections, we have      */
                                                                               /* WeightsMatrix[MatIdx][i][j] = 1.0; */
                   p_put_MatrixElement(WeightsPtr[MatIdx], LayerSize[MatIdx],
                                       i, j, (plural float)1.0);
                 }

         else fprintf(stderr,"\n >>> garbage in netfile ! %d - %d : %d - %d\n", fromA,fromZ, toA,toZ);   /* fatal error */
    }
 }

  fclose (f); /* close network file */
  printf("  %d connections are used\n", NrConns);
  printf("  finished reading the netfile and closed it.\n\n");

  /* printMatrix(0, PE, "matrix 64 x 12 :"); */

}
/*==============================================================================================================================
```

```
|
| allocate_PE_mem          PRIVATE  SINGULAR
|
| Description :
|   Calculates the amount of memory, that is needed for the neural nets
|   allocates the memory, initializes the memory and the pointers into it.
|   if there is not enough memory, the program will be aborted with an error message.
|
| Parameters :
|   NrLayers
|   LayerSize
|
| Globals initialized :
|   WeightsPtr, ActivationsPtr, ThetasPtr
|   NrWeights,  NrActivations,  NrThetas       the number of floats
|
| History :
|   16.Juli 91  Tilo     created
|    2.Dec  91  Tilo     removed a bug in the malloc() call
|
+=============================================================================================================================

int

allocate_PE_mem(LayerSize, NrLayers)
     int LayerSize[], NrLayers;
{
     unsigned         MemUnits, MemBytes, WeightsSize, ActivationsSize, ThetasSize;
     int              i;
     plural float   * fp, * tp;

  /* --- calculate the amount of memory needed : --- */

  MemUnits = WeightsSize = ActivationsSize = ThetasSize = 0;

  /* the weights matrizes : */
  for (i=0; i<NrLayers-1; i++)
      WeightsSize += (LayerSize[i]*LayerSize[i+1]);

  /* the activation vectors : */
  for (i=0; i<NrLayers; i++)
      ActivationsSize += LayerSize[i];

  /* the theta vectors : */
  for (i=1; i<NrLayers; i++)                                  /* ThetasPtr[0] points to theta vector of second layer */
      ThetasSize += LayerSize[i];

  MemUnits = WeightsSize + ActivationsSize + ThetasSize;      /* overall sum of floats */
  MemBytes = MemUnits * SizeOfFloat;                          /* overall sum in bytes  */

  NrThetas      = ThetasSize;                                 /* initialize global variables */
  NrWeights     = WeightsSize;
  NrActivations = ActivationsSize;

  printf("\n  NrWeights %d , NrActivations %d , NrThetas %d\n", NrWeights, NrActivations, NrThetas);

  printf("  %d floats will be allocated\n",MemUnits);

  /* ------ allocate the memory : ------ */

  fp = (plural float *) p_malloc(MemBytes);

  if (fp==NULL)
     then { fprintf(stderr,"\n >>> ERROR in \"read_Net\": couldn't allocate enough PE-memory !\n");
            exit(-1);
          }
     else { tp = fp;                                          /* initialize the memory to 0.0 */
            for (i=0; i<MemUnits; i++)
                *tp++ = 0.0;
          }

  /* --- initialize the pointers to the weight matrices : --- */

  WeightsPtr[0] = fp;

  for (i=1; i<NrLayers-1; i++)
      WeightsPtr[i] =  WeightsPtr[i-1] + (LayerSize[i-1]*LayerSize[i]);

  /* --- initialize the pointers to the activation vectors : --- */

  ActivationsPtr[0] = fp + WeightsSize;

  for (i=1; i<NrLayers; i++)
      ActivationsPtr[i] = ActivationsPtr[i-1] + LayerSize[i-1];

  /* --- initialize the pointers to the theta vectors : --- */

  ThetasPtr[0] = fp + WeightsSize + ActivationsSize;

  for (i=1; i<NrLayers-1; i++)
      ThetasPtr[i] = ThetasPtr[i-1] + LayerSize[i];

}

/*=============================================================================================================================
 |
```

```
| free_PE_mem          PRIVATE  SINGULAR
|
| Description :
|   frees the allocated memory.
|
| Globals used :
|   WheightsPtr        are all set to NIL
|   ActivationsPtr     are all set to NIL
|   ThetasPtr          are all set to NIL
|
| History :
|   16.Juli 91  Tilo     created
|
+=============================================================================================================================

plural void

free_PE_mem()
{
     int i;

  free(WeightsPtr);

  for (i=0; i<MaxMatrices; i++)
     WeightsPtr[i] = NIL;

  for (i=0; i<MaxLayers; i++)
     {
        ActivationsPtr[i] = NIL;
        ThetasPtr[i] = NIL;
     }

}

/*============================================================================================================================
|
| ValidConnections              PRIVATE  SINGULAR
|
| Description :
|   Tests whether fromA, fromZ belong to the same layer L1, whether toA, toZ belong to the same layer L2
|   and whether L1, L2 belong to the same matrix.
|   If all this is true, then the index of the Weightsmatrix is returned and the start and stop values for both
|   indices i,j in this matrix.
|
| Parameters :
|   IN :
|        fromA,fromZ,
|        toA,toZ,
|   OUT:
|       &MatrixIdx       the index of the matrix, to which the connection list belongs to.
|       &from_i, &to_i,  start, stop values for the index i  in the Weightsmatrix
|       &from_j, &to_j   start, stop values for the index j  in the Weightsmatrix
|
| Globals used :
|    NrLayers
|    FirstNeuron
|    LastNeuron
|
| History :
|   18.July 91  Tilo     created
|
+=============================================================================================================================

bool

ValidConnections( fromA,fromZ, toA,toZ,          /* IN  */
                  MatrixIdx,                     /* OUT */
                  from_i, to_i, from_j, to_j )   /* OUT */
      int   fromA, fromZ, toA, toZ,
          * MatrixIdx,
          * from_i, * to_i, * from_j, * to_j;
{
      int   i, j;
      bool  ok;

   *MatrixIdx = 0;
   *from_i = 0; *to_i = 0;
   *from_j = 0; *to_j = 0;

   ok = FALSE;
   for (i=0; ((i<NrLayers) && !ok); i++)
     ok = (  (fromA <= fromZ)                                        /* does fromA come before fromZ ? */
          && (fromA >= FirstNeuron[i]) && (fromA <= LastNeuron[i])   /* is fromA in this layer ?       */
          && (fromZ >= FirstNeuron[i]) && (fromZ <= LastNeuron[i])   /* is fromZ in this layer too ?   */
          );
   i--;                                                              /* to get the last iteration's i */
   if (! ok) then return(FALSE);

   ok = FALSE;
   for (j=0; ((j<NrLayers) && !ok); j++)
     ok = (  (toA <= toZ)                                            /* does toA come before toZ ? */
          && (toA >= FirstNeuron[j]) && (toA <= LastNeuron[j])       /* is toA in this layer ?     */
          && (toZ >= FirstNeuron[j]) && (toZ <= LastNeuron[j])       /* is toZ in this layer too ? */
          );
   j--;                                                              /* to get the last iteration's j */
   if (! ok)
```

```
      then return(FALSE);
      else if (i+1==j)                                          /* are layer i and j successors ?  */
              then { *MatrixIdx=i;                              /* then give back the matrix index */
                     *from_i = fromA - FirstNeuron[i];          /* and the start,stop values for   */
                     *to_i   = fromZ - FirstNeuron[i];          /* the indices i, j                */
                     *from_j = toA - FirstNeuron[i+1];
                     *to_j   = toZ - FirstNeuron[i+1];
                     return(TRUE);
                   }
              else return(FALSE);
}

/*+=============================================================================================================================
  |
  | read_Weights              PRIVATE  SINGULAR
  |
  | Description :
  |   reads netfile and loads the weights into the datastructures for the plural NN's.
  |   First some rubbish and the thetas are read sequentially, then the weights are read
  |   in parallel. The weights are stored at the position, that is determined
  |   by the next position of a 1 in the weightsmatrix.
  |   The whole machine is used for replications of NN's. All NN's have the same topology.
  |
  | Parameters : weightsfile    the name of the weightsfile.
  |
  | Globals initialized :
  |   NrProcsUsed
  |   FramesPerRun
  |
  | Globals assumed :
  |   The weightsmatrices must be initialized before read_Weights is called.
  |   It is assumed that the used connections in the weightsmatrices are initialized to 1 (not necessary in this version),
  |   all other connections are assumed to be 0. This initialization is done by read_Net().
  |
  | Implementation Note :
  |   For all Nets 0..NrNets-1 the thetas and weights are read.
  |
  | Note :
  |   low-level UNIX i/o routines are used instead of file i/o routines, because MasPar doesn't support
  |   the higher level routines.
  |
  | History :
  |   18.July 91  Tilo     created, already for transposed matrices.
  |   24.July 91  Tilo     errors corrected...
  |   25.July 91  Tilo     rearranged ... Joe said, that it is an other order of the weights in the weights-file.
  |   27.July 91  Tilo     no, it was still a wrong order. Here's an other one ... it works ... ! (old lpnn format)
  |    3.Sept 91  Tilo     changed the file format to BAWL format version 0
  |   10.Oct  91  Tilo     introduced timing of this routine.
  |   11.Oct  91  Tilo     new way of reading the weights (one large Buffer per NN is read sequentially).
  |                        This is faster, because the MPDA isn't installed correct (needs one more board to speed it up).
  |                        This routine is faster than the old one, on MasPars without MPDA.
  |
  +=============================================================================================================================*/

void read_Weights(weightsfile)
     char         * weightsfile;
{
     int            wfd;                            /* file descriptor for weightsfile */
     int            rows, len, i, j, Iterations,
                    ThetasRead, WeightsRead;        /* to keep track of the thetas, weights, we read from file */
     int            BytesPerNN, FloatsPerNN, NN;    /* for handling the float-buffer for the sequentiall read operations */
     long           skip, offset;
     plural float   *t, *w, *wEnd;
     float          *Buffer, *BufferPtr;
     Timer          timer;


/*-------------------------------------------------------------------------------------------------------------
  | allocate the buffer for the floats of one NN
  +-------------------------------------------------------------------------------------------------------------*/

 FloatsPerNN = NrThetas + NrConns;                  /* the size of a block is NrThetas plus number of the connections used */
 BytesPerNN  = SizeOfFloat * FloatsPerNN;

 Buffer = (float *) malloc( BytesPerNN );
 if (Buffer==NULL)
    then { fprintf(stderr,"\n >>> ERROR in \"read_Weights\": couldn't allocate enough ACU-memory (%d bytes) !\n",BytesPerNN);
           exit(-1);
         }

/*-------------------------------------------------------------------------------------------------------------
  | open the weights file, start timer
  +-------------------------------------------------------------------------------------------------------------*/

  wfd = open (weightsfile, READ_ONLY);
  readOK(wfd, weightsfile);                                     /* aborts if there was any error */

  printf("  Reading weights file : \"%s\" \n", weightsfile);

  reset_timer(&timer);
  start_timer(&timer);

/*-------------------------------------------------------------------------------------------------------------
  |  read the version number, then the two filenames at the beginning  and the number of iterations. (singular)
  +-------------------------------------------------------------------------------------------------------------*/
```

```
{    char  str[256];

 read_string(wfd, str, &len);                    /* read the Version-Number      */
 if ( strneq(str, "Version 0") &&
      strneq(str, "Version 1") )
    then { fprintf(stderr,"\n >>> ERROR in \"read_Weights\": Can't handle this old file format !\n");
           exit(-1);
         }
 printf("  %s\n", str);                          /* print the version number */
 skip = len;

 read_string(wfd, str, &len);                    /* ignore the network - filename    */
 printf("  %s , ", str);
 skip += len;

 read_string(wfd, str, &len);                    /* ignore the model - filename  */
 printf("%s , ", str);
 skip += len;

 Iterations = read_short(wfd);                   /* number of iterations, in the training */
 printf("%d iterations\n", Iterations);
 skip += SizeOfShort;
}
/* printf("  %d leading bytes in weightsfile will be skipped.\n", skip); */

/*-------------------------------------------------------------------------------------------------------------------------------
 |  position the singular file pointer ... and here we go ...
 +------------------------------------------------------------------------------------------------------------------------------*/

 FramesPerRun = nproc / NrNets;                  /* FramesPerRun = number of PE's / number of NN's in the model */
 NrProcsUsed = NrNets * FramesPerRun;            /* calculate how many PE's we need for this task */

 printf("  predictions for %d frames will be calculated per run.\n  %d predictions for each frame\n", FramesPerRun, NrNets);
 printf("  %d PE's will be used.\n", NrProcsUsed);
 printf("  reading the weights ... \n");

 if (iproc < NrProcsUsed) then                   /* for all Nets do in parallel :          (deactivate unused PE's) */
    {
     /*-------------------------------------------------------------------------------------------------------------------------
      | skip the singular filepointer over the leading header :
      +-------------------------------------------------------------------------------------------------------------------------*/
      offset = lseek(wfd, skip, L_SET);                          /* skip the leading bytes, we allready read */
      if (offset == -1L)
                { /* perror("Fehler beim lseek"); */
                  printf("Fehler %d beim lseek\n", errno);
                  exit(-1);
                }

    /* printf("\n skipped over header and positioned filepointers\n"); */

     /*-------------------------------------------------------------------------------------------------------------------------
      | for all PEs, which will calculate the same neural network :
      +-------------------------------------------------------------------------------------------------------------------------*/

      for (NN=0; NN < NrNets; NN++)                               /* for Neural Nets  0..NrNets-1                 */
        if ((iproc mod NrNets)==NN) then                          /*   activate all PEs in this equivalence class */
          {
            /*-----------------------------------------------------------------------------------------------------------------
            | sequentially read the weights and biases for the neural network NN into one large buffer
            +-----------------------------------------------------------------------------------------------------------------*/

            if (read(wfd, Buffer, BytesPerNN) == -1 )
               then { perror("ERROR in \"read_Weights\": couldn't read buffer"); exit(-1);}

            /*-----------------------------------------------------------------------------------------------------------------
            | for all neurons, assemble the inputs (bias and weights) :
            +-----------------------------------------------------------------------------------------------------------------*/

            ThetasRead = 0;  t = ThetasPtr[0];
            WeightsRead = 0; w = WeightsPtr[0];
            wEnd = WeightsPtr[0];
            BufferPtr = Buffer;

            for (i=0; i<NrLayers-1; i++)                          /* for all matrizes             */
              { rows = LayerSize[i+1];                            /*    target neuron Nr          */
                len  = LayerSize[i];                              /*    source neuron Nr          */

                for (j=0; j<rows; j++)                            /*    for all target neurons                */
                  {
                    READ_FLOAT_SWAP(*BufferPtr);                  /*      swap the bytes of next float, if necessary */
                    *t++ = *BufferPtr++;                          /*      plural copy one bias (theta)              */
                    ThetasRead++;

                /*  printf("Theta %2d = %8.5f\n", ThetasRead, proc[PE].*--t); t++; */

                    wEnd += len;
                    while (w < wEnd)                              /*      while connections left */
                      if (*w == (plural float) 1.0)
                         then {
                                READ_FLOAT_SWAP(*BufferPtr);      /*        swap the bytes of next float, if necessary */
                                *w++ = *BufferPtr++;              /*        plural copy weight, step to next position  */
                                WeightsRead++;
                              }
                         else w++;
                  }
              }
```

```
            }

        stop_timer(&timer);
        print_timer("  elapsed time :", &timer);

        /* The following assertions do still hold, because the matrices are all initialized in parallel,
           which results in the same number of "ones" in equivalent matrices in the different PEs.
         */
        Assert((ThetasRead  == NrThetas),        "[0] in read_weights");    /* test if everything is ok */
        Assert((t <= ThetasPtr[0] + NrThetas),   "[1] in read_weights");
        Assert((WeightsRead == NrConns),         "[2] in read_weights");
        Assert((NrConns <= NrWeights),           "[3] in read_weights");

        printf("  finished reading the weightsfile and closed it.\n\n");

       /*-------------------------------------------------------------------------------------------------------------
        | free the allocated buffer
        +------------------------------------------------------------------------------------------------------------*/
        free(Buffer);
      }
 /*     FOR DEBUGGING :
    printMatrix(0, PE,"matrix 64 x 12 :");
    printMatrix(1, PE,"matrix 12 x 16 :");
    printVector(ThetasPtr[0], 12, PE, "theta vector 1 :");
    printVector(ThetasPtr[1], 16, PE, "theta vector 2 :");
 */
}

/*=============================================================================================================
  |
  | read_FFT              PRIVATE  SINGULAR
  |
  | Description :
  |   reads FFT file
  |
  | Parameters : fftfile    the name of the FFT file.
  |
  | Globals used :
  |   NegFrameSkip
  |
  | Globals initialized :
  |   NrSpeechFrames, NrPredFrames
  |   SpeechFrame[]
  |
  | History :
  |   25.July 91  Tilo   created.
  |   27.Sept 91  Tilo   changed it to run on a MasPar with DEC front end.
  |   24.Oct  91  Tilo   simplified the code, by using the macro READ_FLOAT_SWAP
  |
  +============================================================================================================*/

 void read_FFT(fftfile)
      char * fftfile;
{
      FILE  * f;
      float * p;
      int     i, erg;

   p = SpeechFrame;                                          /* pointer to the first speech-frame */
   NrSpeechFrames = NrPredFrames = 0;
   f = fopen(fftfile, "r");
   if (f != NULL)
     {
       printf("\n  Reading the FFT file : \"%s\" \n", fftfile);

       while (0 < (erg = fread(p, SizeOfFrame, 1, f)) )        /* read a whole frame */
         { for (i=0; i<FrameSize; i++)
             READ_FLOAT_SWAP( *(p+1) );

           NrSpeechFrames++;
         }
       fclose(f);
       printf("  %d frames, covering %6.3f seconds of speech.\n",
              NrSpeechFrames, ((float)NrSpeechFrames/100) );
       printf("  finished reading the FFT file and closed it.\n\n");
     }
}

/*=============================================================================================================
  |
  | p_predict_1             PRIVATE  SINGULAR
  |
  | Description :
  |
  |
  | Globals used :
  |   NegFrameSkip
  |
  | Globals initialized :
  |   NrSpeechFrames, NrPredFrames
  |   SpeechFrame[]
  |
  | History :
  |   25.July 91  Tilo     created.
  |
  +============================================================================================================*/
```

```
void p_predict_1(run)
     int run;
{
     int n;

  for (n=0; n<NrLayers-1; n++)
    {
      p_Multiply_Vector_by_sparse_Matrix( ActivationsPtr[n], WeightsPtr[n], LayerSize[n], LayerSize[n+1],
                                          ActivationsPtr[n+1] );

      p_Add_Vectors( ActivationsPtr[n+1], ThetasPtr[n],  LayerSize[n+1],
                     ActivationsPtr[n+1] );

      p_sigmoid( ActivationsPtr[n+1], LayerSize[n+1] );

    }

  /* calculate distance for run i : */
  Score[run] = p_distance( NominalFrame, ActivationsPtr[NrLayers-1], LayerSize[NrLayers-1] );

}

/*=============================================================================================================================
  |
  | p_predict              PRIVATE  SINGULAR
  |
  | Description :
  |
  |
  | Globals used :
  |   NegFrameSkip
  |
  | Globals initialized :
  |   NrSpeechFrames, NrPredFrames
  |   SpeechFrame[]
  |
  | History :
  |   25.July 91  Tilo     created.
  |
  +============================================================================================================================*/

void p_predict()

{
     int          i, j, k, r,
                  EndFrame, run,
                  delta[MaxInputConn+1];                        /* theee arrays are too large, but NrInputConn  */
     plural float * map[MaxInputConn+1],                        /* which would be appropriate is no constant    */
                  * p;
            float * q;

     plural int   RelativeFrameNr;

all
 {
  for (i=0; i<MaxScores; i++)                                   /* initialize the scores */
      Score[i] = 0.0;                                           /* I should initialize it with the nomonal value */

  NrPredFrames = NrSpeechFrames - NegFrameSkip - PosFrameSkip;  /* calculate how many frames must be predicted */

  printf("  ==> %d frames will be predicted, %d at a time.\n\n", NrPredFrames, FramesPerRun);

  RelativeFrameNr = (iproc / NrNets);                           /* relative frame Nr. of the frame, each PE is working on *

  /* initialize the deltas : */
  for (i=0; i<NrInputConn; i++)
      delta[i] = InputConn[i];
  delta[NrInputConn] = 0;

  /* initialize the mapping according to the deltas : */
  for (i=0; i<NrInputConn; i++)
      map[i] = ActivationsPtr[0] + FrameSize * i;
  map[NrInputConn] = NominalFrame;

  if (delta[0] < 0)
     then j = - delta[0];                                       /* the number of the first frame to be predicted */
     else j = 0;

  if (delta[NrInputConn-1] > 0)
     then EndFrame = NrSpeechFrames - delta[NrInputConn-1];     /* the frame after the last frame to predict */
     else EndFrame = NrSpeechFrames;

  reset_timer(&timer0);
  start_timer(&timer0);
  run = 0;
  do {  r = 0;                                                  /* number of the current frame in this run */
        do { if (RelativeFrameNr == r)
               for (i=0; i<NrInputConn+1; i++)
                 { p = map[i];
                   q = (SpeechFrame + (j+delta[i]) * FrameSize);    /* do the mapping ... cha, cha */
                   for (k = 0; k<FrameSize; k++)
                       *p++ = *q++;                             /* copy frame to the ActivationVector and Nominal frame */
                 }
             r++; j++;
           } while ((r < FramesPerRun) && (j < EndFrame));
```

```
        if (RelativeFrameNr < r)
            p_predict_1(run);                                   /* calculate the predictions and the distances */
        run++;
    }  while (j < EndFrame);

  time0 = stop_timer(&timer0);
  printf("  elapsed time for the parallel predictions : %10.3f seconds\n\n", time0);

 /*
  * printf("  connections : %10.3lf ; connections per second : %10.3lf \n\n",
  *        (double)(NrNets*NrConns*NrPredFrames), ((double)(NrNets*NrConns*NrPredFrames)/ time0) );
  */

 }
}


/*##########################################################################################################################################
 #
 #      PUBLIC ROUTINES
 #
 ##########################################################################################################################################*/

/*=========================================================================================================================================
 |
 | p_write_BrainDump          EXPORTED   PLURAL
 |
 | Description :
 |   writes all vectors and matrices, of all PE's (one PE after the other - in iproc order)
 |   onto a file on the MPDA. (MasPar Parallel Disk Array)
 |
 | Parameters :
 |             All Variables, that will be stored ....
 |
 | History :
 |   24.Oct  91  Tilo     created
 |
 +=========================================================================================================================================*/


/*=========================================================================================================================================
 |
 | p_read_BrainDump          EXPORTED   PLURAL
 |
 | Description :
 |   reads all vectors and matrices, of all PE's (one PE after the other - in iproc order)
 |   from a file on the MPDA. (MasPar Parallel Disk Array)
 |
 | Parameters :
 |             All Variables, that will be read ...
 |
 | History :
 |   24.Oct  91  Tilo     created
 |
 +=========================================================================================================================================*/


/*=========================================================================================================================================
 |
 | p_write_scores          EXPORTED   PLURAL
 |
 | Description :
 |   writes all activation vectors of the last layer, of all PE's (one PE after the other - in iproc order)
 |   onto a file on the MPDA. (MasPar Parallel Disk Array)
 |
 | Parameters : scoresfile   name of the scoresfile
 |
 | Globals used :
 |   SizeOfFloat
 |   Score
 |   NrNets
 |   NrPredFrames
 |   NrProcsUsed
 |   FramesPerRun
 |
 | History :
 |   19.July 91  Tilo     created
 |   21.July 91  Tilo     finished it.
 |   24.Oct  91  Tilo     removed the call of p_write_float. p_WRITE_FLOAT_SWAP is used instead.
 |
 +=========================================================================================================================================*/
visible extern

void p_write_scores(scoresfile)
    char * scoresfile;
{
    int    fd, i,
           FullRuns, Rest;
    FILE * xd;                  /* NUR ZUM TEST */
    plural float f;

all                                                             /* select all processing elements */
 {
  fd = creat(scoresfile, PMODE);                                /* create new file or overwrite old file */
```

```
  readOK(fd, scoresfile);                                          /* aborts, if there was any error        */

  /* xd = fopen("/da0a/janus/ascii.scores", "w"); */                  /* create new file or overwrite old file */

  FullRuns = NrPredFrames / FramesPerRun;     /* DIV ; number of runs, where the machine was filled  */
  Rest     = NrPredFrames % FramesPerRun;     /* MOD ; number of frames in the extra run ...         */

  if (iproc < NrProcsUsed) then               /* for all frames, that filled the machine... ; deactivate unused PE's */
     for (i=0; i<FullRuns; i++)               /* for each frame ... */
      {
      /*  p_fprintf(xd,"%1.17f\n", Score[i]);
       */
        f = Score[i];
        p_WRITE_FLOAT_SWAP(f);
        p_write (fd, &f, SizeOfFloat);        /* write the scores in iproc order */
      }

  if (iproc < (Rest * NrNets)) then           /* write the rest ... ; deactivate unused PE's */
      {
      /*  p_fprintf(xd,"%1.17f\n", Score[i]);
       */
        f = Score[i];
        p_WRITE_FLOAT_SWAP(f);
        p_write (fd, &f, SizeOfFloat);        /* write the scores in iproc order */
      }

  close(fd);

 /* fclose(xd); */

 }
}

/*+=============================================================================================================================+
  |                                                                                                                             |
  | init_MasPar              EXPORTED   SINGULAR                                                                                |
  |                                                                                                                             |
  | Description :                                                                                                               |
  |   initializes the back end, reads init files and allocates memory ...                                                       |
  |                                                                                                                             |
  | Globals :    will be initialized after this routine.                                                                        |
  |                                                                                                                             |
  | History :                                                                                                                   |
  |   19.July 91  Tilo     created                                                                                              |
  |                                                                                                                             |
  +=============================================================================================================================*/

void init_MasPar()
{
 /* in this first version, I use static memory for the speech sample ... */

  read_Model( MODEL_FILENAME );

  read_Net( NET_FILENAME );

  read_Weights( WEIGHTS_FILENAME, NET_FILENAME );

}

/*+=============================================================================================================================+
  |                                                                                                                             |
  | mpl_main              EXPORTED   SINGULAR                                                                                   |
  |                                                                                                                             |
  | Description :                                                                                                               |
  |   sceduling routine for the whole job ...                                                                                   |
  |                                                                                                                             |
  | History :                                                                                                                   |
  |   22.July 91  Tilo     created                                                                                              |
  |   24.Sept 91  Tilo     now the type of the front end is checked and displayed.                                               |
  |   27.Sept 91  Tilo     timing for p_write_scores introduced,  included the copyright notice                                  |
  |                                                                                                                             |
  +=============================================================================================================================*/
visible extern

void mpl_main()
{
#ifndef BRUTE
     visible extern int GiveUp();
#endif

      FILE  * fft_sem;
      int     fd_scores_sem;
      char    mach[40];
      Timer   timer1;
      double  time1;


  /* DO NOT CHANGE : */

  printf("\n  +-------------------------------------------------------------------------+\n");
  printf(  "  |                                                                         |\n");
  printf(  "  |                                                                         |\n");
  printf(  "  |      Parallel Predictions for LPNN based Speech Recognition.            |\n");
  printf(  "  |                                                                         |\n");
  printf(  "  |                                                                         |\n");
```

```
  printf(  "  |  Copyright (C) 1991 by  Tilo Sloboda,   (sloboda@ira.uka.de)           |\n");
  printf(  "  |                                                                       |\n");
  printf(  "  |  All rights reserved - see Copyright file. NO CITING BEFORE JAN. 1992  |\n");
  printf(  "  |                                                                       |\n");
  printf(  "  |  This software was developed at the                                   |\n");
  printf(  "  |                                                                       |\n");
  printf(  "  |                          University of Karlsruhe                      |\n");
  printf(  "  |                          Dept. of Informatics                         |\n");
  printf(  "  |                          Inst. f. Program Structures                  |\n");
  printf(  "  |                          and Data Organisation                        |\n");
  printf(  "  |                          P.O. Box. 6980                               |\n");
  printf(  "  |                          7500 Karlsruhe 1                             |\n");
  printf(  "  |                          WEST GERMANY                                 |\n");
  printf(  "  |                                                                       |\n");
  printf(  "  |  This software is part of a parallel JANUS implementation on a        |\n");
  printf(  "  |  MasPar machine, based on Joe Tebelskis LPNN JANUS system.            |\n");
  printf(  "  |                                                                       |\n");
  printf(  "  |  It may be used for demo purposes by members of the JANUS project only. |\n");
  printf(  "  |                                                                       |\n");
  printf(  "  +-----------------------------------------------------------------------+\n\n");

  printf("  Checking type of MasPar front end ... ");
  check_machine_type(mach);                                   /* check, whether the program was compiled right */
  printf("it's a %s front end !\n\n", mach);

  init_MasPar();                                              /* initialize global variables , allocate memory */

  while (TRUE)
    {
      printf("  ------------------------------------------------------------------------------\n\n");
      printf("  start polling for input file ..."); fflush(stdout);

      fft_sem = fopen( FFT_SEM_NAME, "r");                    /* wait for FFT semaphore file */
      while (fft_sem == NULL)
        {
          fft_sem = fopen( FFT_SEM_NAME, "r");                /* wait for FFT semaphore file */

#ifndef BRUTE
          callRequest( GiveUp, 0);                            /* be a nice process - give up sometimes.     */
#endif                                                        /* No brackets allowed here for "GiveUp" !!! */
        }
      fclose(fft_sem);

      printf(" found FFT semaphore file !\n");
      unlink( FFT_SEM_NAME );                                 /* delete FFT semaphore file */

      read_FFT( FFT_FILENAME );                               /* read FFT file */

      p_predict();                                            /* predict the frames */

      reset_timer(&timer1);
      start_timer(&timer1);

      p_write_scores( SCORES_FILENAME );                      /* creates and writes the scores file */

      time1 = stop_timer(&timer1);

      unlink( FFT_FILENAME );                                 /* delete FFT file */

      fd_scores_sem = creat( SCORES_SEM_NAME, PMODE);         /* create semaphore file for the scores -rw-rw-rw */
      close(fd_scores_sem);

      printf("  wrote the scores files (in %6.3f sec.)\n\n", time1);
    }

}
/*-------------------------------------------------------------------------------------------------------------------------------
```

MatVec_be.h

```
/*###########################################################################################################################
 #
 # MatVec_be.h          parallel implementation of matrix and vector operations for the MasPar
 #                      created 1991 by Tilo Sloboda  (sloboda@ira.uka.de)
 #
 # Implementation Note :
 #
 #   The matrices are stored transposed,
 #   which means that for A[i][j] index i varies faster than index j.
 #   (different from C's standard way to store matrices)
 #
 #-- RCS Info ---------------------------------------------------------------------------------------------------------------
 #
 # $RCSfile: MatVec_be.h,v $     $Revision: 1.1 $        $State: Exp $
 # $Date: 1991/11/29 11:26:11 $    $Author: sloboda $      $Locker:  $
 #
 # $Log: MatVec_be.h,v $
 # Revision 1.1  1991/11/29  11:26:11  sloboda
 # Initial revision
 #
 #
 ###########################################################################################################################*/

#ifndef _MATVEC_BE_H_
#define _MATVEC_BE_H_

/*== Includes ============================================================================================================*/

#include "ts_std.h"

/*== Public Constants ====================================================================================================*/

/*== Public Types ========================================================================================================*/

/*== Public Variables ====================================================================================================*/

/*###########################################################################################################################
 #
 #      PUBLIC DEBUGGING ROUTINES
 #
 ###########################################################################################################################*/

extern void          printVector();

extern void          printMatrix();

/*###########################################################################################################################
 #
 #      PUBLIC ROUTINES
 #
 ###########################################################################################################################*/

extern plural void   p_Add_Vectors();

extern plural void   p_Sub_Vectors();

extern plural void   p_Multiply_Vector_by_full_Matrix();

extern plural void   p_Multiply_Vector_by_sparse_Matrix();

extern plural void   p_sigmoid();

extern plural float  p_distance();

extern plural float  p_get_MatrixElement();

extern plural void   p_put_MatrixElement();

#endif
```

MatVec_be.m

```
/*****************************************************************************************************************
 #
 # MatVec_be.m            parallel implementation of matrix and vector operations for the MasPar
 #                        created 1991 by Tilo Sloboda  (sloboda@ira.uka.de)
 #
 # Description :
 #
 # Implementation Note :
 #
 #   The matrices are stored transposed,
 #   which means that for A[i][j] index i varies faster than index j.
 #   (different from C's standard way to store matrices)
 #
 # Last change :
 #   24.Oct  91  02:45  tilo    created - from lpnn_be.m revision 1.6
 #
 #-- RCS Info ------------------------------------------------------------------------------------------------------
 #
 # $RCSfile: MatVec_be.m,v $     $Revision: 1.1 $        $State: Exp $
 # $Date: 1991/11/29 11:26:37 $    $Author: sloboda $      $Locker:  $
 #
 # $Log: MatVec_be.m,v $
 # Revision 1.1  1991/11/29  11:26:37  sloboda
 # Initial revision
 #
 *****************************************************************************************************************/

/*== Includes ================================================================================================*/

#include <mpl.h>
#include <math.h>
#include <stdio.h>              /* for the debugging routines */

#include "ts_std.h"             /* some things they left out of C */
#include "MatVec_be.h"

/*== Private Constants =======================================================================================*/
/*== Private Types ===========================================================================================*/
/*== Private Variables =======================================================================================*/

static char     RCSid[] = "$header$";

static int      SizeOfFloat = sizeof(float),            /* = sizeof(float) only calculated at the beginning */
                SizeOfShort = sizeof(short);

static int      PE = 1;                                 /*  FOR DEBUGGING ONLY !!!
                                                            to watch the processing element with the Nr. PE */

/*== Public Variables ========================================================================================*/

/*****************************************************************************************************************
 #
 #      PUBLIC  DEBUGGING  ROUTINES
 #
 *****************************************************************************************************************/

/*----------------------------------------------------------------------------------------------------------------
 |
 | printVector                  PUBLIC  SINGULAR
 |
 | Parameters :
 |   Vector     pointer to the vector of floats
 |   max        length of the vector
 |   PE         the PE , the vector should be taken from
 |   str        a string, that is displayed
 |
 | Description :
 |   displays one plural vector of floats, which is stored on one PE
 |
 | History :
 |   16.July 91  Tilo     created
 |   24.Oct  91  Tilo     changed for the stand-alone module
 |
 +----------------------------------------------------------------------------------------------------------------*/

void printVector(Vector, max, PE, str)
    plural float * Vector;
           int     max, PE;
           char  * str;
{
   register int i;

  printf("\n%s\n",str);

  for (i=0; i<max; i++)
     printf(" %8.5f ", proc[PE].*Vector++);

  printf("\n");

}

/*----------------------------------------------------------------------------------------------------------------
 |
```

```
| printMatrix                    PUBLIC  SINGULAR
|
| Parameters :
|   Matrix     pointer to the matrix of floats
|   Max1       size of the matrix in Idx1-direction
|   Max2       size of the matrix in Idx2-direction
|   PE         the PE , the matrix should be taken from
|   str        a string, that is displayed
|
| Description :
|   displays one plural matrix of floats, which is stored on one PE
|   The matrices are stored transposed,
|   which means that for A[i][j] index i varies faster than index j.
|
| History :
|   16.July 91  Tilo     created
|   19.July 91  Tilo     changed it to handle transposed matrices.
|   24.Oct  91  Tilo     changed for the stand-alone module
|
+=============================================================================================================================*/

void printMatrix(Matrix, Max1, Max2, PE,  str)
    plural float * Matrix;
           int     Max1, Max2, PE;
           char  * str;
{
    register int   i,j;
    plural   float  f;

  printf("\ntransposed matrix in PE[%4d] : \t%s\n", PE, str);

  for (i=0; i<Max1; i++)
    {
      for(j=0; j<Max2; j++)
       { f = p_get_MatrixElement(Matrix,Max1, i,j);
         printf("%8.5f ", proc[PE].f);
       }
      printf("\n");
    }

  printf("\n");

}

/*=============================================================================================================================
|
| p_get_MatrixElement                  PUBLIC  PLURAL
|
| Parameters :
|   Matrix     pointer to the matrix of floats
|   Max1       size of the matrix in Idx1-direction
|   Idx1       first index in the matrix
|   Idx2       second index in the matrix
|
| Returns : the PLURAL values of the elements with that indices in the Matrix.
|           which is :  value = Matrix[Idx1][Idx2]
|
| Description :
|   returns the plural values that are stored in the matrix at Idx1 , Idx2.
|   The matrices are stored transposed,
|   which means that for A[i][j] index i varies faster than index j.
|
| History :
|   16.July 91  Tilo     created
|   19.July 91  Tilo     changed it to handle transposed matrices.
|   24.Oct  91  Tilo     changed for the stand-alone module
|
+=============================================================================================================================*/

plural float

p_get_MatrixElement(Matrix, Max1, Idx1, Idx2)

    plural float  * Matrix;
           int      Max1,
                    Idx1, Idx2;
{

  return( *( Matrix + (Idx1 + Idx2 * Max1) ) );

}
/*=============================================================================================================================
|
| p_put_MatrixElement                  PUBLIC  PLURAL
|
| Parameters :
|   Matrix     pointer to the matrix of floats
|   Max1       size of the matrix in Idx1-direction
|   Idx1       first index in the matrix
|   Idx2       second index in the matrix
|   value      the new value                    (plural float) !!!
|
| Description :
|   stores the plural values into the Matrix at Idx1 , Idx2.
```

```
  |   which is :  Matrix[Idx1][Idx2] = value
  |   The matrices are stored transposed,
  |   which means that for A[i][j] index i varies faster than index j.
  |
  | History :
  |   16.July 91  Tilo     created
  |   19.July 91  Tilo     changed it to handle transposed matrices.
  |   24.Oct  91  Tilo     changed for the stand-alone module
  |
  +-------------------------------------------------------------------------------------------------------------------------------*/

plural void

p_put_MatrixElement(Matrix, Max1, Idx1, Idx2, value)

    plural float  * Matrix;
          int     Max1,
                  Idx1, Idx2;
    plural float  value;            /* the new value */
{
  *( Matrix + (Idx1 + Idx2 * Max1) ) = value;

}

/*-------------------------------------------------------------------------------------------------------------------------------
  |
  | p_Add_Vectors                         PUBLIC  PLURAL
  |
  | Description :
  |
  |        OutVector := InVector + ThetaVector
  |          dim n       dim n        dim n
  |
  | Parameters :
  |   IN :
  |      plural    InVector      n dimensional vector of floats
  |      plural    ThetaVector   n dimensional vector of floats
  |      singular  dim           n
  |   OUT:
  |      plural    OutVector     n dimensional vector of floats
  |
  | History :
  |   29.Juni.91  Tilo     created
  |    5.Juli 91  Tilo     modified for malloc'ed matrices and vectors
  |
  +-------------------------------------------------------------------------------------------------------------------------------*/

plural void

p_Add_Vectors(InVector, ThetaVector, dim,  /* IN  */
              OutVector)                   /* OUT */

  plural float  *InVector, *ThetaVector, *OutVector;   /* singular pointers to plural Data */
        int    dim;
{
  register int i;

  for (i=0; i<dim; i++)
    *OutVector++ = *InVector++ + *ThetaVector++;      /*  OutVector[i] =  InVector[i] + ThetaVector[i]; */
}
/*-------------------------------------------------------------------------------------------------------------------------------
  |
  | p_Sub_Vectors                         PUBLIC  PLURAL
  |
  | Description :
  |
  |        OutVector := InVector - ThetaVector
  |          dim n       dim n        dim n
  |
  | Parameters :
  |   IN :
  |      plural    InVector      n dimensional vector of floats
  |      plural    ThetaVector   n dimensional vector of floats
  |      singular  dim           n
  |   OUT:
  |      plural    OutVector     n dimensional vector of floats
  |
  | History :
  |   26.Juli 91  Tilo     createdfor malloc'ed matrices and vectors
  |
  +-------------------------------------------------------------------------------------------------------------------------------*/

plural void

p_Sub_Vectors(InVector, ThetaVector, dim,  /* IN  */
              OutVector)                   /* OUT */

  plural float  *InVector, *ThetaVector, *OutVector;   /* singular pointers to plural Data */
        int    dim;
{
  register int i;

  for (i=0; i<dim; i++)
    *OutVector++ = *InVector++ - *ThetaVector++;      /*  OutVector[i] =  InVector[i] - ThetaVector[i]; */
}
```

```
/*=============================================================================================================================
  |
  | p_Multiply_Vector_by_full_Matrix      PUBLIC  PLURAL
  |
  | Description :
  |
  |       OutVector := InVector *  Matrix
  |         dimN         dimM     dimM*dimN
  |
  | Parameters :
  |   IN :
  |       InVector      transposed m dimensional vector of floats
  |       Matrix        dimM by dimN Matrix of floats
  |       dimM
  |       dimN
  |   OUT:
  |       OutVector     transposed n dimensional vector of floats
  |
  | Note :
  |   For both matrix and vectors the dimensions m,n are assumed to point into the datastructures ! Crashing otherwise.
  |
  | Implementation note :
  |   All the index calculations that are listed in the comments on the right hand side, are performed by all the
  |   incrementing and assigning of pointers you see on the left hand side (believe Me :-) )
  |
  |   The matrices are now stored "transposed" in memory, this was necessary. And it makes it a little bit faster.
  |
  | History :
  |   27.Juni 91  Tilo     created
  |    5.Juli 91  Tilo     modified for malloc'ed matrices and vectors
  |   19.July 91  Tilo     changed it to handle transposed matrices.
  |
  +============================================================================================================================*/

plural void

p_Multiply_Vector_by_full_Matrix(InVector, Matrix, dimM, dimN,       /* IN  */
                                 OutVector)                          /* OUT */

  plural float  *InVector, *OutVector, *Matrix; /* singular pointers to plural floats */
        int     dimM, dimN;                     /* singular */
{
  register        int     i,j;
  register plural float   tempSum;/* plural temporal storage for faster summation */
  register plural float * MPtr,   /* singular pointer to plural floats , points to actual position in the matrix        */
                        * InPtr,  /* singular pointer to plural floats , points to actual position in the input vector  */
                        * OutPtr; /* singular pointer to plural floats , points to actual position in the output vector */

  MPtr = Matrix;
  OutPtr = OutVector;

  for (j=0; j<dimN; j++)                          /* for (j=0; j<n; j++)                                          */
    {                                             /*  {                                                           */
      InPtr  = InVector;
      tempSum = 0.0;                              /*    OutVector[j]=0;                                           */

      for (i=0; i<dimM; i++)                      /*    for (i=0; i<m; i++)                                       */
         tempSum += *InPtr++ * *MPtr++;           /*       OutVector[j]  +=  InVector[i] * Matrix[i][j];          */

      *OutPtr++ = tempSum;
    }                                             /*  }                                                           */
}

/*=============================================================================================================================
  |
  | p_Multiply_Vector_by_sparse_Matrix    PUBLIC  PLURAL
  |
  | Description :
  |
  |       OutVector := InVector *  Matrix
  |         dimN         dimM     dimM*dimN
  |
  | Parameters :
  |   IN :
  |       InVector      transposed m dimensional vector of floats
  |       Matrix        dimM by dimN Matrix of floats
  |       dimM
  |       dimN
  |   OUT:
  |       OutVector     transposed n dimensional vector of floats
  |
  | Note :
  |   For both matrix and vectors the dimensions m,n are assumed to point into the datastructures ! Crashing otherwise.
  |
  | Implementation note :
  |   All the index calculations that are listed in the comments on the right hand side, are performed by all the
  |   incrementing and assigning of pointers you see on the left hand side (believe Me :-) )
  |
  |   The matrices are now stored "transposed" in memory, this was necessary. And it makes it a little bit faster.
  |
  | History :
  |   27.Juni 91  Tilo     created
  |    5.Juli 91  Tilo     modified for malloc'ed matrices and vectors
  |   12.Juli 91  Tilo     inserted the case distinction for 0.0
```

```
|   19.July 91  Tilo     changed it to handle transposed matrices.
|
+-------------------------------------------------------------------------------------------------------------------------------

plural void

p_Multiply_Vector_by_sparse_Matrix (InVector, Matrix, dimM, dimN,       /* IN  */
                                    OutVector)                          /* OUT */

  plural float  *InVector, *OutVector, *Matrix; /* singular pointers to plural floats */
        int     dimM, dimN;                     /* singular */
{
  register        int     i,j;
  register plural float   tempSum,/* plural temporal storage for faster summation */
                          tempMat;
  register plural float * MPtr,   /* singular pointer to plural floats , points to actual position in the matrix        */
                        * InPtr,  /* singular pointer to plural floats , points to actual position in the input vector  */
                        * OutPtr; /* singular pointer to plural floats , points to actual position in the output vector */

  MPtr = Matrix;
  OutPtr = OutVector;

  for (j=0; j<dimN; j++)                        /* for (j=0; j<n; j++)                                  */
    {
      InPtr  = InVector;
      tempSum = 0.0;                            /*  { OutVector[j]=0;                                   */

      for (i=0; i<dimM; i++)                    /*    for (i=0; i<m; i++)                               */
        {
          tempMat = *MPtr++;
          if (tempMat != 0.0) then              /* 0.0 is very common in sparse connected matrices ... */
            {
             /* printf("i= %2d, j= %2d,  *InPtr= %8.5f, *MPtr= %8.5f, tempSum= %8.5f, ",
              *         i,j,proc[PE].*InPtr, proc[PE].tempMat, proc[PE].tempSum);
              */

              tempSum += *InPtr * tempMat;      /*     OutVector[j]  +=  InVector[i] * Matrix[i][j];   */

             /* printf("\t==> tempSum= %8.5f\n", proc[PE].tempSum);
              */
            }
          InPtr++;
        }
      *OutPtr++ = tempSum;
    }                                           /*  }                                                   */
}

/*=============================================================================================================================
|
| p_sigmoid                             PUBLIC  PLURAL
|
| Description :
|                                                    1
|   calculates the sigmoid function.  sig(x) =  ----------
|                                                      -x
|                                              1 + e
| Parameters :
|   IN :     dim          dimension of the vector
|
|   IN/OUT : Vector       transposed m dimensional vector of floats
|
| Note :
|   For the vector the dimensions is assumed to point into the datastructure ! Crashing otherwise.
|
| History :
|   14.Juli 91  Tilo     created
|
+-------------------------------------------------------------------------------------------------------------------------------

plural void

p_sigmoid (Vector, dim)
  plural float  * Vector;       /* singular pointers to plural floats */
        int     dim;            /* singular */
{
  register        int     i;
  register plural float   x;    /* plural temporal storage */
  register plural float * v;

  v = Vector;
  for(i=0; i<dim; i++)
    {
      x = *v;

      if (x < -10.0)
         then *v = 0.0;
         else if (x > 10.0)
              then *v = 1.0;
              else *v = (1.0 / (1.0 + fp_exp (- x)));
      v++;
    }

}

/*=============================================================================================================================
|
```

```
| p_distance                          PUBLIC  PLURAL
|
| Description :
|
|   calculates the euclidian distance between two vectors.
|
| Parameters :
|   IN :     dim          dimension of the vectors
|            Vector1      transposed m dimensional vector of floats
|            Vector2      transposed m dimensional vector of Floats
|
| Returns :  scaled, euclidian distance between two vectors.
|
| Note :
|   For the vector the dimensions is assumed to point into the datastructure ! Crashing otherwise.
|
| History :
|   16.Juli 91  Tilo     created
|
+=============================================================================================================================

plural float

p_distance(Vector1, Vector2, dim)
      plural float  * Vector1, * Vector2;
      int             dim;
{
      register int    i;
      register
      plural float    delta, sum,
                      * v1,  * v2;      /* use the pointers in registers to speed the whole thing up */
  v1 = Vector1;
  v2 = Vector2;
  sum = 0.0;
  for (i=0; i<dim; i++)
    {
      delta = *v1++ - *v2++;            /* soll - ist ...  */
      sum += delta * delta;             /* sum the squares */
    }

  return(0.4 * fp_sqrt(sum));           /* 0.4 is an arbitrary scaling factor for the DTW (introduced by otto) */
}

/*-----------------------------------------------------------------------------------------------------------------------------
```

D.4 Suche der N besten Satzhypothesen

definitions.txt

```
/*****************************************************************************************************************************
;
; bawl.h:  Header file for the BAWL speech recognition system.  BAWL = Backpropagate at Word Level.
;    This system uses either a TDNN classifier or predictive networks to model the low-level acoustics of speech.
;    In either case, error is backpropagated at the word level, rather than at the phoneme level.
;    The current implementation assumes we are using the Conference Registration database.
;
;
; Conventions in the BAWL system:
;
;   1. CONSTANTS and TYPES are entirely capitalized.  Variables are in lower case, with capital letters for readability.
;
;   2. Variables are initialized when they are declared, whenever possible.  Example: "INT frameX = frameXI".
;
;   3. Most indices are 0-relative.  The only exceptions are (batchX, dialogX, sentX): these are 1-relative
; in order to be consistent with human-readable files which describe the conversations.
; Data structures which use these indices must allocate one extra element, and waste the 0th element.
;
;   4. Prefixes:
; p = phoneme.  Example: pStateX = phoneme-relative state index.
; w = word.  Example: wStateX = word-relative state index.
; s = sentence.  Example: sFrameX = sentence-relative frame index.
; b = batch.  Example: bError = total error over the entire batch of sentences.
; i/j = prev/next Example: isFrameX, jsFrameX = prev/next sentence-relative frame indices, used in DP routines.
;
;   5. Suffixes:
; X = index.  Example: frameX = frame index.
; I = initial.  Example: frameXI = initial frame index.
; F = final.  Example: frameXF = final frame index.  Looping: "for (frameX=frameXI; frameX<=frameXF; frameX++)..."
; E = end = F+1.  Example: frameXE = end frame index = frameXF+1.
; N = number of.  Example: frameN = number of frames.  (Typically frameXE==frameN, but perhaps frameXE<frameN.)
; Z = maximum.  Example: frameXZ = runtime maximum frame index, >= current value of frameN-1.
; P = pointer.  Example: modelP = pointer to a model.
; A = array.  Example: phonemeA = array of phonemes.  (May be either static or dynamically allocated.)
; M = matrix.  Example: traceM = trace matrix.
; C = char.  Example: allocCA = allocate an array of characters (or bytes).
; S = string.  Example: allocS  = allocate a string.
; T = type.  Example: allocTA = allocate an array whose elements are of a given type.
; R = relative.  Example: inFrameR = relative offset of an input frame (relative to the current frame).
;
;   6. Concatenated prefixes and suffixes are read from the outside-in.  Examples:
; wsPhonX = w(sPhonX) = word's (sentence-relative phoneme index).
; sFrameXF = (sFrameX)F = final (sentence-relative frame index).
; thingPI = (thingP)I = initial (pointer to a thing),  i.e., head of a linked list of things.
; thingIP = (thingI)P = pointer to (initial thing),    i.e., sequential data structure is not specified.
; thingPA = (thingP)A = array of (pointers to things), i.e., *(thingPA[2]) = thing.
; thingAP is bad style; equivalent to thingA, i.e., the address of an array of things.  Use thingA [2], not thingAP [2].
;
;   7. Semantic definitions:
; "rewrites" = rewrite rules which are applied to raw phoneme labels (in the labelfile and the dictionary file)
; to transform them internally into another set of phoneme labels which we are prepared to model.
; Example: if the labelfile uses "KCL K", but we don't want to model K-closures, we can apply "KCL K => K".
; "phoneme" means canonical arpabet phoneme.  Typically there are 40 phonemes, and phonemeX = (0..39).
; "phone" refers to an instance of a phoneme within a word, BEFORE rewrite rules are applied to word pronunciations.
; "phon" refers to an instance of a phoneme within a word, AFTER rewrite rules are applied.  (Much more pervasive.)
; Example:  If a word has 5 phonemes, then phonN=5, and for phonX=(0..4) => phonA[phonX] = phonemeX.
; "model" = phoneme model = structure of states, transitions, and neural network(s) which implement a phoneme.
; "state" = HMM state.  (stateP = pointer to a state.  pStateX/wStateX = phoneme/word- relative state index.)
; "dict" = dictionary:  dictX uniquely identifies a word's lexical spelling, but not its pronunciation.
; Each dictionary entry, dictA[dictX], has a linked list of variant pronunciations, i.e., "variantP"s.
; All known variant pronunciations are listed in the dictionary file, which is loaded at runtime.
; "vocab" = vocabulary, containing all known variant pronunciations for a desired subset of dictionary entries.
; This subset is indicated at runtime by a vocabulary file.
; Testing evaluates each variantP = vocabA[vocabX], for each vocabX = (0..vocabN-1).
; "sample" = speech sample, i.e. spectrogram for one continuous speech utterance.
;
;   8. Documentation:
; Every file begins with a header (inside *********'s), containing:
; Name of file, and a brief description.
; History of revision (very coarse level).
; Every routine begins with a header (inside =========='s), containing:
; Name of routine, and a brief description.
; Parameters -- name and description of each.  Separate IN-parameters from OUT-parameters, if appropriate.
; Return value, if there is one.
; History of revision.
; Optional information:  longer description, algorithm, conventions, assumptions, global variables....
; Major comments in the code appear on their own lines (inside ----------'s), indented with the code.
; Minor comments appear to the right of each line, whenever they may be useful.
; If a procedure takes both in-parameters (listed first) and out-parameters (listed last), the procedure call should
; show the boundary between them by using white space (i.e., a few extra spaces, or a newline).
;
; History:
;   June 11 1991   jmt   Created, based on Joe Tebelskis's "lpnn.h" and Patrick Haffner's "types.h".
;*****************************************************************************************************************************/
```

dp.h

```
/*****************************************************************************************************************************
 *
 *  dp.h            1991 by  Tilo Sloboda,   (sloboda@ira.uka.de)
 *
 *****************************************************************************************************************************
 *
 * Description :
 *   This is the header file for the parallel DTW implementation,
 *   with the constants and types that are both used in front end and back end code.
 *
 * Note :
 *
 *
 * Implementation Note :
 *
 *
 * Last change :
 *   28.Nov  91  13:30  Tilo    created
 *    2.Dec  91   1:09  Tilo    moved all common constants, types and SizeOf... variables to this file
 *
 *-- RCS Info ------------------------------------------------------------------------------------------------------------------
 *
 * $RCSfile$     $Revision$        $State$
 * $Date$    $Author$      $Locker$
 *
 * $Log$
 *
 *****************************************************************************************************************************/

#ifndef _DP_H_
#define _DP_H_

/*== Includes ================================================================================================================*/

/*== Public Constants ========================================================================================================*/

#define SILENCE                   0         /* vocabulary index of the word "SILENCE" */

#define MAX_N                     4

#define BIG_NUMBER             9999.0       /* a score for DTW paths that are unlikely  */
#define INFINITY             999999.0       /* a score, which no DTW path should exceed */

#define MAX_Models                8         /* usually just   2 models  , actual number will be malloc'ed on the ACU   */
#define MAX_Phonemes             50         /* usually about 50 phonemes, actual number will be malloc'ed on the ACU   */

#define MAX_PrecStates            2         /* max. number of final states per model */
#define MAX_StatesPerModel        6         /* max. number of states per model      */

#define MAX_TransPerState         3         /* max. number of outgoing transitions per model-state  <= MAX_StatesPerModel */

#define MAX_NetsPerModel          3         /* max. number of neural nets per model */
#define MAX_CharsPerModel        10         /* length of a model name               */
#define MAX_CharsPerPhoneme       2         /* length of a phonemes name            */

#define MAX_PEsX                128
#define MAX_PEsY                128
#define MAX_PEs               16384

/*-- some file io stuff : --------*/

#define PMODE         0666                    /* everybody can read and write */

#define READ_ONLY      0                      /* UNIX basic file i/o modi */
#define WRITE_ONLY     1
#define READ_WRITE     2

/*-- some filenames : ------------*/

#define MODEL_FILENAME   "model"
#define NET_FILENAME     "network"
#define WEIGHTS_FILENAME "weights"

#define SCORES_FILENAME  "data.scores"
#define SCORES_SEM_NAME  "data.scores-ready"
#define HYPO_FILENAME    "data.hypo"
#define HYPO_SEM_NAME    "data.hypo-ready"

/*== Public Types ============================================================================================================*/

/* basic types : */

typedef float             ScoreT;

typedef unsigned short    LIdxT;
typedef unsigned char     IdxT;                          /* a state number or a NN index.
                                                            pos. indices are state numbers 0..N in the same phoneme,
                                                            neg. indices are state numbers 0..N in previous phonemes.
                                                            is MAX_StatesPerModel for invalid transitions.
                                                          */
/* to be substituted in all *.c, *.m files : */

typedef LIdxT             WordRefT;                      /* reference of words    by numbers */
```

```
typedef IdxT              VariantRefT;                    /* reference of variants by numbers */
typedef IdxT              PhonemeRefT;                    /* reference of phonemes by numbers */
typedef IdxT              ModelRefT;                      /* reference of models   by numbers */
typedef char              StateRefT;                      /* states referenced by positive and negative numbers */

/* aggregate types : */

typedef struct WordTransS
  { ScoreT   penalty;
    LIdxT    word;
  }                       WordTransT;

typedef struct TransS
  { StateRefT  toState;
    ScoreT     penalty;
  }                       TransT;

typedef TransT            TransDescrT [MAX_TransPerState];    /* a list (array) of all transitions to previous states */
typedef TransDescrT       TransTableT [MAX_StatesPerModel];

typedef IdxT              NetTableT   [MAX_StatesPerModel];   /* describes the implementation of the states
                                                                 (mapping NNs to states)
                                                                */

typedef struct ModelS
  { char          Name [MAX_CharsPerModel];                   /* name of this model */
    card          NrOfStates,                                 /* assumed : state 0 is initial, state NrOfStates-1 is final */
                  NrOfNets;
    TransTableT   Trans;                                      /* should be initialized by state Nr. -1     */
    NetTableT     Net;                                        /* which state is implemented by which NN .. */
  }                       ModelT;

typedef struct PhonemeS
  { char          Spelling [MAX_CharsPerPhoneme];             /* spelling of the phoneme, ie: "SH"     */
    char          Symbol;                                     /* 1 char symbolic name for the phoneme  */
    ModelRefT     Model;                                      /* pointer to the model for this phoneme */
  }                       PhonemeT;

/*##########################################################################################################################
 #
 #      PUBLIC VARIABLES
 #
 ##########################################################################################################################*/

/*--------------------------------------------------------------------------------------------------------------------------
 | the fe_PrecWords matrix still must be transformed into an array of predecessor lists (not yet implemented)
 +-------------------------------------------------------------------------------------------------------------------------*/

#ifdef _MPL /*-- back end declarations : ------------------------------------------------------------------------------------*/

visible extern    int           nproc, nxproc, nyproc;          /* MasPar machine limits, as defined in mpl.h */

visible extern    ModelT        fe_ModelA     [MAX_Models];     /* stores all models (for read_models() ) */
                                                                /* modelN says how many models we have */
visible extern    IdxT          fe_NetsPerPhonemeA[MAX_Phonemes];/* for distribution of the scores to the PEs */

#else /*-------- front end declarations : -----------------------------------------------------------------------------------*/

extern          int             nproc, nyproc, nxproc;          /* MasPar machine limits, as defined in mpl.h */

#endif

/*##########################################################################################################################
 #
 #      PUBLIC ROUTINES
 #
 ##########################################################################################################################*/

#ifdef _MPL /*-- back end declarations : ------------------------------------------------------------------------------------*/

visible extern void           init_DTW();
visible extern void           dp_sentence();

#else /*-------- front end declarations : -----------------------------------------------------------------------------------*/

extern void                   init_DTW();
extern void                   dp_sentence();

#endif

/*--------------------------------------------------------------------------------------------------------------------------*/

#endif _DP_H_
```

dp_fe.h

```
/*****************************************************************************************************************************
 *
 *  dp_fe.h             1991 by  Tilo Sloboda,   (sloboda@ira.uka.de)
 *
 *****************************************************************************************************************************
 *
 * Description :
 *   This is the front end part of the DTW source.
 *
 *
 * Note :
 *   see ...
 *
 * Implementation Note :
 *
 *
 *
 * Last change :
 *   28.Oct  91  13:30  tilo    created
 *
 *
 *-- RCS Info ------------------------------------------------------------------------------------------------------------------
 *
 * $RCSfile$     $Revision$        $State$
 * $Date$    $Author$      $Locker$
 *
 * $Log$
 *
 *****************************************************************************************************************************/

#ifndef _DP_FE_H_
#define _DP_FE_H_

/*== Includes ================================================================================================================*/

/*== Public Constants ========================================================================================================*/

/*== Public Types ============================================================================================================*/

/*== Public Variables ========================================================================================================*/

/*****************************************************************************************************************************
 *
 *      PUBLIC VARIABLES
 *
 *****************************************************************************************************************************/

/*----------------------------------------------------------------------------------------------------------------------------
 | front end variable for the ACU :
 +---------------------------------------------------------------------------------------------------------------------------*/

ModelT       fe_ModelA    [MAX_Models];          /* stores all models (for read_models() ) */
                                                 /* modelN says how many models we have */
IdxT         fe_NetsPerPhonemeA[MAX_Phonemes];   /* for distribution of the scores to the PEs */

/*----------------------------------------------------------------------------------------------------------------------------
 | the following variables are ALL pointers to malloc'ed arrays, with nproc elements ,
 | those named fe_... are copied to the PEs with blockIn :
 +---------------------------------------------------------------------------------------------------------------------------*/

boolean     *fe_firstPhonemeBP,
            *fe_lastPhonemeBP;
ModelRefT   *fe_modelXBP;
WordRefT    *fe_dictXBP;
VariantRefT *fe_variantXBP;
PhonemeRefT *fe_phonemeXBP;

/*----------------------------------------------------------------------------------------------------------------------------
 | the fe_PrecWords matrix still must be transformed into an array of predecessor lists (not yet implemented)
 +---------------------------------------------------------------------------------------------------------------------------*/

/*****************************************************************************************************************************
 *
 *      PUBLIC DEBUGGING ROUTINES
 *
 *****************************************************************************************************************************/

/* extern ... */

/*****************************************************************************************************************************
 *
 *      PUBLIC ROUTINES
 *
 *****************************************************************************************************************************/

/* extern ...           from other fe-modules */

/* visible extern ...   from be-modules  */

/*----------------------------------------------------------------------------------------------------------------------------
 | memo
 +---------------------------------------------------------------------------------------------------------------------------*/
```

dp_fe.c

```
/*****************************************************************************************************************************
 *
 *  dp_fe.c             1991 by  Tilo Sloboda,   (sloboda@ira.uka.de)
 *
 *****************************************************************************************************************************
 *
 * Description :
 *   This is the front end part of the DTW source.
 *
 *
 * Note :
 *   see ...
 *
 * Implementation Note :
 *
 *
 *
 * Last change :
 *   28.Oct  91  13:30  Tilo    created
 *   18.Nov  91  21:01  Tilo
 *   29.Nov  91  14:00  Tilo    integrated BAWL style file io
 *
 *-- RCS Info ------------------------------------------------------------------------------------------------------------------
 *
 * $RCSfile$     $Revision$        $State$
 * $Date$    $Author$      $Locker$
 *
 * $Log$
 *
 *****************************************************************************************************************************/

/*== Includes ================================================================================================================*/

#include <stdio.h>
#include <string.h>

#include "ts_std.h"                         /* things, they left out of the C language                          */
#include "dp.h"                             /* common constants and types for the parallel DTW implementation */
#include "dp_fe.h"                          /* front end specific stuff                                         */
#include "BAWLfiles_fe.h"                   /* read the basic files in BAWL style                               */

/*============================================================================================================================*
 | for   Public Variables, Types, Constants   see  :   dp_fe.h
 +============================================================================================================================*/

/*== Private Constants =======================================================================================================*/

/*== Private Types ===========================================================================================================*/

/*== Private Variables =======================================================================================================*/

static        char          RCSid[] = "$header$";

static        card          ChunkSize;                        /* max. size of scores, that can be handled in DTW */

static        card          NrOfPEs,
                            NrOfPEsX,
                            NrOfPEsY;

/*----------------------------------------------------------------------------------------------------------------------------*
 | the following variable is a pointer to a malloc'ed array, with nproc elements :
 +----------------------------------------------------------------------------------------------------------------------------*/

static        boolean       *PE_usedBP;             /* this one is used internaly in the front end */

/*----------------------------------------------------------------------------------------------------------------------------*
 | the fe_PrecWords matrix still must be transformed into an array of predecessor lists (not yet implemented)
 +----------------------------------------------------------------------------------------------------------------------------*/

static        ScoreT        *fe_PrecWordsBP;        /* 2 dim matrix for inverting the bigrams,     ((dictN * dictN) scores)   */
static        card          *NrPrecWordsBP,         /* array of predecessors per dictionary entry, (         dictN   cards)   *
                            MaxPrecWords;           /* max number of preceeding words per word */


/*****************************************************************************************************************************
 *
 *      PRIVATE MACROS
 *
 *****************************************************************************************************************************/

/*  access to malloc'ed, two dimensional arrays,
 *  indices used different than as in C (here : x first) :
 */

#define put( M, x,y, maxX, value ) /* M is an array ; x,y indices in that array ; value is the new value for this element */ \
                                   \
   *(M + x + y*maxX) = value       \

#define get( M, x,y, maxX )    /* M is an array ; x,y indices in that array ; returns the value, stored in this element */ \
                               \
   *(M + x + y*maxX)           \
```

```
/*##########################################################################################################################
 #
 #      PRIVATE DEBUGGING ROUTINES
 #
 ##########################################################################################################################*/

/*==========================================================================================================================
 |
 | print_models           PRIVATE
 |
 | Description :
 |    prints the internal representation of the models
 |    for printing the BAWL representation, use "print_BAWL_models()"
 |
 | Parameter : fe_Models[], NrModelsUsed
 |
 | History :
 |   29.Nov  91  Tilo     created
 |
 +=========================================================================================================================*/

void print_models(NrModelsUsed, fe_ModelA)

    int      NrModelsUsed;
    ModelT   fe_ModelA[];
{
    int      m, s, t;
    ModelT  *Mptr;
    TransT  *Tptr;

 printf("\n models used : %d\n\n", NrModelsUsed);

 for (m=0; m < NrModelsUsed; m++)
  {
    Mptr = &(fe_ModelA[m]);

    printf("\n model %d ; name : \"%s\" ;  states : %d \n\n", m, Mptr->Name, Mptr->NrOfStates);
    printf(" from   to     penalty\tnetNr\n");

    for (s=0; s < Mptr->NrOfStates; s++)
     {
       for (t=0; t < MAX_TransPerState; t++)
        { Tptr = &(Mptr->Trans[s][t]);

          printf("  %2d    %2d     %8.2f\n" , s, Tptr->toState, Tptr->penalty);
        }
       printf("\t\t\t %2d\n" , Mptr->Net[s]);
     }
    printf("\n");
  }
 printf("\n");
}

/*==========================================================================================================================
 |
 | print_bool_array           PRIVATE
 |
 | Description :
 |
 | Parameter :
 |
 | History :
 |    2.Dec  91  Tilo     created
 |
 +=========================================================================================================================*/

void print_bool_array(A, maxX, maxY)
   boolean *A;
   int      maxX, maxY;
{
   int      x,y;

 printf("\n");
 for(y=0; y<maxY; y++)
  {
    for(x=0; x<maxX; x++)
     ( get(A, x,y, NrOfPEsX) ? printf("T ") :  printf("F "));

   printf("\n");
  }
 printf("\n");
}

/*==========================================================================================================================
 |
 | show_phonemes           PRIVATE
 |
 | Description :
 |
 | Parameter :
 |
 | History :
 |    2.Dec  91  Tilo     created
 |    9.Jan  91  Tilo     changed - now the phonemes of a word are connected by "-" signs.
 +=========================================================================================================================*/
```

```
void show_phonemes()
{
   int      x,y;
   boolean  first, last, used;

 printf("\n words, as distributed to the PEs :\n\n");
 printf(" the phonemes of the words are connected by \"-\" signs. Empty PEs are marked with \"~\" signs.\n\n");

 for(y=0; y<NrOfPEsY; y++)
  {
    for(x=0; x<NrOfPEsX; x++)
     { used  = get(PE_usedBP, x,y, NrOfPEsX);
       first = get(fe_firstPhonemeBP, x,y, NrOfPEsX);
       last  = get(fe_lastPhonemeBP, x,y, NrOfPEsX);

       if (used)
          then {
                 printf("%c", phonemeA[ get(fe_phonemeXBP, x,y, NrOfPEsX) ].char1);
                 if   (first AND last)  then printf(" ");        /* a one phoneme word */
                 elif (first)           then printf("-");        /* a word beginning   */
                 elif (last)            then printf(" ");        /* a word end         */
                                        else printf("-");        /* inner phoneme      */
               }
          else printf("~ ");                                    /* empty PE           */
     }
   printf("\n");
  }
 printf("\n");
}
/*=============================================================================================================================
 |
 | show_words           PRIVATE
 |
 | Description :
 |
 | Parameter :
 |
 | History :
 |    2.Dec  91  Tilo     created
 |
 +============================================================================================================================*/

void show_words()
{
   int      x,y, idx;
   boolean  first, last, used;

 printf("\n words, as distributed to the PEs :\n\n");
 printf("  #  stands for a one phoneme word\n  [  stands for the first phoneme in a word\n");
 printf("  +  stands for a phoneme in the middle of a word\n  ]  stands for the last phoneme in a word\n");
 printf("  -  stands for an empty processing element\n\n");

 for(y=0; y<NrOfPEsY; y++)
  {
    for(x=0; x<NrOfPEsX; x++)
     { used  = get(PE_usedBP, x,y, NrOfPEsX);
       first = get(fe_firstPhonemeBP, x,y, NrOfPEsX);
       last  = get(fe_lastPhonemeBP, x,y, NrOfPEsX);

       if (used)
          then if   (first AND last)  then printf(" #");        /* a one phoneme word */
               elif (first)           then printf(" [");        /* a word beginning   */
               elif (last)            then printf(" ]");        /* a word end         */
                                      else printf(" +");        /* inner phoneme      */
          else printf(" -");                                    /* empty PE           */
     }
   printf("\n");
  }
 printf("\n");
}
/*=============================================================================================================================
 |
 | show_word_indices           PRIVATE
 |
 | Description :
 |
 | Parameter :
 |
 | History :
 |    2.Dec  91  Tilo     created
 |
 +============================================================================================================================*/

void show_word_indices()
{
   int      x,y, idx;

 printf("\n");
 for(y=0; y<NrOfPEsY; y++)
  {
    for(x=0; x<NrOfPEsX; x++)
     { idx = get(fe_dictXBP, x,y, NrOfPEsX);
       if (idx>999) then idx = -1;      /* ugly, but at the moment, we just have about 500 words */

       printf(" %3d", idx);
     }
```

```
    printf("\n");
  }
 printf("\n");
}

/*=============================================================================================================================
 |
 | print_bigrams            PRIVATE
 |
 | Description :
 |
 | Parameter :
 |
 | History :
 |    3.Dec  91  Tilo     created
 |
 +============================================================================================================================*/

void print_bigrams(dictN, NrPrecWordsBP)

     INT   dictN;
     int   NrPrecWordsBP[];
{
     int   i;

  printf("\ninverted word transitions   for the words 0 to %d : \n\n", dictN-1);
  for (i=0; i<dictN; i++)
    printf(" %3d %3d\n", i,  NrPrecWordsBP[i]);
}

/*############################################################################################################################
 #
 #      PRIVATE INITIALIZATION ROUTINES
 #
 ############################################################################################################################*/

/*=============================================================================================================================
 |
 | get_DPU_configuration           PRIVATE
 |
 | Description :
 |    finds out how many PE's the DPU has. ( The amount of PE memory isn't available yet.)
 |
 | Parameter : NrOfPEs , NrOfPEsX , NrOfPEsY
 |
 | History :
 |   29.Nov  91  Tilo     created
 |
 +============================================================================================================================*/

void get_DPU_configuration(NrOfPEs, NrOfPEsX, NrOfPEsY)

   card  *NrOfPEs, *NrOfPEsX, *NrOfPEsY;
{
   int    bytes = sizeof(int);     /* oooops, not very sensible !  This is the front end representation */

   copyIn( &nproc,  NrOfPEs,  bytes);
   copyIn( &nxproc, NrOfPEsX, bytes);
   copyIn( &nyproc, NrOfPEsY, bytes);

   printf("\n DPU limits :\n\n nproc  = %d\n nxproc = %d\n nyproc = %d\n\n",  *NrOfPEs, *NrOfPEsX, *NrOfPEsY);
}

/*=============================================================================================================================
 |
 | allocate                 PRIVATE
 |
 | Description :
 |    allocates memory for the fe_  variables
 |
 | Parameter :     NrOfPEs, NrOfPEsX, NrOfPEsY : the actual number of PEs
 |
 | Globals initialized :
 |
 | History :
 |    1.Dec  91  Tilo     created
 |
 +============================================================================================================================*/

void allocate(NrOfPEs, NrOfPEsX, NrOfPEsY, dictN)
     int      NrOfPEs, NrOfPEsX, NrOfPEsY, dictN;    /* number of processing elements, number of words in dictionary */
{
     int      x, y;
     boolean *usedP, *firstP, *lastP;
     ScoreT  *floatP;
     card    *cardP;

  printf("\n allocating fe_ variables ...");  fflush(stdout);

  PE_usedBP         = (boolean *)     malloc(NrOfPEs*sizeof(boolean));
  fe_firstPhonemeBP = (boolean *)     malloc(NrOfPEs*sizeof(boolean));
  fe_lastPhonemeBP  = (boolean *)     malloc(NrOfPEs*sizeof(boolean));
  fe_modelXBP       = (ModelRefT *)   malloc(NrOfPEs*sizeof(ModelRefT));
  fe_dictXBP        = (WordRefT *)    malloc(NrOfPEs*sizeof(WordRefT));
  fe_variantXBP     = (VariantRefT *) malloc(NrOfPEs*sizeof(VariantRefT));
  fe_phonemeXBP     = (PhonemeRefT *) malloc(NrOfPEs*sizeof(PhonemeRefT));
```

```
  fe_PrecWordsBP    = (ScoreT *)      malloc(dictN*dictN * sizeof(ScoreT));
  NrPrecWordsBP     = (card *)        malloc(dictN*sizeof(card));

  AssertAndExit((PE_usedBP         != NIL),"allocating PE_usedBP on the front end failed",        -1);
  AssertAndExit((fe_firstPhonemeBP != NIL),"allocating fe_firstPhonemeBP on the front end failed",-1);
  AssertAndExit((fe_lastPhonemeBP  != NIL),"allocating fe_lastPhonemeBP on the front end failed", -1);
  AssertAndExit((fe_modelXBP       != NIL),"allocating fe_modelXBP on the front end failed",      -1);
  AssertAndExit((fe_dictXBP        != NIL),"allocating fe_dictXBP on the front end failed",       -1);
  AssertAndExit((fe_variantXBP     != NIL),"allocating fe_variantXBP on the front end failed",    -1);
  AssertAndExit((fe_phonemeXBP     != NIL),"allocating fe_phonemeXBP on the front end failed",    -1);
  AssertAndExit((fe_PrecWordsBP    != NIL),"allocating fe_PrecWordsBP on the front end failed",   -1);
  AssertAndExit((NrPrecWordsBP     != NIL),"allocating NrPrecWordsBP on the front end failed",    -1);

  printf(" initializing ...");  fflush(stdout);

  /* raw initialize the arrays */

  usedP  = PE_usedBP;
  firstP = fe_firstPhonemeBP;
  lastP  = fe_lastPhonemeBP;

  for (y=0; y<NrOfPEsY; y++)
    for (x=0; x<NrOfPEsX; x++)
      {
        *(usedP++)  = FALSE;
        *(firstP++) = FALSE;
        *(lastP++)  = FALSE;

     /*  put(PE_usedBP,        x,y, NrOfPEsX, FALSE);
        put(fe_firstPhonemeBP,x,y, NrOfPEsX, FALSE);
        put(fe_lastPhonemeBP, x,y, NrOfPEsX, FALSE);
      */
        put(fe_modelXBP,      x,y, NrOfPEsX, -1);
        put(fe_dictXBP,       x,y, NrOfPEsX, -1);
        put(fe_variantXBP,    x,y, NrOfPEsX, -1);
        put(fe_phonemeXBP,    x,y, NrOfPEsX, -1);
      }
  printf("...");  fflush(stdout);

  /* initialize the preceeding word penalty table : */

  cardP = NrPrecWordsBP;
  for (x=0; x<dictN; x++)
    *(cardP)++ = 0;

  y = dictN*dictN;
  floatP = fe_PrecWordsBP;
  for (x=0; x<y; x++)
    *(floatP++) = INFINITY;

  printf(" done.\n\n");
}

/*+=============================================================================================================================
  |
  | map_words_to_PEs            PRIVATE
  |
  | Description :
  |    maps the words to the processing elements  (initializes the fe_ variables)
  |
  | Parameters : dictN, dictA, phonemeN, phonemeA, modelN, modelA
  |
  |
  | Globals initialized :     fe_ variables
  |
  | Note :    - the memory for the fe_ variables has to be allocated first.
  |           - in this version, the words are distributed phoneme by phoneme (not network by network).
  |           - phonemes are distributed to a PE-row with a small index, if they fit in there (to fill up empty spaces).
  |
  | History :
  |    2.Dec  91  Tilo      created
  |    4.Dec  91  Tilo      changed the heuristic a little, long words are now distributed first,
  |                         short words are distributed at the and. A word is only distributed to a row
  |                         of processing elements, if there isn't only one free PE remaining.
  +=============================================================================================================================*/
#define SmallWord  5 /* a word with that many phonemes and less is considered to be "small" (will be distributed at the end) */

int  map_words_to_PEs(dictN, dictA, phonemeN, phonemeA, modelN, modelA)

     INT         dictN, phonemeN, modelN;
     DICT        dictA[];
     PHONEME     phonemeA[];
     MODEL       modelA[];

{
     INT         dictX, p, variantX, phonemeX, phonemes, PhonemesInDict;
     DICT*       dictP;
     VARIANT*    variantP;
     int         i, x, y, PEsUsed;
     card        remaining[MAX_PEsY];                  /* how many PEs stay free in the row y ? */
     card        statistic[MAX_PHONESperWORD];         /* how many words have which length (in phonemes) */
     card       *cardP, longest;
     boolean     distributed;
     boolean     PhonemeDistributed[MAX_VOCAB],        /* which variants are distributed ? The entries are ordered ! */
                *boolP;
```

```
  /* initialize the local data structures : */

  cardP = remaining;
  for (i=0; i < NrOfPEsY; i++)
    *(cardP++) = NrOfPEsX;                              /* mark all PE rows as unused */

  cardP = statistic;
  for (i=0; i < MAX_PHONESperWORD; i++)
    *(cardP++) = 0;                                     /* mark "phonemes per word" - statistics as empty */

  PEsUsed = x = y = 0;                                  /* x,y indices of possible target PE */

  /* analyze the dictionary : */

  printf("\n analyzing the %d dictionary entries ...", dictN); fflush(stdout);

  boolP = PhonemeDistributed;
  PhonemesInDict = 0;
  for(dictX=0; dictX<dictN; dictX++)                    /* for all words in the dictionary */
   { dictP = &dictA[dictX];
     variantP = dictP->variantPI;
     variantX = 0;
     while (variantP != NIL)                            /* for all variants of this word */
       {
         *(boolP++) = FALSE;                            /* mark them as not yet distributed */
         PhonemesInDict += variantP->phonN;
         statistic[variantP->phonN]++;                  /* increase the "phonemes per word" - statistics */
         variantP = variantP->nextP;
         variantX++;
       }
   }
  printf("\n \"phonemes per word\" statistic :\n\n length   words\n");

  longest = 0;
  for (i=0; i < MAX_PHONESperWORD; i++)
    if (statistic[i]>0) then
      {
        printf("   %3d     %3d\n", i, statistic[i]);
        if (i>longest) then longest = i;                /* how long is the longest  word */
      }

  printf("\n %6d phonemes in the dictionary ... done.\n", PhonemesInDict);

  AssertAndExit((PhonemesInDict<=NrOfPEs), "too many phonemes in dictionary !", -1);

  /*== for all words in the vocabluary, distribute it phoneme by phoneme to the PE-matrix ==*/

  printf("\n distributing the %d dictionary entries ...\n", dictN);


  /*-- distribute all words with more than SmallWord phonemes : --*/

  for(dictX=0; dictX<dictN; dictX++)                                    /* for all words in the dictionary */
   {
#ifdef DEBUG
     printf(" %4d |", dictX);
#endif
     dictP = &dictA[dictX];
     variantP = dictP->variantPI;
     variantX = 0;

     while (variantP != NIL)                                    /* for all variants of this word */
       { phonemes = variantP->phonN;
         distributed = FALSE;

         /* search in all rows of PEs for one, with enough space in it : */
         y = 0;
         while ((phonemes>SmallWord) AND (NOT distributed) AND (y < NrOfPEsY))
           {/* if enough space to hold the word in row y, distribute it */

             if ((phonemes <= remaining[y]) AND (remaining[y]-phonemes != 1))
               then
                { x = NrOfPEsX - remaining[y];
#ifdef DEBUG
                  printf(" y=%2d x=%2d | %3d | ", y,x, phonemes);
#endif
                  for (p=0;  p<phonemes; p++)                   /* for all phonemes in this word */
                    {
                      phonemeX = variantP->phonA[p];
#ifdef DEBUG
                      printf(" %c", phonemeA[phonemeX].char1);
#endif
                      /* now distribute this phoneme to a PE */

                      put( fe_phonemeXBP, x,y, NrOfPEsX, phonemeX);

                      put( fe_variantXBP, x,y, NrOfPEsX, variantX);
                      put( fe_dictXBP,    x,y, NrOfPEsX, dictX);

                      put( fe_modelXBP,   x,y, NrOfPEsX, findModelX(modelN, modelA, phonemeA[phonemeX].modelP) );

                      put( fe_firstPhonemeBP, x,y, NrOfPEsX, ((p==0)         ? TRUE : FALSE) );/* (condition ? true : false) */
                      put( fe_lastPhonemeBP,  x,y, NrOfPEsX, ((p==phonemes-1) ? TRUE : FALSE) );

                      put( PE_usedBP,     x,y, NrOfPEsX, TRUE);
```

```
                    PEsUsed++; x++;
                  }
#ifdef DEBUG
                printf("\n");
#endif
                remaining[y] -= phonemes;                    /* keep track of the number of available PEs in that row */
                distributed = TRUE;
              }
            y++;
          }

        if ((phonemes>SmallWord) AND NOT distributed) then
           fprintf(stderr, ">>> ERROR 1 : couldn't distribute word %3d, variant %3d\n", dictX, variantX);

        variantP = variantP->nextP;
        variantX++;
      }
   }
/*
 *  show_phonemes();
 *  show_words();
 */
  /*-- now distribute all words with ShortWord phonemes and less : --*/

  for(dictX=0; dictX<dictN; dictX++)                                  /* for all words in the dictionary */
   {
#ifdef DEBUG
     printf(" %4d |", dictX);
#endif
     dictP = &dictA[dictX];
     variantP = dictP->variantPI;
     variantX = 0;

     while (variantP != NIL)                                         /* for all variants of this word */
       { phonemes = variantP->phonN;
         distributed = FALSE;

         /* search in all rows of PEs for one,with enough space in it */
         y = 0;
         while ((phonemes<=SmallWord) AND (NOT distributed) AND (y < NrOfPEsY))
           {/* if enough space to hold the word in row y, distribute it */

             if ((phonemes <= remaining[y]) AND (remaining[y]-phonemes != 1)) then
                { x = NrOfPEsX - remaining[y];
#ifdef DEBUG
                  printf(" y=%2d x=%2d | %3d | ", y,x, phonemes);
#endif
                  for (p=0;  p<phonemes; p++)                     /* for all phonemes in this word */
                    {
                      phonemeX = variantP->phonA[p];
#ifdef DEBUG
                      printf(" %c", phonemeA[phonemeX].char1);
#endif
                      /* now distribute this phoneme to a PE */

                      put( fe_variantXBP, x,y, NrOfPEsX, variantX);
                      put( fe_dictXBP,    x,y, NrOfPEsX, dictX);

                      put( fe_phonemeXBP, x,y, NrOfPEsX, phonemeX);

                      put( fe_modelXBP,   x,y, NrOfPEsX, findModelX(modelN, modelA, phonemeA[phonemeX].modelP) );

                      put( fe_firstPhonemeBP, x,y, NrOfPEsX, ((p==0)        ? TRUE : FALSE) );/* (condition ? true : false) */
                      put( fe_lastPhonemeBP,  x,y, NrOfPEsX, ((p==phonemes-1) ? TRUE : FALSE) );

                      put( PE_usedBP,     x,y, NrOfPEsX, TRUE);

                      PEsUsed++; x++;
                    }
#ifdef DEBUG
                  printf("\n");
#endif
                  remaining[y] -= phonemes;                    /* keep track of the number of available PEs in that row */
                  distributed = TRUE;
                }
            y++;
           }
         if ((phonemes<=SmallWord) AND NOT distributed) then
            fprintf(stderr, ">>> ERROR 2 : couldn't distribute word %3d, variant %3d\n", dictX, variantX);

         variantP = variantP->nextP;
         variantX++;
       }
   }
 printf(" %6d PEs used \n", PEsUsed);
 AssertAndExit((PEsUsed == PhonemesInDict), "FATAL ERROR : some word(s) couldn't be distributed !", -1);
}

/*=============================================================================================================================
 | invert_bigrams   the BAWL bigrams are stored as lists of legal successors.
 |                  put for the DTW we need the predecessors - so every wordpair is entered
 |                  into a two dimensional array of size dictN*dictN floats (the word transition penalties)
 |
 | Parameters :   in  :  dictN,  dictA[]
 |                out :  *fe_PrecWords
 |
```

```
| Note :  fe_PrecWordsBP must have allready been malloc'ed.
|
| History:
|    2.Dec 91   Tilo   Created.
|    9.Dec 91   Tilo   added the initialization of MaxPrecWordsP (where SILENCE is ignored)
|=============================================================================================================

void  invert_bigrams (dictN, dictA, fe_PrecWordsBP, NrPrecWordsBP, MaxPrecWordsP)

    INT      dictN;
    DICT     dictA[];
    ScoreT  *fe_PrecWordsBP;
    card     NrPrecWordsBP[];
    card    *MaxPrecWordsP;
{
    BIGRAM  *bigramP;
    INT     *succP;
    FLOAT   *biasP;
    int      dictX, succX, succN, succWordX, UnusedPhonemes, Unused;
    VARIANT *variantP;
    INT      variantX;
    card    *cardP;


  printf(" inverting the bigrams ...\n\n words without successors :\n\n");

  bigramP = bigramA;
  UnusedPhonemes = 0;
  for(dictX=0; dictX<dictN; dictX++, bigramP++)
   { succN = bigramP->succN;
     if ( succN == 0) then
       {
         variantP = dictA[dictX].variantPI;
         variantX = 0;
         Unused = 0;
         while (variantP != NIL)                                  /* for all variants of this word */
           {
              Unused += variantP->phonN;
              variantP = variantP->nextP;
              variantX++;
           }
 /*      printf(" %3d  %-30s   has %3d phonemes\n", dictX, dictA[dictX].spellP, Unused);
  */
         UnusedPhonemes += Unused;
       }
   }
  printf("\n a total of %d unused phonemes ...", UnusedPhonemes); fflush(stdout);

  bigramP = bigramA;

  for(dictX=0; dictX<dictN; dictX++, bigramP++)
   {
     succP = bigramP->succDictXA;
     biasP = bigramP->succBiasA;
     succN = bigramP->succN;

     for(succX=0; succX<succN; succX++)
      { succWordX = succP[succX];

       /*                from     to          maxY     penalty   */
         put(fe_PrecWordsBP, dictX, succWordX, dictN, biasP[succX]);

         NrPrecWordsBP[ succWordX ]++;
       }
   }

  *MaxPrecWordsP = 0;
  cardP = &(NrPrecWordsBP[1]);

  for(dictX=1; dictX<dictN; dictX++, cardP++)          /* we don't count predecessors of SILENCE */
    if ((*cardP) > (*MaxPrecWordsP))
       then *MaxPrecWordsP = *cardP;

  printf(" done.\n\n");
}

/*=============================================================================================================
| init_phonemes
|
| Parameters :   in  :  phonemeN,       phonemeA[]   BAWL style variables
|                out :       fe_NetsPerPhonemeA[]
|
| History:
|   10.Dec 91   Tilo   Created.
+============================================================================================================*/

VOID init_phonemes( phonemeN, phonemeA, modelA,  fe_NetsPerPhonemeA )
     INT          phonemeN;
     PHONEME      phonemeA[];
     MODEL        modelA[];
     ModelRefT    fe_NetsPerPhonemeA[];
{
     INT          i, modelX;
     PHONEME     *p;
     ModelRefT   *r;
```

```
 p = phonemeA;
 r = fe_NetsPerPhonemeA;

 for(i=0; i<phonemeN; i++, p++)
   { modelX = findModelX(modelN, modelA, p->modelP);
     *(r++) = modelA[modelX].impl.netN;
   }
}

/*=============================================================================================================================
 | invert_models    transforms all BAWL style phoneme models (forward models with positive delta states)
 |                  to models for the parallel DTW (backward models with positive and negative state numbers)
 |                  (negative state numbers are assumed to be in the previous model)
 |                  This routine looks pretty confusing, because it has access to both data structures,
 |                  BAWL style and internal ones - so, don't worry, if you need some time ...
 |
 | Parameters :   in  :  modelN,       modelA[]   BAWL style variables
 |               out :             fe_ModelA[]   new, internal format
 |
 | Note :  each BAWL-transition, that leads out of the actual model,
 |         is interpreted as a transition leading out of state 0 to a previous model.
 |
 | History:
 |   29.Nov 91   Tilo   Created.
 +============================================================================================================================*/

VOID  invert_models (modelN, modelA, fe_ModelA)
     INT      modelN;
     MODEL    modelA[];
     ModelT   fe_ModelA[];
{
     INT      m, s, t, src, dest, delta, states;
     MODEL*   modelPtr;
     STATE*   statePtr;
     TRANS*   transPtr;
     TransT  *Tptr;
     ModelT  *Mptr;
     int      nextT[MAX_StatesPerModel];              /* shows which transition to "fill out" next */


  printf("\n inverting all models ..."); fflush(stdout);

  for(m=0; m<modelN; m++)
    {
      Mptr = &(fe_ModelA[m]);

      /*-- first, initialize the target model : --*/

      strcpy (Mptr->Name, modelA[m].nameP);
      states =
      Mptr->NrOfStates = modelA[m].stateN;
      Mptr->NrOfNets   = modelA[m].impl.netN;

      for (s=0; s<MAX_StatesPerModel; s++)
        {
          nextT[s] = 0;                                /* first transition in each state has index zero */
          Mptr->Net[s] = modelA[m].stateA[s].impl.netX;
        }

      /*-- get the information of the BAWL model into the internal representation : --*/

      modelPtr = &(modelA[m]);

      for(s=0; s<modelPtr->stateN; s++)
        {
          statePtr = &(modelPtr->stateA[s]);
          for(t=0; t<statePtr->transN; t++)
            {
              transPtr = &(statePtr->transA[t]);
              delta = transPtr->deltaState;
              dest  = s + delta;                              /* s    is source state */

              if (dest >= states)
                 then {
                        Tptr = &(Mptr->Trans[states-dest][ nextT[states-dest]++ ]);     /* wuerg ! */
                        Tptr->toState = -delta;
                        Tptr->penalty = transPtr->penalty;
                      }
                 else { Tptr = &(Mptr->Trans[dest][ nextT[dest]++ ]);
                        Tptr->toState = s;
                        Tptr->penalty = transPtr->penalty;
                      }
            }
        }

      /*-- invalidate all remaining transitions : --*/

      for (s=0; s<MAX_StatesPerModel; s++)
        for (t=nextT[s]; t<MAX_TransPerState; t++)
            {
              Tptr = &(Mptr->Trans[s][t]);
              Tptr->toState = MAX_StatesPerModel;
              Tptr->penalty = INFINITY;
            }
    }
 printf(" done.\n");
```

```
}
/*+=============================================================================================================================
  |
  | read_BAWL_files               PRIVATE
  |
  | Description :
  |    does the BAWL style file i/o
  |
  | Parameter :
  |
  | Globals initialized :
  |
  | Note :
  |
  | History :
  |   19.Nov  91  Tilo     created
  |
  +=============================================================================================================================*/

void read_BAWL_files()
{
  printf(" ------------------------------------------------------------------------\n\n");
  read_model ("model", modelA, &modelN, phonemeA, &phonemeN, &networkN);
  read_rewrites("rewrites",rewriteA,&rewriteN);
  read_dict ("dict", rewriteA, rewriteN, phonemeN, dictA, &dictN);
  read_bigrams (dictN, "bigrams", bigramA);
  printf(" ------------------------------------------------------------------------\n");
}

/*#############################################################################################################################
 #
 #      PRIVATE ROUTINES
 #
 #############################################################################################################################*/

/*#############################################################################################################################
 #
 #      PUBLIC DEBUGGING ROUTINES
 #
 #############################################################################################################################*/

/*#############################################################################################################################
 #
 #      PUBLIC ROUTINES
 #
 #############################################################################################################################*/

void main(argc, argv)
      int    argc;
      char * argv[];
{
      FILE  * scores_sem;
      int     fd_hypo_sem;

#ifndef BRUTE
      extern int GiveUp();
#endif


   z1 = 3 / 10;

   get_DPU_configuration( &NrOfPEs, &NrOfPEsX, &NrOfPEsY);

   read_BAWL_files();                                          /* initializes dict, bigrams, models, phonemes */

/*
 * print_BAWL_models(modelN, modelA);
 */

   invert_models(modelN, modelA, fe_ModelA);

/*
 * print_models(modelN, fe_ModelA);
 *
 * print_BAWL_phonemes(phonemeN, phonemeA, modelN, modelA);
 */
/*  print_BAWL_words(dictN, dictA, TRUE);    /* FALSE = schreiben als Schrift, sonst : schreiben als Index ... ;-) */
/*
 * print_BAWL_bigrams (dictN, dictA, biramA);
 */

   allocate(NrOfPEs, NrOfPEsX, NrOfPEsY, dictN);

   invert_bigrams (dictN, dictA, fe_PrecWordsBP, NrPrecWordsBP, &MaxPrecWords);

   init_phonemes(phonemeN, phonemeA, modelA, fe_NetsPerPhonemeA);

/*
 * print_bigrams(dictN, NrPrecWordsBP);
 */

   map_words_to_PEs(dictN, dictA, phonemeN, phonemeA, modelN, modelA);

/*
 * print_bool_array(PE_usedBP, NrOfPEsX, NrOfPEsY);
```

```
 */

  show_phonemes();
  show_words();

/*-------------------------------------------------------------------------------------------------------------------------------
 |  MAIN CONTROL LOOP :
 +------------------------------------------------------------------------------------------------------------------------------*/

 /*  init_DTW(modelN, phonemeN, networkN, MaxPrecWords);  */

  callRequest( init_DTW, (4*4)+(6*4), modelN, phonemeN, networkN, MaxPrecWords,
               fe_modelXBP, fe_dictXBP, fe_variantXBP, fe_phonemeXBP, fe_firstPhonemeBP, fe_lastPhonemeBP );

  while (TRUE)
     {
       printf("  ------------------------------------------------------------------------\n\n");
       printf("  start polling for input file ..."); fflush(stdout);

       scores_sem = fopen(SCORES_SEM_NAME, "r");                    /* wait for scores semaphore file */
       while (scores_sem == NULL)
       {
            scores_sem = fopen(SCORES_SEM_NAME, "r");               /* wait for scores semaphore file */

#ifndef BRUTE
            GiveUp();                                               /* be a nice process - give up sometimes.     */
#endif                                                              /* No brackets allowed here for "GiveUp" !!! */
       }
       fclose(scores_sem);

       printf(" found scores semaphore file !\n");
       unlink( SCORES_SEM_NAME );                                   /* delete scores semaphore file */


       callRequest(dp_sentence, 0);                                 /* do the N-best DTW */


 /*    write_hypotheses( HYPO_FILENAME );                           /* creates and writes the scores file */


       unlink(  SCORES_FILENAME );                                  /* delete scores file */

       fd_hypo_sem = creat( HYPO_SEM_NAME, PMODE);                  /* create semaphore file for the NbestHypo -rw-rw-rw */
       close(fd_hypo_sem);

       printf("  wrote the hypotheses file.\n\n");
     }
}

/*-------------------------------------------------------------------------------------------------------------------------------
```

dp_be.h

```
/*##########################################################################################################################################
 #
 #  dp_be.h             1991 by  Tilo Sloboda,   (sloboda@ira.uka.de)
 #
 ##########################################################################################################################################
 #
 # Description :
 #
 # Note :
 #
 # Implementation Note :
 #
 # Last change :
 #   28.Nov  91  17:00  tilo    created
 #
 #
 #-- RCS Info ------------------------------------------------------------------------------------------------------------------------------
 #
 # $RCSfile$     $Revision$        $State$
 # $Date$    $Author$      $Locker$
 #
 # $Log$
 #
 ##########################################################################################################################################*/

/*----------------------------------------------------------------------------------------------------------------------------------------
 | memo
 +---------------------------------------------------------------------------------------------------------------------------------------*/

#ifndef _DP_BE_H_
#define _DP_BE_H_

/*##########################################################################################################################################
 #
 #      PUBLIC DEBUGGING ROUTINES
 #
 ##########################################################################################################################################*/

/*##########################################################################################################################################
 #
 #      PUBLIC ROUTINES
 #
 ##########################################################################################################################################*/

#endif
```

dp_be.m

```
/*****************************************************************************************************************************
 *
 *  dp_be.m             1991 by  Tilo Sloboda,   (sloboda@ira.uka.de)
 *
 *****************************************************************************************************************************
 *
 * Description :
 *   In this implementation each PE ...
 *
 * General Infos :
 *   - you should be familiar with DTW and N-best DTW, DTW for connected speech recognition.
 *   - in this implementation, each processing element (PE) stores exactly one phoneme.
 *     The phonemes of a word are stored in one column of the MasPar's two dimensional X-Net array.
 *     Multiple words can be stored in such a column.
 *   - Scores can be LPNN or LVQ scores or similar scores (dependent on the kind of speech recognizer we use).
 *   - The assumption is made, that only one phoneme of one word resides on each PE.
 *   - all N-best lists have the same length "N".
 *
 * Note :
 *   see ...
 *
 * Implementation Note :
 *
 *
 *
 * Last change :
 *   13.Oct  91  17:00  Tilo    created
 *   29.Oct  91  20:34  Tilo    some minor routines implemented
 *    6.Nov  91  21:04  Tilo    merging routines implemented
 *    8.Nov  91  11:30  Tilo     "  "   "   "   "   "
 *   13.Nov  91  15:00  Tilo    all data structures changed :-(
 *    6.Dec  91  23:00  Tilo
 *
 *-- RCS Info ------------------------------------------------------------------------------------------------------------------
 *
 * $RCSfile$     $Revision$        $State$
 * $Date$    $Author$      $Locker$
 *
 * $Log$
 *****************************************************************************************************************************/

/*-----------------------------------------------------------------------------------------------------------------------------
  | Memo :
  |        N-best mit variierbarem N --> Programm schl"agt einen Wert f"ur N vor.
  |
  | Maximale Konfigurationen :   (Stand :  14.Nov 91  -  ohne Grammatik , d.h. das Array "PrecWords" fehlte ! )
  |
  | fuer Satz-Hypothesen der Laenge 20 Worte :
  |
  |    N         laenge der utterance in frames
  |
  |    1         1190
  |    2         1100
  |    3         1030
  |    4          950
  |    5          840
  |      . . .
  |    8          600
  |   10          460
  |   12          310
  |   14          150
  |   15           70
  |   16          --- nicht moeglich :-(
  |
  | ------------------------------------
  |
  | fuer Satz-Hypothesen der Laenge 29 Worte :
  |
  |    5          730
  |      . . .
  |   10          200
  |   11           90
  |
  +----------------------------------------------------------------------------------------------------------------------------*/

/*== Includes ================================================================================================================*/

#include <mpl.h>
#include <stdio.h>
#include <string.h>
#include <sys/file.h>
#include <sys/types.h>                  /* for stat() */
#include <sys/stat.h>                   /* for stat() */

#include "ts_std.h"                     /* things, they left out of the C language                          */
#include "dp.h"                         /* contains stuff that is used by both, front end and back end      */
#include "dp_be.h"                      /* back end specific stuff                                          */
#include "fileIO_be.h"                  /* device independent low level file IO routines (reading scores) */

/*== Private Constants =======================================================================================================*/
```

```
#define PE  35          /* for debugging ! */

#define STATE_OFFSET  (MAX_PrecStates + 1)  /* offset to the first inner state of the PE's DTW_column.
                                               + 1 is a kludge, invalid transitions are mapped to DTW_Column[0][0],
                                               which is a Nbest list, filled up with "INFINITY" elements.
                                            */
#define MAX_States    (MAX_StatesPerModel\
                       + STATE_OFFSET)      /* length of a PE's DTW_column  */

#define MAX_WordsPerHypo          10        /* max. number of words in a sentence hypothesis , was 29 */

#define MAX_Duration             150        /* we have to split the time axis into such parts (16 KBytes per PE) */
                                            /* I WILL ALLOCATE THIS DYNAMICLY SOON !!! */
#define MAX_PrecWords             29

/* #define PHONEME_TRANSITION_PENALTY 0.0    not defined !  Program is faster  this way ! */
#define WORD_TRANS_PENALTY        0.005

/*== Private Types ========================================================================================================

typedef WordRefT          HypoElemT;                        /* optional a struct as the basic type.
  ----------------------------------                           additional elements could be :
   optional :  card    ElapsedTime;                          - to keep track of the elapsed time for the word,
                                                               gives us a possibility to see how long the word was.
                                                               (as an extra information for the grammar)
   optional :  ScoreT  Penalties;                            - to accumulate the word transition penalties ...
                                                               (accoustic evidence vs. phoneme penalties)
  ----------------------------------                         */

typedef HypoElemT         HypoT  [MAX_WordsPerHypo + 1];    /* +1 to store the length of the hypothesis at index 0 */

typedef struct DTWatomS
  { ScoreT      CumScore;                                   /* cummulated DTW score at that point */
    HypoT       Hypothesis;                                 /* sentence hypothesis at that point  */
  }             DTWatomT;

typedef DTWatomT          NbestListT [MAX_N];               /* a N-best list */

typedef NbestListT        DTWcolumnT [MAX_States];


/*== Private Variables ====================================================================================================

static      char        RCSid[] = "$header$";

/*== SEQUENTIAL variables for DTW : =======================================================================================

static      card        N = MAX_N;                          /* ... N-best ... */

static      address     Adr_fe_modelXB;                     /* holds front end address of array of ModelRefT */
static      address     Adr_fe_dictXB;                      /* holds front end address of array of WordRefT  */
static      address     Adr_fe_variantXB;                   /* holds front end address of array of VariantRefT  */
static      address     Adr_fe_phonemeXB;                   /* holds front end address of array of PhonemeRefT */
static      address     Adr_fe_firstPhonemeB;               /* holds front end address of array of boolean  */
static      address     Adr_fe_lastPhonemeB;                /* holds front end address of array of boolean  */

/*-----------------------------------------------------------------------------------------------------------------------
 | the scores are read into ACU memory, in packets of ChunkSize, which equals a duration of ChunkTime.
 +---------------------------------------------------------------------------------------------------------------------*/

static      ScoreT     *ScoreBP;                            /* ScoreBP[ChunkSize][NrOfNets] */

static      FILE       *fdP;                                /* pointer to file descriptor for the scores file */

static      card        ChunkSize,                          /* = ChunkTime*SizeOfFrame         */
                        ChunkTime,                          /* duration that equals ChunkSize  */
                        SizeOfFrame;                        /* = sizeof(ScoreT) * NrOfNets     */

/*-----------------------------------------------------------------------------------------------------------------------
 | the models are just needed during initialization, can be freed after distribution to the PEs
 | The NetsPerPhoneme are needed, for distribution of the actual scores to the PEs.
 +---------------------------------------------------------------------------------------------------------------------*/

static      ModelT     *ModelBP;                            /* only used for data distribution from FE to BE   */

static      IdxT       *NetsPerPhonemeBP;

static      card        NrOfModels;
static      card        NrOfPhonemes;
static      card        NrOfNets;                           /* overall number of neural nets (from model file) */
static      card        MaxPrecWords;

/*-----------------------------------------------------------------------------------------------------------------------
 | The DTW matrix is spread over the PEs, for each point of time, we just need two cloumns of the (imaginary) DTW martix.
 | The "left" of the two columns has a time stamp associated with it. This time stamp can be referenced to as an absolute
 | or a relative value. (CumTime : absolute to the beginning of the utterance; RelTime : relative to the beginning of the
 | last chunk of scores, that was read by readScores() ).
 | The two columns are indexed through DTWsrc, DTWdest - these variables are toggled after each point of time.
 +---------------------------------------------------------------------------------------------------------------------*/

static      IdxT        DTWsrc   = 0,                       /* indices for the merging steps */
                        DTWdest  = 1;

static      card        RelTime,                            /* index rel. to beginning of the last chunk of scores */
                        CumTime;                            /* index rel. to beginning of the utterance            */
```

```
/*== PARALLEL variables for DTW : ================================================================================================*/

static plural WordTransT  *PrecWordsBP;                                  /* PrecWordsBP[MAX_PrecWords]  */

static plural ScoreT      ScoreOfNet   [MAX_NetsPerModel];               /* to buffer the scores of the neural nets at the actual
                                                                           * point of time .
                                                                           */
/*-------------------------------------------------------------------------------------------------------------------------------
 | The following data structures are parts of the DTW matrix, stored locally on each PE.
 | They represent two columns (source, destination). Each of them consists of two parts :
 |  - the states of the actual model (accessed by positive indices)  and
 |  - the states of the previous model (accessed by negative indices)
 +------------------------------------------------------------------------------------------------------------------------------*/

static plural DTWcolumnT  DTWcolumn[2];

/*-------------------------------------------------------------------------------------------------------------------------------
 | informations about the local phoneme :
 +------------------------------------------------------------------------------------------------------------------------------*/

static plural card        StatesInModel,
                          NetsInModel;

static plural TransTableT TransTab;
static plural NetTableT   NetTab;

/*-------------------------------------------------------------------------------------------------------------------------------
 | informations about the local word :
 +------------------------------------------------------------------------------------------------------------------------------*/

static plural WordRefT    Word;
static plural VariantRefT Variant;                                       /* just for debugging */
static plural PhonemeRefT Phoneme;                                       /* just for debugging */
static plural ModelRefT   Model;                                         /* just for debugging and data distribution */
static plural boolean     LastPhoneme  = FALSE,
                          FirstPhoneme = FALSE;

/*-------------------------------------------------------------------------------------------------------------------------------
 | informations about the  previous phoneme (in the same word or another word) :
 +------------------------------------------------------------------------------------------------------------------------------*/

static plural IdxT        NrOfPrecStates;                                /* Number of States in preceding phoneme */

static plural boolean     IsInitialState    [MAX_StatesPerModel];        /* information about the local phoneme. */

static plural boolean     ParPrecStateArray [MAX_StatesPerModel],        /* accessed through negative, relative indices */
                        * IsPrecState;                                   /* shows, which states in the prec. phoneme
                                                                            are final states (to be fetched during phoneme-to-
                                                                            phoneme and word-to-word transitions). neg. indices !
                                                                            ... used for ph-to-ph transitions and "global or"-ing
                                                                          */
static        boolean     SeqPrecStateArray [MAX_StatesPerModel],        /* accessed through negative, relative indices */
                        * CanBeAPrecState;                               /* This is a "global or" of all "PE local" information
                                                                            about preceeding final states. negative indices !
                                                                            (where -1 means last state of previous phoneme ...)
                                                                          */


/*################################################################################################################################
 #
 #      PRIVATE MACROS
 #
 ################################################################################################################################*/

/*================================================================================================================================
 |
 | access_Score                  PRIVATE   MACRO
 |
 | Description :   gives read and write access to the Scores  (access to malloc'ed, two dimensional arrays)
 |                 indices used as in C (y is rolled out first) :
 | History :
 |    5.Dec  91  Tilo   created.
 +================================================================================================================================*/

#define access_Score( x,y )   /* x,y indices in that array ; returns the value, stored in this element */ \
                                                                                                            \
   *((ScoreBP) + ((x)*SizeOfFrame) + (y))  \

/*================================================================================================================================
 |
 | ToggleIndices                   PRIVATE  SINGULAR   MACRO
 |
 | Description :
 |    toggles two singular indices, that have the value 0 or 1, both indices are assumed to be different.
 |
 | History :
 |   10.Nov  91  Tilo   created.
 +================================================================================================================================*/

#define ToggleIndices(a,b)  /* a=0,b=1 or a=1,b=0 */  \
   if ((a)==0)                                        \
      then { (a)=1; (b)=0;}                           \
      else { (a)=0; (b)=1;}                           \
```

```
/*###########################################################################################################################
 #
 #      PRIVATE DEBUGGING ROUTINES
 #
 ###########################################################################################################################*/

/* only the PE with number pe is active : */

void showNbestList(N, p_listAP, pe)

            int                 N, pe;
    plural DTWatomT * plural    p_listAP;
{
    plural DTWatomT *           listAP;
            int                 n;
            WordRefT            h, HypoLen;
    plural WordRefT * plural    HypoP;
            ScoreT              CumScore;

  listAP = proc[pe].p_listAP;

  printf("\n  N    CumScore    HypoLen\n",  pe);
  for(n=0; n<N; n++)
    {
      CumScore = proc[pe].listAP[n].CumScore;
      HypoLen  = proc[pe].listAP[n].Hypothesis[0];

      printf(" %2d    %6.4f     %2d     | ",    n, CumScore,  HypoLen);

      HypoP = listAP[n].Hypothesis;  HypoP++;

      if (iproc == pe) then
        for (h=0; h<HypoLen; h++, HypoP++)
          p_printf(" %2d", *HypoP );

      printf("\n");
    }
}
/*+=========================================================================================================================
 |
 +=========================================================================================================================*/

void showPElist(N, pe, listX, str)
    int         N, pe, listX;
    char       *str;
{
    StateRefT  s;

  printf("\n %s  PE %d   ListX=%d   DTWsrc=%d   DTWdest=%d\n",  str, pe, listX, DTWsrc, DTWdest);

  for (s=0; s<MAX_PrecStates+MAX_StatesPerModel+1; s++)
      showNbestList(N, (plural DTWatomT * plural) DTWcolumn[listX][s], pe);
}

/*+=========================================================================================================================
 |
 +=========================================================================================================================*/

void showSizes()
{
  printf("\n sizes of the main data types :\n\n");

  printf(" sizeof(ScoreT)     =  %d\n", sizeof(ScoreT));
  printf(" sizeof(IdxT)       =  %d\n", sizeof(IdxT));
  printf(" sizeof(LIdxT)      =  %d\n", sizeof(LIdxT));
  printf(" sizeof(WordTransT) =  %d\n", sizeof(WordTransT));
  printf(" sizeof(TransT)     =  %d\n", sizeof(TransT));
  printf(" sizeof(TransDescrT)=  %d\n", sizeof(TransDescrT));
  printf(" sizeof(TransTableT)=  %d\n", sizeof(TransTableT));
  printf(" sizeof(NetTableT)  =  %d\n", sizeof(NetTableT));
  printf(" sizeof(ModelT)     =  %d\n", sizeof(ModelT));
  printf(" sizeof(PhonemeT)   =  %d\n", sizeof(PhonemeT));
  printf(" sizeof(HypoElemT)  =  %d\n", sizeof(HypoElemT));
  printf(" sizeof(HypoT)      =  %d\n", sizeof(HypoT));
  printf(" sizeof(DTWatomT)   =  %d\n", sizeof(DTWatomT));
  printf(" sizeof(NbestListT) =  %d\n", sizeof(NbestListT));
  printf(" sizeof(DTWcolumnT) =  %d\n", sizeof(DTWcolumnT));

  printf("\n sizes and addresses of the main data structures (decimal) :\n\n");
  printf(" ACU variables :\n\n");

  printf(" PE  variables :\n\n");


  printf(" address of DTWcolumn[0] : %8d\n", DTWcolumn[0] );
  printf(" address of DTWcolumn[1] : %8d\n", DTWcolumn[1] );

  printf(" address of ScoreOfNet   : %8d  ; size : %4d bytes\n", ScoreOfNet, sizeof(ScoreOfNet) );

  printf("\n");
}

/*###########################################################################################################################
 #
 #      PRIVATE ROUTINES
 #
```

```
/*****************************************************************************************************************************
|
| pp_CopyDTWatom                    PRIVATE  SINGULAR  FUNCTION
|
| Description :
|    copies a DTW atom (one item of a N-best list) from source to destination.
|
| Parameters :
|     src    plural pointer to plural DTWatomT
|     dest   plural pointer to plural DTWatomT
|
| Implementation Note :
|    it's assumed, that the first entry in a HypoT array is the length of the hypothesis.
|    so we just have to copy as many entries as necessary (incl. the length information).
|
| History :
|   29.Oct  91  Tilo   created
|   16.Nov  91  Tilo   changed it to a macro (now handles both singular and plural data types)
|   13.Dec  91  Tilo   plural pointers to plural data !
*****************************************************************************************************************************/

void pp_CopyDTWatom(src,  dest)
   plural DTWatomT * plural  src;
   plural DTWatomT * plural  dest;
{
   plural int            max, i;

  dest->CumScore = src->CumScore;
  max = src->Hypothesis[0];

  for(i=0; i<=max; i++)                          /* sequential loop */
     dest->Hypothesis[i] = src->Hypothesis[i];   /* copy word indices and length of hypothesis */
}


/*****************************************************************************************************************************
|
| initACUmem                        PRIVATE   SINGULAR
|
| Description :
|    called once, after vacabluary was read into the front end and NrOfModels is known.
|    allocates the memory for some ACU variables, and initializes them.
|
| History :
|   6.Dec  91  Tilo   created
|   9.Dec  91  Tilo   initialization changed, Scores are now held in the ACU !
|  10.Dec  91  Tilo   some changes
*****************************************************************************************************************************/

void initACUmem( NrOfModels, NrOfPhonemes, NrOfNets,                          /* in  */
                 ModelBPP, NetBPP, ChunkSizeP, ChunkTimeP, ScoreBPP )         /* out */

    card        NrOfModels, NrOfPhonemes, NrOfNets, *ChunkSizeP, *ChunkTimeP;
    ModelT    **ModelBPP;
    IdxT      **NetBPP;
    ScoreT    **ScoreBPP;
{
  printf("\n allocating ACU variables ..."); fflush(stdout);

  *ModelBPP    = (ModelT *)    malloc(NrOfModels * sizeof(ModelT));
  *NetBPP      = (IdxT *)      malloc(NrOfPhonemes * sizeof(IdxT));
  *ScoreBPP    = (ScoreT *)    malloc(MAX_Duration * SizeOfFrame);

  AssertAndExit((*ModelBPP    != NIL),"allocating ModelBP on the ACU failed",  -1);
  AssertAndExit((*NetBPP      != NIL),"allocating NetsPerPhonemeBP on the ACU failed",  -1);
  AssertAndExit((*ScoreBPP    != NIL),"allocating ScoreBP on the ACU failed",  -1);

  printf("\n address of ModelBP        : %8x  ; size : %4d bytes ; models   : %4d\n",
         *ModelBPP, NrOfModels * sizeof(ModelT), NrOfModels );
  printf(  " address of NetsPerPhoneme : %8x  ; size : %4d bytes ; phonemes : %4d\n",
         *NetBPP,   NrOfPhonemes * sizeof(IdxT), NrOfPhonemes );
  printf(  " address of Scores         : %8x  ; size : %4d bytes ; duration : %4d\n",
         *ScoreBPP, MAX_Duration * SizeOfFrame, MAX_Duration );

  printf(" done.\n loading models and phoneme info  into ACU variables ..."); fflush(stdout);

  copyIn(fe_ModelA, *ModelBPP, NrOfModels*sizeof(ModelT) );
  copyIn(fe_NetsPerPhonemeA, *NetBPP, NrOfPhonemes*sizeof(IdxT) );

  *ChunkSizeP = MAX_Duration * SizeOfFrame;
  *ChunkTimeP = MAX_Duration;

  printf(" done.\n");
}
/*****************************************************************************************************************************
|
| initPEmem                         PRIVATE   SINGULAR
|
| Description :
|    called once, after vacabluary was read into the front end.
|    allocates the memory for some PE variables and initializes them.
|
| History :
|   6.Dec  91  Tilo   created
|   9.Dec  91  Tilo   initialization changed.
```

```
+=============================================================================================================================

void initPEmem( MaxPrecWords,                /* in  */
                p_PrecWordsBPP )             /* out */

          card                  MaxPrecWords;
    plural WordTransT * plural  * p_PrecWordsBPP;
{

  printf(" allocating PE variables ..."); fflush(stdout);
/*
  *p_PrecWordsBPP = (plural WordTransT * plural) p_malloc(MaxPrecWords * sizeof(WordTransT));

  AssertAndExit((*p_PrecWordsBPP != NIL),"allocating PrecWordsBP on the PEs failed",-1);
 */
  printf(" done.\n loading values into PE variables ..."); fflush(stdout);

  blockIn(Adr_fe_phonemeXB,      &Phoneme,      0,0, nxproc,nyproc, sizeof(PhonemeRefT));
  blockIn(Adr_fe_modelXB,        &Model,        0,0, nxproc,nyproc, sizeof(ModelRefT));
  blockIn(Adr_fe_dictXB,         &Word,         0,0, nxproc,nyproc, sizeof(WordRefT));
  blockIn(Adr_fe_variantXB,      &Variant,      0,0, nxproc,nyproc, sizeof(VariantRefT));
  blockIn(Adr_fe_firstPhonemeB,  &FirstPhoneme, 0,0, nxproc,nyproc, sizeof(boolean));
  blockIn(Adr_fe_lastPhonemeB,   &LastPhoneme,  0,0, nxproc,nyproc, sizeof(boolean));
/*
  blockIn(Adr_fe_PrecWordsB,   *p_PrecWordsBP,  0,0, nxproc,nyproc, MaxPrecWords*sizeof(WordTransT));
 */
  printf(" done.\n");
}
/*=============================================================================================================================
 |
 | distributeModels                          PRIVATE   PLURAL
 |
 | Description :
 |    called once, after initACUmem()
 |    distributes the Model information from the ACU to the PEs according to the PE variable "Model"
 |
 | History :
 |   9.Dec  91  Tilo   created
 |  13.Dec  91  Tilo   TransTab is initialized now
 +============================================================================================================================*/

void distributeModels( NrOfModels, ModelBP )

          card      NrOfModels;
          ModelT  * ModelBP;
{
          TransT  * TransP;
    plural TransT * pTransP;
          IdxT    * NetP;
    plural IdxT   * pNetP;
          card      m,t,s;

 /*----------------------------------------------------------------------------------------------------------------------------
  | initialize the transition table
  +---------------------------------------------------------------------------------------------------------------------------*/

  pTransP    = (plural TransT *) TransTab;

  for(s=0; s<MAX_StatesPerModel; s++)
    for(t=0; t<MAX_TransPerState; t++)
      {
        pTransP->toState = MAX_StatesPerModel;          /* eg. : 6   -   means : invalid transition */
        pTransP->penalty = INFINITY;
        pTransP++;
      }

 /*----------------------------------------------------------------------------------------------------------------------------
  | copy the models from the ModelB to the PEs :
  +---------------------------------------------------------------------------------------------------------------------------*/

  for (m=0; m<NrOfModels; m++, ModelBP++)
    {
      if (Model == m) then                              /* for all PEs with model m */
        {
          StatesInModel = ModelBP->NrOfStates;
          NetsInModel   = ModelBP->NrOfNets;

        /* copy the transition table : */

          TransP     = (TransT *) (ModelBP->Trans);
          pTransP    = (plural TransT *) TransTab;

          for(s=0; s<MAX_StatesPerModel; s++)
            for(t=0; t<MAX_TransPerState; t++)
              {                                         /* copy struct (one transition) */
                pTransP->toState = TransP->toState;
                pTransP->penalty = TransP->penalty;
                pTransP++; TransP++;
              }
        /* copy the net table : */

          NetP       = (IdxT *) ModelBP->Net;
          pNetP      = (plural IdxT *) NetTab;

          for(s=0; s<MAX_StatesPerModel; s++)
              *(pNetP++) = *(NetP++);                    /* copy one net index */
```

```
        }
    }
}

/*=============================================================================================================================
 |
 | reinit_DTW                PRIVATE   SINGULAR
 |
 | Description :
 |    called every time, a new DTW has to be performed (prior to the first dtw step). init_DTW has to be called once before.
 |    initializes the PE's part of the DTW matrix, temp-variables, sentence hypothesis, ...
 |    Only the NN scores of the silence phoneme are mapped to the initial states of silence.
 |    All others are BIG_NUMBER.   All hypothesis have length zero.
 |
 | Globals used :
 |
 | Implementation Note :
 |
 | History :
 |   13.Nov  91  Tilo   created
 |    7.Dec  91  Tilo   changed it a little (other DTWcolumn structure)
 |   12.Dec  91  Tilo   changed initialisation.
 |   17.Dec  91  Tilo   valid sentences have to start (and end) with the silence phoneme
 +============================================================================================================================*/

void reinit_DTW()
{
    register plural DTWatomT            *ACUatomP0;          /* ACU pointers to plural DTWatoms */
    register plural DTWatomT            *ACUatomP1;          /* ACU pointers to plural DTWatoms */

                card                     i,items;
                card                     s, n;
                card                     nets;
           plural DTWatomT   * plural    atomP;
           plural ScoreT     * plural    p_scoreP;
           plural ScoreT                 actScore;


 /*----------------------------------------------------------------------------------------------------------------------------
  | initialize the two DTW columns to cumScore == BIG_NUMBER   and   hypotheses with length zero
  +---------------------------------------------------------------------------------------------------------------------------*/
 ACUatomP0 = (plural DTWatomT *)DTWcolumn[0];          /* pointer to the first DTW column */
 ACUatomP1 = (plural DTWatomT *)DTWcolumn[1];          /* pointer to the second DTW column */

 items = MAX_N * MAX_States;

 for(i=0; i<items; i++)                                /* for all elements of all Nbest lists in one column ... */
   {                                                   /* initialize both columns with one loop */
     ACUatomP0->CumScore      = BIG_NUMBER;            /* set initial score to a "unlikely" value */
     ACUatomP0->Hypothesis[0] = 0;                     /* no hypo. yet - set the hypo. length to zero */
     ACUatomP1->CumScore      = BIG_NUMBER;            /* set initial score to a "unlikely" value */
     ACUatomP1->Hypothesis[0] = 0;                     /* no hypo. yet - set the hypo. length to zero */

     ACUatomP0++; ACUatomP1++;
   }

 /*----------------------------------------------------------------------------------------------------------------------------
  | map next scores for time==0  from the ACU to PE local buffer variable :
  | (only the scores for the silence phoneme have to be mapped here)
  +---------------------------------------------------------------------------------------------------------------------------*/
 p_scoreP  = (plural ScoreT * plural) ScoreOfNet;
 nets   = NetsPerPhonemeBP[SILENCE];                   /* how many nets has the SILENCE phoneme ? */
 if (Phoneme == SILENCE)
   then {
          printf(" phoneme %2d, %1d nets, PEs :", SILENCE, nets );
          p_printf(" %d",iproc);
          printf("\n");

          for (n=0; n<nets; n++)                       /* sequential loop is faster */
            *(p_scoreP++) = access_Score(0, n);        /* time == 0 */
        }

 /*----------------------------------------------------------------------------------------------------------------------------
  | copy the scores of the initial states in the FirstPhonemes of SILENCE to the actual DTW-source-column :
  +---------------------------------------------------------------------------------------------------------------------------*/
 if (FirstPhoneme AND (Phoneme == SILENCE)) then
   for (s=0; s<StatesInModel; s++)                                 /* O(N * MAX_StatesPerModel) */
     if (IsInitialState[s]) then
       {
         atomP =    DTWcolumn[DTWsrc][MAX_PrecStates+ s];         /* get the pointer to the Nbest list for this state    */
         actScore = ScoreOfNet[ NetTab[s] ];                      /* get state's actual score (which is allready mapped) */

         atomP->CumScore = actScore;                              /*   copy actual score to cummulated score of list zero */
   }
}

/*#############################################################################################################################
 #
 #      PRIVATE ROUTINES   FOR FILE IO
 #
 #----------------------------------------------------------------------------------------------------------------------------
 #  The scoresfile is opened by openScores, some variables get initialized.
 #  As the scoresfile is quite large (500 KBytes) compared to the size of ACU's memory,
 #  portions of this file are read in by the routine readScores.
 #  readScores tries to read the scores for a number of time frames. It returns the actual number of frames read.
 #  finally the file is closed by closeScores
```

```
/*##########################################################################################################################
  |
  | openScores                          PRIVATE   SINGULAR
  |
  | Description :
  |    opens a score file, sets the file descriptor fd, sets the pointer ScoreP to the beginning of the Scores,
  |    resets TimeSpanRead to zero, sets the variable "duration" to the duration of the utterance.
  |
  | History :
  |   9.Dec  91  Tilo   created
  |
  +=========================================================================================================================*/
card openScores(FileName, SizeOfFrame,        /* in  */
                fdPP)                         /* out */
      char    *FileName;
      card     SizeOfFrame;
      FILE   **fdPP;
{
         card  Duration;
   struct stat  StatBuf;

  /* open the scores file (approx. size 500 KBytes) */

  *fdPP = fopen(FileName, "r");
  if (*fdPP == NULL)
     then { fprintf(stderr, " >> FATAL ERROR : can not open scores file !\n");
            exit (-1);
          }
     else { stat(FileName, &StatBuf);
            Duration = (StatBuf.st_size) DIV SizeOfFrame;
            AssertAndExit( (((StatBuf.st_size) MOD SizeOfFrame) == 0), "FATAL ERROR : odd length of scores file !", -1);

            printf("\n  Reading the scores file \"%s\" ,covering %6.3f seconds of speech, to ACU memory.\n",
                   FileName, ((float)Duration/100) );
          }
  return(Duration);
}

/*=========================================================================================================================
  |
  | readScores                          PRIVATE   SINGULAR
  |
  | Description :
  |
  | returns : the number of score-frames, that were read
  |
  | History :
  |   9.Dec  91  Tilo   created
  +=========================================================================================================================*/

card readScores(fdP, SizeOfFrame, NrOfNets, RelDuration)     /* in  */

     FILE     *fdP;
     card      SizeOfFrame, NrOfNets, RelDuration;
{
     boolean   erg;
     card      i, scores;
     ScoreT   *actScoreP;                              /* points into ScoreBP */

 actScoreP = ScoreBP;
  erg = fread(actScoreP, SizeOfFrame, RelDuration, fdP);   /* read several whole score-frames */
  if (0 < erg)
    then { scores = RelDuration * NrOfNets;                /* the number of scores, that were read */
           for (i=0; i<scores; i++, actScoreP++)           /* convert every score from the ...     */
             READ_FLOAT_SWAP( *actScoreP);                /* ... machine independent format, to the internal format */
         }
    else fprintf (stderr, " >> couldn't read %d score-frames, error number : %d\n", RelDuration, erg);

  return (erg);
}

/*=========================================================================================================================
  |
  | closeScores                         PRIVATE   SINGULAR
  |
  | Description :
  |
  | History :
  |   9.Dec  91  Tilo   created
  +=========================================================================================================================*/

void closeScores(fdP)
   FILE   *fdP;
{
  fclose(fdP);

  printf("  finished reading the scores file and closed it.\n\n");
}

/*##########################################################################################################################
  #
  #      PRIVATE ROUTINES   FOR COPYING AND MERGING
  #
  ##########################################################################################################################*/
```

```
/*=============================================================================================================================
  |
  | HyposEqual                          PRIVATE   SINGULAR
  |
  | Description :
  |    compares two sentence Hypothesis
  |
  | Implementation Note :
  |    it's assumed, that the first entry in a HypoT array is the length of the hypothesis.
  |    hypos are different, if they are of different length.
  |
  | History :
  |   29.Oct  91  Tilo   created
  |
  +============================================================================================================================*/

boolean HyposEqual (h1, h2)
        HypoElemT   h1[], h2[];
{
        boolean   IsEqual;
        int       i, len;

   if (h1[0] != h2[0])                                          /* if the length differs, they are different ! */
      then  return FALSE;
      else { IsEqual = TRUE;
             len = h1[0];
             for (i=len; i>=1; i--)
                 IsEqual = ( IsEqual AND (h1[i] == h2[i]) );
           }
   return (IsEqual);
}


/*=============================================================================================================================
  |
  | get_WordPenalty                     PRIVATE   PLURAL
  |
  | Description :
  |    get the word transition penalty, for a given PreviousWord.
  |    If the word is not allowed as a predecessor, return INFINITY.
  |
  | History :
  |   16.Nov  91  Tilo   created
  |
  +============================================================================================================================*/
#ifdef 0
            ######  B A U S T E L L E  ##########################################################################
plural
ScoreT get_WordPenalty (PreviousWord)

            WordRefT     PreviousWord;
{
            int          i;
    plural ScoreT        penalty = (plural ScoreT) INFINITY;
    plural WordTransT   *WordPtr;


  WordPtr = PrecWordsBP;

  for (i=0; i<MAX_PrecWords; i++, WordPtr++)
     if (WordPtr->word == PreviousWord)
        then penalty = WordPtr->penalty;

  return (penalty);
}
            ##############################################################################################################
#endif
/*=============================================================================================================================
  |
  | FinalNbestMerge                PRIVATE   SINGULAR
  |
  | Description :
  |    merges all final N-best lists into one N-best list.  -   Is used at the end of the DTW.
  |    All valid hypotheses have to end with SILENCE.
  |
  | Globals used :
  |
  | Implementation Note :
  |
  | NOTE :  falls mehrere Instanzen von Satzende-SILENCE verwendet werden, muessen deren N-best-Listen
  |         noch gemischt werden !
  |
  |
  | History :
  |   18.Nov  91  Tilo   created
  |
  +============================================================================================================================*/

void  FinalNbestMerge ()
{
      int         FinalLists = 0;
      StateRefT   s;
      char        dummy[128];
      int         pe;

 /* zu einfach : */
```

```
 printf("\n Ausgabe der N best Listen von SILENCE : \n");

 if ((Word == SILENCE) AND (LastPhoneme)) then
   for (s= -1; s>-StatesInModel; s--)
     {
       printf("rel. State %d   ,  abs. State %d\n", s, MAX_PrecStates+ StatesInModel+s);

      if (CanBeAPrecState[s])
        then showNbestList(N, (plural DTWatomT * plural) DTWcolumn[DTWdest][MAX_PrecStates+ StatesInModel+s], selectOne() );
     }

 printf("\n press RETURN to continue"); scanf("%*c");

 printf("\n Ausgabe aller N best Listen : \n");

 for (pe=0; pe<42; pe++)
   if (iproc == pe) then
     {
       printf(" PE=%4d, ", pe);
       p_printf("Word=%3d, Phoneme=%2d, First=%d Last=%d\n", Word, Phoneme, FirstPhoneme, LastPhoneme );

       for (s=0; s<StatesInModel; s++)
         {
           printf("State %d: \n", s);

           showNbestList(N, (plural DTWatomT * plural) DTWcolumn[DTWdest][MAX_PrecStates+s], pe);
         }
       printf("\n press RETURN to continue"); scanf("%*c");
     }
/* ------------------------------------------------------------------------------------------------------------------------
   all if ((Word == SILENCE) AND (LastPhoneme)) then
         {
           for (s= -1; s>-StatesInModel; s--)
             if (CanBeAPrecState[s])
                then { FinalLists++;
                       showNbestList(N, (plural DTWatomT * plural) DTWcolumn[DTWdest][MAX_PrecStates+ StatesInModel+s],
                                     selectOne(), "Hypotheses with SILENCE as the last word :");

                       Copy_NbestList_to_FE(   DTWcolumn[DTWdest][MAX_PrecStates+ StatesInModel+s]  );
                     }
         }
    ------------------------------------------------------------------------------------------------------------------------
 */
}


/*#########################################################################################################################
 #
 #      PRIVATE ROUTINES   FOR COPYING AND MERGING
 #
 #########################################################################################################################

/*=========================================================================================================================
 |
 | merge2_NbestLists                PRIVATE   PLURAL
 |
 | Description :
 |    Used during the pre-calculation of the word-to-word transitions.
 |    Merges two N-best lists into one N-best list. The usage of each source implies a penalty.
 |    The two penalties are "compound penalties" ... max. 2 penalties are added  here  for each of them :
 |      one for a phoneme-to-phoneme transition (constant) and
 |      one for a word-to-word transition (variable)
 |
 | Parameters :
 |    N           the length of a N-best list
 |    src1,src2   two pointers to two source N-best lists
 |    dest        pointer to a destination N-best list
 |    penalty1,   penaltiy, coresponding to the transition to src1.   (max. 2 penalties are added here !)
 |    penalty2    penaltiy, coresponding to the transition to src2.   (max. 2 penalties are added here !)
 |
 | Globals used :  none.
 |
 | Implementation Note :
 |
 | History :
 |   28.Oct  91  Tilo   created
 |   30.Oct  91  Tilo
 |    6.Nov. 91  Tilo   minor change
 |    6.Dec  91  Tilo   plural pointers to plural data !
 |   13.Dec  91  Tilo   sequential penalties
 +=========================================================================================================================
plural
void merge2_NbestLists (N,  src1, penalty1,  src2, penalty2,  dest)

          int                N;                          /* length of a N-best list */
    plural DTWatomT * plural  src1;                       /* pointers to the N-best lists */
    plural DTWatomT * plural  src2;
    plural DTWatomT * plural  dest;
          ScoreT             penalty1, penalty2;         /* coresponding transition penalties to the sources */
{
    plural ScoreT            Score1, Score2;             /* temporal storage */
          int                n;                          /* counter */


  AssertAndExit ( ((dest != NIL) AND (src1 != NIL) AND (src2 != NIL)), "in merge2_NbestLists", -1 );
```

```
  Score1 = src1->CumScore + penalty1;
  Score2 = src2->CumScore + penalty2;

  for (n=0; n<N; n++)                                          /* until all items of destinations N-best list are sampled */
     if (Score1 <= Score2)
        then { pp_CopyDTWatom (src1, dest);
               dest->CumScore = Score1;                        /* if penalties shall be added to the cummulative DTW-Score */
               src1++; Score1 = src1->CumScore + penalty1;
               dest++;
             }
        else { pp_CopyDTWatom (src2, dest);
               dest->CumScore = Score2;                        /* if penalties shall be added to the cummulative DTW-Score */
               src2++; Score2 = src2->CumScore + penalty2;
               dest++;
             }
}

/*=============================================================================================================================
  |
  | merge3_NbestLists                PRIVATE   PLURAL
  |
  | Description :
  |    Used during the final-calculation of the inner-word transitions.
  |    Merges three N-best lists into one N-best list. The usage of each source implies a penalty.
  |    The three penalties are "compound penalties" ... max. 3 penalties are added for each of them :
  |      one penalty coresponding to the state-to-state transition (see model file),
  |      one for a phoneme-to-phoneme transition (constant) and
  |      one for a word-to-word transition (variable)
  |
  | Parameters :
  |    N           the length of a N-best list
  |    src1,src2,
  |    src3        two pointers to two source N-best lists
  |    dest        pointer to a destination N-best list
  |    penalty1,   penaltiy, coresponding to the transition to src1.   (max. 3 penalties are added here !)
  |    penalty2,   penaltiy, coresponding to the transition to src2.   (max. 3 penalties are added here !)
  |    penalty3    penaltiy, coresponding to the transition to src3.   (max. 3 penalties are added here !)
  |
  | Globals used :  none.
  |
  | Implementation Note :
  |
  | History :
  |    7.Nov. 91  Tilo   created
  |    6.Dec  91  Tilo   plural pointers to plural data !
  +============================================================================================================================*/
plural
void merge3_NbestLists (N,  src1, penalty1,  src2, penalty2,  src3, penalty3,  dest)

          int                N;                              /* length of a N-best list */
    plural DTWatomT * plural  src1;                           /* pointers to the N-best lists */
    plural DTWatomT * plural  src2;
    plural DTWatomT * plural  src3;
    plural DTWatomT * plural  dest;
    plural ScoreT             penalty1, penalty2, penalty3; /* coresponding transition penalties to the sources */
{
    plural ScoreT             Score1, Score2, Score3;       /* temporal storage */
          int                 n;                            /* counter */


  AssertAndExit ( ((dest != NIL) AND (src1 != NIL) AND (src2 != NIL) AND (src3 != NIL)), "in merge3_NbestLists", -1 );

  Score1 = src1->CumScore + penalty1;
  Score2 = src2->CumScore + penalty2;
  Score3 = src3->CumScore + penalty3;

  for (n=0; n<N; n++)                                          /* until all items of destinations N-best list are sampled */
   { if (Score1 <= Score2)
        then if (Score1 <= Score3)
                then { pp_CopyDTWatom (src1, dest);
                       dest->CumScore = Score1;                /* if penalties shall be added to the cummulative DTW-Score */
                       src1++;
                       Score1 = src1->CumScore + penalty1;
                       dest++;
                     }
                else { pp_CopyDTWatom (src3, dest);
                       dest->CumScore = Score3;                /* if penalties shall be added to the cummulative DTW-Score */
                       src3++;
                       Score3 = src3->CumScore + penalty3;
                       dest++;
                     }
        else if (Score2 <= Score3)
                then { pp_CopyDTWatom (src2, dest);
                       dest->CumScore = Score2;                /* if penalties shall be added to the cummulative DTW-Score */
                       src2++;
                       Score2 = src2->CumScore + penalty2;
                       dest++;
                     }
                else { pp_CopyDTWatom (src3, dest);
                       dest->CumScore = Score3;                /* if penalties shall be added to the cummulative DTW-Score */
                       src3++;
                       Score3 = src3->CumScore + penalty3;
                       dest++;
                     }
   }
```

```
}

/*##########################################################################################################################
 #
 #      PRIVATE ROUTINES   FOR DTW FUNCTIONS
 #
 ##########################################################################################################################*/

/*==========================================================================================================================
 |
 | map_and_add_Scores                    PRIVATE   PLURAL
 |
 | Description :
 |    Maps the actual Scores from the ACU to the PEs and adds them to the PE local DTW column "DTWdest".
 |
 | Implementation Note :
 |    It is assumed, that the neural Nets, that are used in one phoneme, have successive indices and
 |    that   NetTab[0] <= NetTab[1] <= ... <= NetTab[max]
 |
 | History :
 |   10.Dec  91  Tilo   created
 |
 +=========================================================================================================================*/

void map_and_add_Scores(RelTime, DTWdest, NrOfPhonemes, NrOfNets, N, NetsPerPhonemeBP)

          card                  RelTime, DTWdest, NrOfPhonemes, NrOfNets, N;
          IdxT                  NetsPerPhonemeBP[];
{
          card                  p, s, n;
          card                  actNetX, nets;
   plural DTWatomT    * plural  atomP;
   plural ScoreT      * plural  p_scoreP;
   plural ScoreT                actScore;

 /*-------------------------------------------------------------------------------------------------------------------------
  | map next scores from the ACU to PE local buffer variable :
  +-------------------------------------------------------------------------------------------------------------------------*/
  actNetX = 0;                                                  /* counts from 0...117 (if we are using 118 nets) */
  p_scoreP = (plural ScoreT * plural) ScoreOfNet;

  for(p=0; p<NrOfPhonemes; p++)                                 /* for all Phonemes that are mapped to the PEs ; O(NrOfNets) *
     { nets   = NetsPerPhonemeBP[p];

       if (Phoneme == p) then
         for (n=0; n<nets; n++)                                 /* sequential loop is faster */
           *(p_scoreP++) = access_Score(0, actNetX+n);          /* time == 0 */

       actNetX += nets;
     }

 /*-------------------------------------------------------------------------------------------------------------------------
  | add the score for the actual dtw-column to the results :
  +-------------------------------------------------------------------------------------------------------------------------*/

  for (s=0; s<StatesInModel; s++)                               /* O(N * MAX_StatesPerModel) */
   {
     atomP =    DTWcolumn[DTWdest][MAX_PrecStates+ s];         /* get the pointer to the Nbest list for this state    */
     actScore = ScoreOfNet[ NetTab[s] ];                       /* get state's actual score (which is allready mapped) */

     for (n=0; n<N; n++, atomP++)                               /* for all atoms in this list ...             */
        atomP->CumScore += actScore;                            /*    add actual score to cummulated score    */
   }
}

/*==========================================================================================================================
 |
 | phoneme_transitions                PRIVATE   PLURAL
 |
 | Description :
 |    performes the dp-function between phonemes, in every word.
 |    and add phoneme transition penalties
 |    therefore it copies the needed N-best lists for the phoneme-to-phoneme transitions
 |
 | Globals used :
 |
 |
 | Implementation Note :
 |    All transitions are from-state , to-previous-state. If a state number is negative,
 |    the state is in a previous phoneme.
 |    For all phonemes in words (excepting the first phoneme of a word), transfer the CumScores to the successive phoneme
 |    in the same word.  The X-Net is used for this transfer. Words are not wrapped around ixproc borders - so, we can use
 |    simple X-Net operations of length one, in just one direction (east).
 |    The assumption is made, that only one phoneme of one word resides on each PE.
 |
 | History :
 |   28.Oct  91  Tilo   created
 |   16.Nov  91  Tilo   simplified.
 |
 +=========================================================================================================================*/
plural
void phoneme_transitions()
{
     register         IdxT        src, n;
              plural  IdxT        RelNegState;
```

```
    register        DTWatomT   *ACUatomPtr;

 all if (NOT FirstPhoneme) then                                  /* "not first phoneme of word" implies that ixproc > 0      */
       for(src=0; src<NrOfPrecStates; src++)                     /* for all states in previous phoneme ...                   */
         { RelNegState = src - NrOfPrecStates;                   /* get relative, negative index for this state              */

           if (CanBeAPrecState[RelNegState]) then                /* if we need this state of the previous phoneme, copy it ! */
             { ss_xfetch( -1, 0,                                 /* ss_xfetch( dx, dy, srcPtr, destPtr, bytes);              */
                 DTWcolumn[DTWsrc][MAX_PrecStates+ src],         /* copies a whole NbestList from previous phoneme           */
                 DTWcolumn[DTWsrc][MAX_PrecStates+ RelNegState],     /* to target state in our (this) phoneme        */
                  sizeof(NbestListT) );

#ifdef PHONEME_TRANSITION_PENALTY
             /*-------------------------------------------------------------------------------------------------------------
              | add phoneme transition penalties
              +-------------------------------------------------------------------------------------------------------------*/

              ACUatomPtr =  DTWcolumn[DTWsrc][MAX_PrecStates+ RelNegState];   /* Spezialfall SILENCE beachten  ???    */
              for (n=0; n<N; n++, ACUatomPtr++)
                ACUatomPtr->CumScore += PHONEME_TRANSITION_PENALTY;
#endif PHONEME_TRANSITION_PENALTY

           }
         }
}

/*=============================================================================================================================
  |
  | word_transitions                  PRIVATE   PLURAL
  |
  | Description :
  |    performes the dp-function between phonemes, at word boundaries.
  |    and add word transition penalties.
  |
  | Globals used :
  |
  |
  | Implementation Note :
  |    All transitions are from-state , to-previous-state. If a state number is negative,
  |    the state is in a previous phoneme.
  |    For all last phonemes of words a broadcast to all first phonemes of words (including itself) is performed, via the ACU.
  |    Each first phoneme has a list of valid preceding words and corespondent word transition penalties.
  |    The word "SILENCE" is allways a valid predecessor.
  |    The assumption is made, that only one phoneme of one word resides on each PE.
  |
  | History :
  |   25.Oct  91  Tilo   created
  |   16.Nov  91  Tilo   major changes
  |   18.Nov  91  Tilo   minor changes
  |
  +============================================================================================================================*/
plural
void word_transitions()
{
    register        int                 pe;
                    int                 max, i;                                 /* for loops, that copy DTWatoms */
                    StateRefT           s;
                    IdxT                WMsrc    = 0,                           /* WM stands for  Word Merging   */
                                        WMdest   = 1;
    register        IdxT                n, temp;
                    StateRefT           RelNegState;
                    IdxT                StatesInThisModel;
    register plural ScoreT              p_CumScore;             /* just for copying */
    register plural WordRefT            p_WordX;                /* just for copying */
    register plural boolean             isActive;
                    WordRefT            SourceWord, HypoLen;
                    NbestListT          ACUstatesBuffer[MAX_StatesPerModel];
                    DTWatomT           *ACUatomP;
             plural DTWatomT           *p_ACUatomP;
             plural DTWatomT  * plural atom1P;
             plural DTWatomT  * plural atom2P;

    plural TransT       * plural   T0;
    plural TransT       * plural   T1;
    plural StateRefT               to0, to1;
    plural ScoreT                  p0,  p1;
    plural DTWatomT     * plural   dest;
    plural DTWatomT     * plural   src0;
    plural DTWatomT     * plural   src1;


 all { if (LastPhoneme) then              /* then... step by step grap one PE out of the active set and broadcast it's data */
       { isActive = TRUE;                 /*    O(words) Communications (loop approx. 400 times executed) */
         while (isActive)
           if ((pe=selectOne()) == iproc) then
             { isActive = FALSE;

               SourceWord = proc[pe].Word;     /* copy index of SourceWord to the ACU */  /* Spezialfall SILENCE beachten */
               StatesInThisModel = proc[pe].StatesInModel;
/*
               printf("\n distributing word %d, from PE %d\n", SourceWord, pe);
 */
             /*-------------------------------------------------------------------------------------------------------------
```

```
| for all states  of this one "LastPhoneme" PE  that may be needed by following Phonemes ...
+-----------------------------------------------------------------------------------------------------------------------------------------

for (s=0; s<StatesInThisModel; s++)              /* just  ONE  PE is active */
  { RelNegState = s - StatesInThisModel;         /* calculate the relative negative index */
/*
    printf("\n rel neg state is : %d\n", RelNegState);
 */
    if (CanBeAPrecState[RelNegState]) then /* Can the |RelNegState|-last state, be a final state ? */
      {
        printf("\n from PE-state %d to ACU-state %d : ",  s, MAX_StatesPerModel+RelNegState);
        /*-------------------------------------------------------------------------------------------------------------------------------
        | copy the N-best list of this "final" state to an ACU temp variable.
        +-----------------------------------------------------------------------------------------------------------------------------*/
        p_ACUatomP =  DTWcolumn[DTWsrc][MAX_PrecStates+ s];                    /*eg.: state 1(2-state-model)*/
        ACUatomP = ACUstatesBuffer[ MAX_StatesPerModel + RelNegState ];        /*eg.: state 5(6-state-model)*/

        for (n=0; n<N; n++, p_ACUatomP++, ACUatomP++)           /* for all N-best lists */
          {
            p_CumScore = (p_ACUatomP->CumScore);
            ACUatomP->CumScore =  proc[pe].p_CumScore;

            printf("%f ", ACUatomP->CumScore);

            p_WordX = p_ACUatomP->Hypothesis[0];
            max =  proc[pe].p_WordX;

            for(i=0; i<=max; i++)                                              /* sequential loop */
               ACUatomP->Hypothesis[i] = proc[pe].p_ACUatomP->Hypothesis[i];   /* copy word indices and
                                                                                  length of hypothesis */
          }

        printf("\n");

        /*-------------------------------------------------------------------------------------------------------------------------------
        | expand all hypotheses of this state by the index of the SourceWord
        | don't expand the hypotheses of the word SILENCE
        +-----------------------------------------------------------------------------------------------------------------------------*/
/*        if ((SourceWord != SILENCE) AND (CumScore < BIG_NUMBER)) then */
          {
            ACUatomP = ACUstatesBuffer[MAX_StatesPerModel + RelNegState];
            for (n=0; n<N; n++, ACUatomP++)              /* Spezialfall (CumScore < BIG_NUMBER) */
              {
                HypoLen = ++(ACUatomP->Hypothesis[0]);
                ACUatomP->Hypothesis[HypoLen] = SourceWord;
              }
          }
      }
  }

/*-----------------------------------------------------------------------------------------------------------------------------------
| Now, all N-best lists of possible final states are stored in the ACU.
| select all PEs that have a first phoneme and broadcast the ACU temp variable to them :
+-----------------------------------------------------------------------------------------------------------------------------------*/

all if (FirstPhoneme) then
    {
      /*-------------------------------------------------------------------------------------------------------------------------------
      | copy the needed states of the SourceWord to the PE's that have a FirstPhoneme.
      | The column DTWdest is used as a temporal storage.
      +-----------------------------------------------------------------------------------------------------------------------------*/

      for (s= -1; s>-MAX_PrecStates; s--)                    /* for all possible, preceeding states : */
        if (CanBeAPrecState[s]) then
          {
            /*---------------------------------------------------------------------------------------------------------------------
            | copy the Nbest List (use the DTWdest column as a temporary storage) :
            +---------------------------------------------------------------------------------------------------------------------*/
            temp = MAX_StatesPerModel + s;

            ACUatomP = ACUstatesBuffer[temp];
            atom1P   = DTWcolumn[DTWdest][MAX_PrecStates+ temp];

            for (n=0; n<N; n++, ACUatomP++, atom1P++)
              {
                atom1P->CumScore = ACUatomP->CumScore;
                max = ACUatomP->Hypothesis[0];

                for(i=0; i<=max; i++)                                  /* sequential loop */
                  atom1P->Hypothesis[i] = ACUatomP->Hypothesis[i];      /* copy word indices and
                                                                          length of hypothesis */
              }
            /*---------------------------------------------------------------------------------------------------------------------
            | now, the PEs temporal word-transition N-best list has to be merged with the new list,
            | including the coresponding, PE local, word-transition penalty for the SourceWord.
            +---------------------------------------------------------------------------------------------------------------------*/
            /* the second list takes "get_WordPenalty(SourceWord)" as a penalty, if grammar is on  */

            merge2_NbestLists(N,
                        (plural DTWatomT * plural) DTWcolumn[WMsrc][MAX_PrecStates+ s],
                          0.0,
                        (plural DTWatomT * plural) DTWcolumn[DTWdest][MAX_PrecStates+ temp],
                         WORD_TRANS_PENALTY,
                        (plural DTWatomT * plural) DTWcolumn[WMdest][MAX_PrecStates+ s]  );

            ToggleIndices(WMsrc, WMdest);
```

```
                    }
                }
            }
          /*-------------------------------------------------------------------------------------------------------------------
           | Now all LastPhoneme - PEs are active again.
           +------------------------------------------------------------------------------------------------------------------*/
        }
      /*---------------------------------------------------------------------------------------------------------------------
       | Now ALL PEs are active again.
       | The source column WMsrc now contains the result of the temporary merged word-transition N-best lists.   :-)
       | If this column isn't the same as the DTW source column, move these word-transition N-best lists over ...
       +--------------------------------------------------------------------------------------------------------------------*/

       if (WMsrc != DTWsrc) then
        {
/*
         printf("/n have to copy ... \n");
 */
         for (s= -1; s>-MAX_PrecStates; s--)                    /* for all possible, preceeding states : */
           if (CanBeAPrecState[s]) then
             /*-----------------------------------------------------------------------------------------------------------
              | copy the NbestLists :
              +----------------------------------------------------------------------------------------------------------*/
              atom1P = DTWcolumn[WMsrc][MAX_PrecStates+ s];
              atom2P = DTWcolumn[DTWsrc][MAX_PrecStates+ s];

              for (n=0; n<N; n++, atom1P++, atom2P++)
                 pp_CopyDTWatom (atom1P, atom2P);
      }
    }
}

/*#########################################################################################################################
 #
 #      PUBLIC ROUTINES
 #
 #########################################################################################################################*/

/*=========================================================================================================================
 |
 | init_DTW                          PUBLIC   SINGULAR
 |
 | Description :
 |    called once, after vacabluary was read in.  reinit_DTW() has to be called prior to each dtw step.
 |    initializes the DTW (matrices, temp-variables, sentence hypothesis, ...)
 |
 |    gathers information about the previous phoneme :
 |     - which transitions to the previous phoneme are made
 |     - how many states the pervious phoneme has.
 |
 | Globals initialized :
 |    NrOfPrecStates, IsPrecState, CanBeAPrecState, IsInitialState, DTWcolumn-Pointer
 |    ChunkSize, ChunkTime, NrOfNets
 |
 | Implementation Note :
 |
 | History :
 |   11.Nov  91  Tilo   created
 |    9.Dec  91  Tilo   finished initialization
 +========================================================================================================================*/

void init_DTW(FE_NrOfModels, FE_NrOfPhonemes, FE_NrOfNets, Predecessors,
              AdrModels, AdrDict, AdrVariant, AdrPhoneme, AdrFirst, AdrLast)

            card       FE_NrOfModels, FE_NrOfPhonemes, FE_NrOfNets,  Predecessors;
            address    AdrModels, AdrDict, AdrVariant, AdrPhoneme, AdrFirst, AdrLast;
{
     plural IdxT       s, t;
     plural StateRefT  toState;
     plural boolean   *p;
            boolean   *q;
            char       machine[40];

  NrOfModels   = FE_NrOfModels;
  NrOfPhonemes = FE_NrOfPhonemes;
  NrOfNets     = FE_NrOfNets;                                  /* initialize global variable */
  MaxPrecWords = Predecessors;

  Adr_fe_modelXB       = AdrModels;                            /* holds front end address of array of ModelRefT */
  Adr_fe_dictXB        = AdrDict;                              /* holds front end address of array of WordRefT  */
  Adr_fe_variantXB     = AdrVariant;                           /* holds front end address of array of VariantRefT  */
  Adr_fe_phonemeXB     = AdrPhoneme;                           /* holds front end address of array of PhonemeRefT */
  Adr_fe_firstPhonemeB = AdrFirst;                             /* holds front end address of array of boolean  */
  Adr_fe_lastPhonemeB  = AdrLast;                              /* holds front end address of array of boolean  */

  printf("  Checking type of MasPar front end ... ");          /* front end and back end have allways the same type */
  check_machine_type(machine);                                 /* check, whether the program was compiled right     */
  printf("it's a %s front end !\n\n", machine);

  SizeOfFrame = sizeof(ScoreT) * NrOfNets;

  showSizes();

  /* allocate and init ACU memory and PE memory, load models and phoneme information from front end into ACU : */
```

```
  initACUmem( NrOfModels, NrOfPhonemes, NrOfNets,
              &ModelBP, &NetsPerPhonemeBP, &ChunkSize, &ChunkTime, &ScoreBP );  /* loads models and phoneme info */
  initPEmem( MaxPrecWords, &PrecWordsBP );                                      /* loads PE variables from front end */

  distributeModels( NrOfModels, ModelBP );                                      /* distributes models from ACU to PEs */

 /*-------------------------------------------------------------------------------------------------------------------------------
  | initialize SEQUENTIAL variables :
  |    - pos. indices are in the actual phoneme.
  |    - neg. indices are in the previous phoneme.
  |    - state 6 is a kludge - used for invalid transitions.
  +------------------------------------------------------------------------------------------------------------------------------*/

  IsPrecState     = &(ParPrecStateArray[MAX_StatesPerModel]);   /* these two arrays can now be indexed by negative numbers */
  CanBeAPrecState = &(SeqPrecStateArray[MAX_StatesPerModel]);   /* (where -1 means last state of previous phoneme ...)      */
                                                                /* valid indices are  -1 ... -MAX_StatesPerModel  for both */
  /* initialize PARALLEL variables : */

  all { /* for all phonemes, look how many states the previous phoneme has : */

        NrOfPrecStates = xnetW[1].StatesInModel;                /* XNet wrap-around doesn't matter here ! */

       /* some initialization : */

        for(s=0; s<MAX_StatesPerModel; s++)                     /* for all states ... */
          {
            ParPrecStateArray[s] = FALSE;                       /* mark all states as not initial                          */
            SeqPrecStateArray[s] = FALSE;                       /* later we don't want to copy every states N-best list ! */
            IsInitialState[s]    = FALSE;                       /* mark transitions to previous phoneme as not used        */
          }

       /*-------------------------------------------------------------------------------------------------------------------------
        | In all phonemes, we have to know to how many of the final states of the previous phoneme we have a transition.
        | The array  IsPrecState     is initialized, so we know which preceding states we might have to copy (later)
        | The array  IsInitialState  is initialized, so we know which states are initial :-) (for reinit_DTW() )
        +------------------------------------------------------------------------------------------------------------------------*/

        for(s=0; s<StatesInModel; s++)                          /* for all states in this phoneme ... */
          for(t=0; t<MAX_TransPerState; t++)                     /* for all transitions to this state  */
            {
              toState = TransTab[s][t].toState;
              if (toState < 0)                                  /* if this is a transition to the previous phoneme */
                then {
                       IsInitialState [s] = TRUE;               /* the source state is an initial state */
                       IsPrecState [toState] = TRUE;            /* mark the transition as used !   */
                     }
            }

       /*-------------------------------------------------------------------------------------------------------------------------
        | Now we "global or" the information about the transitions to previous Phonemes.
        | The array  CanBeAPrecState  is initialized, so we know which preceding states we *really* have to copy ! (later)
        | (so we can skip those, that are never used by any following phoneme)
        +------------------------------------------------------------------------------------------------------------------------*/

        q = SeqPrecStateArray;                                  /* IsPrecState[-MAX_StatesPerModel]     */
        p = ParPrecStateArray;                                  /* CanBeAPrecState[-MAX_StatesPerModel] */
        for(s=0; s<MAX_StatesPerModel; s++)                     /* find out, which states are "final" in any PE */
          *q++ = globalor( *p++ );                              /* CanBeAPrecState gets initialized this way !  */
      }
}

/*==============================================================================================================================
  |
  | dp_sentence                   PRIVATE   SINGULAR
  |
  | Description :
  |    performes the DTW of one spoken sentence (simliar function as "dp_connected" in Joe Tebelskis sources)
  |
  |    The scores are read from the ScoresFile in this routine, the result is copied to the front end
  |    (only the front end knows the ASCII representation of the words)
  |
  | Parameters :  ScoresFile
  |
  | Globals used :
  |
  | History :
  |   28.Oct  91  Tilo   created
  |   14.Nov  91  Tilo   changed
  |    7.Dec  91  Tilo   ...
  +============================================================================================================================*/

void dp_sentence()
{
          FILE                * fd;                             /* buffered file io */
          card                  Duration;                       /* length of the utterance */
          card                  TimeSpanRead,
                                RemainingTimeSpan;
          int                   s;
    plural TransT       * plural   T0;
    plural TransT       * plural   T1;
    plural TransT       * plural   T2;
    plural StateRefT               to0, to1, to2;
    plural ScoreT                  p0,  p1,  p2;
    plural DTWatomT     * plural   dest;
    plural DTWatomT     * plural   src0;
    plural DTWatomT     * plural   src1;
```

```
    plural DTWatomT    * plural   src2;

  /* initialize some variables */

  CumTime = RelTime = 0;
  DTWsrc = 0; DTWdest = 1;

  RemainingTimeSpan = Duration = openScores(SCORES_FILENAME,  SizeOfFrame, &fdP);

  TimeSpanRead = readScores(fdP, SizeOfFrame, NrOfNets, ChunkTime);            /* changes actScoreP */
  RemainingTimeSpan -= TimeSpanRead;

  reinit_DTW();                                        /* maps NN scores for RelTime 0 to DTW input */

 /*-------------------------------------------------------------------------------------------------------------------------------
  | we don't need phoneme-to-phoneme and word-to-word transitions at the beginning of the DTW
  +-------------------------------------------------------------------------------------------------------------------------------*/

 /*-------------------------------------------------------------------------------------------------------------------------------
  | do the inner-word DTW (including the phoneme-to-phoneme and word-to-word transitions if available) :
  +-------------------------------------------------------------------------------------------------------------------------------*/

  for(s=0; s<StatesInModel; s++)                          /* for all states in this phoneme ... */
    {
      T0 = &(TransTab[s][0]);
      T1 = &(TransTab[s][1]);
      T2 = &(TransTab[s][2]);

      to0 = T0->toState; p0 = T0->penalty;
      to1 = T1->toState; p1 = T1->penalty;
      to2 = T2->toState; p2 = T2->penalty;

      src0 =  DTWcolumn[DTWsrc][MAX_PrecStates+ to0];
      src1 =  DTWcolumn[DTWsrc][MAX_PrecStates+ to1];
      src2 =  DTWcolumn[DTWsrc][MAX_PrecStates+ to2];
      dest =  DTWcolumn[DTWdest][MAX_PrecStates+  s];
/*
      printf("\n");

      printf(" to0 = %d ; to1 = %d ; to2 = %d ; s = %d\n",            proc[PE].to0, proc[PE].to1, proc[PE].to2 , s );
      printf(" DTWsrc = %d ; DTWdest = %d ; MAX_PrecStates = %d\n\n",  DTWsrc, DTWdest,  MAX_PrecStates );

      printf(" address of DTWcolumn[DTWsrc][MAX_PrecStates+ to0] : %8d\n", proc[PE].src0);
      printf(" address of DTWcolumn[DTWsrc][MAX_PrecStates+ to1] : %8d\n", proc[PE].src1);
      printf(" address of DTWcolumn[DTWsrc][MAX_PrecStates+ to2] : %8d\n", proc[PE].src2);
      printf(" address of DTWcolumn[DTWdest][MAX_PrecStates+  s] : %8d\n", proc[PE].dest);
 */
      merge3_NbestLists (N, src0,p0, src1,p1, src2,p2, dest );
/*
      printf("\n everything merged :-) \n");
 */
    }
 /*-------------------------------------------------------------------------------------------------------------------------------
  | map the scores and add the score for the actual dtw-column to the results :
  +-------------------------------------------------------------------------------------------------------------------------------*/
  map_and_add_Scores(RelTime, DTWdest, NrOfPhonemes, NrOfNets, N, NetsPerPhonemeBP);

  ToggleIndices(DTWsrc, DTWdest);
  RelTime++; CumTime++;                                  /* we just finished calculations for CumTime == 0 */

  while (CumTime < Duration)                             /* while  whole utterance not finished yet */
    {
/*    printf("\n CumTime = %d ; Duration = %d\n", CumTime, Duration );
 */
      while (RelTime < TimeSpanRead)                     /* while  chunk of scores not finished yet */
        {
/*        printf("\n RelTime = %d ; TimeSpanRead = %d\n", RelTime, TimeSpanRead );
 */
          word_transitions();                            /* prepare word-to-word transitions
                                                          * (and add word transition penalties)
                                                          */
          phoneme_transitions();                         /* prepare phoneme-to-phoneme transitions
                                                          * (and add phoneme transition penalties)
                                                          */
         /*-------------------------------------------------------------------------------------------------------------------------
          | do the inner-word DTW (including the phoneme-to-phoneme and word-to-word transitions if available)
          +-------------------------------------------------------------------------------------------------------------------------*/

          for(s=0; s<StatesInModel; s++)                   /* for all states in this phoneme ... */
            {
              T0 = &(TransTab[s][0]);
              T1 = &(TransTab[s][1]);
              T2 = &(TransTab[s][2]);

              to0 = T0->toState; p0 = T0->penalty;
              to1 = T1->toState; p1 = T1->penalty;
              to2 = T2->toState; p2 = T2->penalty;

              src0 =  DTWcolumn[DTWsrc][MAX_PrecStates+ to0];
              src1 =  DTWcolumn[DTWsrc][MAX_PrecStates+ to1];
              src2 =  DTWcolumn[DTWsrc][MAX_PrecStates+ to2];
              dest =  DTWcolumn[DTWdest][MAX_PrecStates+  s];
/*
      printf("\n");

      printf(" to0 = %d ; to1 = %d ; to2 = %d ; s = %d\n",            proc[PE].to0, proc[PE].to1, proc[PE].to2 , s );
```

```
      printf(" DTWsrc = %d ; DTWdest = %d ; MAX_PrecStates = %d\n\n",   DTWsrc, DTWdest,  MAX_PrecStates );

      printf(" address of DTWcolumn[DTWsrc][MAX_PrecStates+ to0] : %8d\n", proc[PE].src0);
      printf(" address of DTWcolumn[DTWsrc][MAX_PrecStates+ to1] : %8d\n", proc[PE].src1);
      printf(" address of DTWcolumn[DTWsrc][MAX_PrecStates+ to2] : %8d\n", proc[PE].src2);
      printf(" address of DTWcolumn[DTWdest][MAX_PrecStates+  s] : %8d\n", proc[PE].dest);
 */
              merge3_NbestLists (N, src0,p0, src1,p1, src2,p2, dest );

/*    printf("\n everything merged :-) \n");
 */
            }

          /*-------------------------------------------------------------------------------------------------------------------------
           | map the scores and add the score for the actual dtw-column to the results :
           +-------------------------------------------------------------------------------------------------------------------------*/
           map_and_add_Scores(RelTime, DTWdest, NrOfPhonemes, NrOfNets, N, NetsPerPhonemeBP);

           ToggleIndices(DTWsrc, DTWdest);
           RelTime++; CumTime++;

        }
/*
printf("\n CumTime = %d ; Duration = %d\n", CumTime, Duration );
printf("\n RelTime = %d ; TimeSpanRead = %d\n", RelTime, TimeSpanRead );
 */
      if (CumTime < Duration)
         then { TimeSpanRead = readScores(fdP, SizeOfFrame, NrOfNets, ChunkTime);            /* changes actScoreP */
                RemainingTimeSpan -= TimeSpanRead;
              }
      RelTime = 0;
    }
  closeScores(fdP);

  printf("finished ParDP !\n");

  FinalNbestMerge();

}
/*-------------------------------------------------------------------------------------------------------------------------*/
```

BAWLfiles_fe.h

```
/*****************************************************************************************************************************
; BAWLfiles_fe.h        derived from the BAWL files : bawl.h   and   network.h
;
; To be fully compatible with BAWL, the following constants, types, datastructures and variables are used.
;
; This source code is from Joe Tebelskis.
;*****************************************************************************************************************************

#ifndef _BAWL_H_
#define _BAWL_H_

#include <stdio.h>
#include <string.h>
#include <math.h>
#include "ts_std.h"

/*---------------------------------------------------------------------------------------------------------------------------
;  Common types.  These are capitalized to make them easy to distinguish from variables during initialization.
;--------------------------------------------------------------------------------------------------------------------------*/

#define INT int
#define FLOAT float
#define CHAR char
#define BOOLEAN char    /* int */
#define REG register
#define PTR int*
#define VOID void    /* int */

/*---------------------------------------------------------------------------------------------------------------------------
;  Common constants:
;--------------------------------------------------------------------------------------------------------------------------*/

#ifndef SILENCE                         /* I leave the following definition in here, although it is in "dp.h" now (Tilo) */
#define SILENCE    0 /* index of 'silence' (in phonemeA, dictA, and vocabPA) */
#endif

#define SILENCE_DUR 4 /* number of frames of leading/trailing silence around a sentence */
#define OPTIONAL 0 /* success of an operation is optional */
#define REQUIRED 1 /* success of an operation is required */

#define MAX_VOCAB    1000 /* max number of words in vocabulary */
#define MAX_DICT 700 /* max number of unique words in dictionary */
#define MAX_PHONEMES 200 /* max number of recognized phonemes in FFT files, before they are rewritten */
#define MAX_MODELS 60 /* max number of different phoneme models, typically 2 */
#define MAX_BATCHES 1 /* max number of recording batches */
#define MAX_DIALOGSperBATCH 1 /* max number of dialogs per recording batch */
#define MAX_SENTSperDIALOG 1 /* max number of sentences per dialog */
#define MAX_WORDSperSENT 200 /* max number of words per sentence */
#define MAX_CHARSperPHON        5
#define MAX_CHARSperWORD 50 /* max number of characters per word */
#define MAX_PHONSperWORD 50 /* max number of phonemes per word, after rewrites */
#define MAX_STATESperMODEL 7 /* max number of states per phoneme model */  5
#define MAX_TRANSperSTATE 3 /* max number of transitions per state */
#define MAX_FRAMESperSENT 1000 /* max number of 10 msec frames in a sentence sample 1000 nessesary for german */
#define MAX_STATESperPHON MAX_STATESperMODEL /* synonyms... */
#define MAX_STATESperWORD MAX_STATESperPHON * MAX_PHONSperWORD
#define MAX_PHONESperWORD MAX_PHONSperWORD /* (before rewrites) */
#define MAX_DIALOGS MAX_BATCHES * MAX_DIALOGSperBATCH
#define MAX_SENTENCES MAX_DIALOGS * MAX_SENTSperDIALOG

/*---------------------------------------------------------------------------------------------------------------------------
;  Bit mapped constants for the verbosity/debugging flag:
;--------------------------------------------------------------------------------------------------------------------------*/

#define MEMORY 32 /* show dynamic memory allocation? */

/*---------------------------------------------------------------------------------------------------------------------------
;  Network header file, required for STATE and MODEL type definitions, below:
;---------------------------------------------------------------------------------------------------------------------------
; originally from :
;
; network.h:  Header file describing the topology of the network(s) being used in the BAWL system.
;       This version is for Predictive networks.  There will be an alternate version for TDNNs.
;
; History:
;   June 11 1991    jmt    Created, based on "lpnn.h" and Patrick Haffner's "types.h".
;--------------------------------------------------------------------------------------------------------------------------*/

/*---------------------------------------------------------------------------------------------------------------------------
;  Common constants:
;--------------------------------------------------------------------------------------------------------------------------*/

#define MAX_LAYERSperNET 3 /* max number of layers of units per network */
#define MAX_UNITSperNET 95 /* max number of units per network */
#define MAX_INperNET 65 /* max number of input units per network */
#define MAX_OUTperNET 16 /* max number of output units per network */
#define MAX_CONNSperNET 1000 /* max number of connections per network */
#define MAX_INFRAMES 10 /* max number of contextual input frames for prediction/interpol. */
#define MAX_OUTCONNSperUNIT MAX_UNITSperNET /* max number of output connections per unit */

#define MAX_NETSperMODEL 4 /* max number of networks per phoneme model */
#define MAX_NETSperPHONEME MAX_NETSperMODEL /* synonym */
```

```
/*-----------------------------------------------------------------------------------------------------------------------------
;  Data structures describing the topology of a network:
;-----------------------------------------------------------------------------------------------------------------------------*/

typedef struct LAYER { /* layer type: */
    int unitXI; /*     this layer's initial unit index */
    int unitXF; /*     this layer's final unit index */
} LAYER;

/*-----------------------------------------------------------------------------------------------------------------------------
;  The following types define the implementation of the STATE and MODEL types:
;-----------------------------------------------------------------------------------------------------------------------------*/

typedef struct STATE_IMPL { /* state implementation type: */
    INT netX; /*     net index.  (Note: phonemeX is part of the STATE type.) */
} STATE_IMPL;

typedef struct MODEL_IMPL { /* model implementation type: */
    INT netN; /*     number of networks in this phoneme model */
} MODEL_IMPL;

/*-----------------------------------------------------------------------------------------------------------------------------
;  The following are used for DP search.  Note that with predictive networks we want to minimize the cumulative DP score.
;-----------------------------------------------------------------------------------------------------------------------------*/

#define better(score1, score2) (score1 < score2) /* is score1 "better" than score2? */
#define INITCUM 9999.0 /* initial value: worst possible cumulative DP score */
#define GOOD_PHONTARGET 0.0 /* best phoneme score (prediction error) = 0.0 */
#define BAD_PHONTARGET coefN * 1.0 /* worst phoneme score (prediction error) = 16.0 */

/*-----------------------------------------------------------------------------------------------------------------------------
;  End of included file :  network.h
;-----------------------------------------------------------------------------------------------------------------------------*/

/*-----------------------------------------------------------------------------------------------------------------------------
;  File names:
;-----------------------------------------------------------------------------------------------------------------------------*/

#define MAX_FILENAMELEN    100 /* max length of a filename */

CHAR    modelfile [MAX_FILENAMELEN]; /* name of model file */

#define SENTFILE 's' /* file format = sentences */
#define WORDFILE 'w' /* file format = words */
#define DICTFILE 'd' /* file format = dictionary */

/*-----------------------------------------------------------------------------------------------------------------------------
;  Stuff for Memory Management:
;-----------------------------------------------------------------------------------------------------------------------------*/

#define MAX_DYNAMIC_BYTES 200000 /* max bytes reserved for dynamic memory allocation */
#define MAX_DYNAMIC_INTS MAX_DYNAMIC_BYTES/sizeof(INT) /* max integers reserved for dynamic memory allocation */
INT dynamicA [MAX_DYNAMIC_INTS]; /* reserved for dynamic memory allocation */
INT dynamicN; /* number of dynamic INTs allocated so far */

#define allocT(type) /* allocate a <type> */ \
      (type *) allocCA (sizeof(type), "1 type")
#define allocT_copy(type,sourceA) /* allcoate a <type>, then copy from <sourceA> */ \
      (type *) allocCA_copy (sizeof(type), sourceA, "1 type")
#define allocTA(type, num)  /* allocate an array of <num> <types> */ \
      (type *) allocCA ((num)*sizeof(type), "num type")
#define allocTA_copy(type,num,sourceA)  /* allocate an array of <num> <types>, then copy from <sourceA>. */ \
      (type *) allocCA_copy ((num)*sizeof(type), sourceA, "num type")

/*-----------------------------------------------------------------------------------------------------------------------------
;  Data structures related to phoneme models:
;-----------------------------------------------------------------------------------------------------------------------------*/

typedef struct TRANS { /* transition type: */
    INT    deltaState; /* how to get to next state from here;  to-from */
    FLOAT  penalty; /* transition penalty */
} TRANS;

typedef struct STATE { /* state type: */
    INT    phonemeX; /* identify corresponding phonemeX (for duration, diagnostics) */
    INT    pStateX; /* identify corresponding stateX in phoneme (for diagnostics) */
    FLOAT* scoreA; /* scoreA [sFrameX] = state's score at each frame */
    INT    transN; /* number of transitions from this state */
    TRANS* transA; /*      list of transitions from this state */
    STATE_IMPL  impl; /* implementation of this state: see network.h */
} STATE;

typedef struct MODEL { /* phoneme model type (typically 2 canonical, 40 instantiated): */
    CHAR*   nameP; /*  name of phoneme model, eg: "6-state" */
    INT     stateN; /* number of states in this model */
    STATE*  stateA; /* all states in the model */
    MODEL_IMPL  impl; /* implementation of this model: see network.h */
} MODEL;
MODEL modelA [MAX_MODELS]; /* all canonical phoneme models */
INT modelN; /* number of canonical phoneme models */

typedef struct PHONEME { /* phoneme type: */
    CHAR   spelling [MAX_CHARSperPHON]; /*     spelling of the phoneme, eg: "sh" */
    CHAR   char1; /*     1-char symbol for the phoneme, eg: "S" */
    INT    minDur; /*     minimum allowed duration of this phoneme */
    INT    maxDur; /*     maximum allowed duration of this phoneme */
```

```
    MODEL* modelP; /*     pointer to the instantiated model for this phoneme */
} PHONEME;
PHONEME phonemeA [MAX_PHONEMES]; /* all phonemes */
INT phonemeN; /* number of phonemes being modeled */

/*-------------------------------------------------------------------------------------------------------------------------------
;  Data structures related to the dictionary:
;------------------------------------------------------------------------------------------------------------------------------*/

typedef struct VARIANT { /* variant pronunciation type: */
    INT     dictX; /*    dictionary index */
    INT     vocabX; /*    vocabulary index */
    INT     phonN; /*    number of phonemes in phonetic spelling (after rewrites) */
    INT*    phonA; /*    phonetic spelling, in terms of model indices */
    INT     stateN; /*    number of states in this word */
    STATE** statePA; /*    array of state-pointers for this word */
    struct VARIANT* nextP; /*    pointer to next variant pronunciation of dictX */
} VARIANT;
typedef VARIANT VOCAB; /* each VOCAB entry is a VARIANT */

typedef struct DICT { /* dictionary type (all pronunciations of all known words): */
    CHAR*   spellP; /*    lexical spelling of the word */
    BOOLEAN used; /*    is this word used in the current vocabulary? */
    INT     minDur; /*    minimum allowed duration of word */
    INT     maxDur; /*    maximum allowed duration of word */
    FLOAT   avgDur; /*    average duration of this word (= labelTotalDur/labelCount) */
    INT     labelCount; /*    number of times this word occurred in labelfile */
    INT     labelTotalDur; /*    total duration of all labeled instances of this word */
    VARIANT* variantPI; /*    linked list of all variant pronunciations of this word */
} DICT;
DICT dictA [MAX_DICT]; /* the dictionary of all known words */
INT dictN; /* number of unique words in the dictionary */

VOCAB* silenceP; /* SILENCE is a special word */

/*-------------------------------------------------------------------------------------------------------------------------------
;  Data structures related to the rewrite rules:
;------------------------------------------------------------------------------------------------------------------------------*/

#define MAX_REWRITES 100 /* max number of rewrite rules */
#define MAX_LHS 5 /* max number of tokens on left hand side of a rewrite rule */
#define MAX_RHS 5 /* max number of tokens on right hand side of a rewrite rule */

typedef struct PSPELL { /* phoneme spelling type: */
    CHAR pSpell [MAX_CHARSperPHON];
} PSPELL;

typedef struct REWRITE { /* rewrite rule type: */
    INT    lhsN; /*    number of tokens on left hand side */
    INT    rhsN; /*    number of tokens on right hand side */
    PSPELL lhsA [MAX_LHS]; /*    left hand side tokens */
    PSPELL rhsA [MAX_RHS]; /*    right hand side tokens */
} REWRITE;
REWRITE  rewriteA [MAX_REWRITES]; /* the rewrite rules */
INT      rewriteN;   /* number of rewrite rules */


/*-------------------------------------------------------------------------------------------------------------------------------
;  Data structures related to language modeling (bigrams):
;------------------------------------------------------------------------------------------------------------------------------*/

#define NO_GRAMMAR 'n' /* use no grammar */
#define WORDPAIRS 'w' /* use word pair grammar */
#define BIGRAMS 'b' /* use bigram grammar */

typedef char GRAMMAR; /* grammar type = character */

typedef struct BIGRAM { /* bigram type: */
    INT    succN; /*     number of legal successors for this word */
    INT*   succDictXA; /*     array of legal successors for this word */
    FLOAT* succBiasA; /*     array of probability biases for those successors */
} BIGRAM;
BIGRAM  bigramA [MAX_DICT];
FLOAT z1; /* bias for bigram constraint: see biasBigram */

/*-------------------------------------------------------------------------------------------------------------------------------
;  Miscellaneous variables:
;------------------------------------------------------------------------------------------------------------------------------*/

INT     networkN;                                   /* overall number of neural nets (from model file) */

INT verbose; /* verbosity flag */

/*-------------------------------------------------------------------------------------------------------------------------------
;      ROUTINES
;------------------------------------------------------------------------------------------------------------------------------*/

extern VOID read_model ();   /* Reads a model file, describing all the phoneme models to be built (and network info) */
extern FILE*   open_readfile ();  /* Opens a file for read access; remembers the name of the file */
extern VOID checkString ();   /* Checks a string's contents; if it has an unexpected value, aborts the program */
extern VOID read_rewrites (); /* Reads a rewrites file */
extern VOID read_dict ();     /* Reads a dictionary file */
extern VOID read_bigrams ();  /* Reads a bigram file */
extern MODEL* findModelP ();    /* Finds the model corresponding to a given model name */
extern INT findPhonemeX ();  /* Finds the phoneme index corresponding to a given phoneme spelling */
extern INT findDictX ();     /* Finds the dictionary index corresponding to a word spelling */
```

```
extern VARIANT* findVariantP ();  /* Finds the variant of a word corresponding to a phonetic spelling */
extern VOID do_rewrites ();   /* Rewrites a phonetic spelling according to the rewrite rules */

extern INT findModelX ();     /* Finds the model index corresponding to a given model pointer */

extern VOID     print_BAWL_models();   /* prints all models, that were read */
extern VOID     print_BAWL_phonemes(); /* prints all phonemes, that were read */
extern VOID     print_BAWL_words();    /* prints all words, that were read */
extern VOID     print_BAWL_bigrams();  /* prints all bigrams, and the list of words without successor */

#endif
```

BAWLfiles_fe.h

```
/*****************************************************************************************************************************
; BAWL_files_fe.c       derived from several BAWL files.
;
; To be fully compatible with BAWL, the following file io routines are used.
;
; This source code is originaly from Joe Tebelskis.
; minor changes were made, for sake of better structure. Tilo Sloboda.  (Sake ist immer gut)
;*****************************************************************************************************************************/

#include "BAWLfiles_fe.h"                              /* derived from bawl.h; now includes network.h */

/*---------------------------------------------------------------------------------------------------------------------------
;     FORWARD DEFINITION OF ALL ROUTINES
;---------------------------------------------------------------------------------------------------------------------------*/

PTR allocCA (); /* Allocates an array of characters */
PTR allocCA_copy (); /* Allocates an array of characters, then copies another array into it */
CHAR* allocS_copy (); /* Allocates a copy of a string */
VOID read_model (); /* Reads a model file, describing all the phoneme models to be built (and network info) */
FILE*   open_readfile (); /* Opens a file for read access; remembers the name of the file */
VOID readOK (); /* Checks the result of fscanf; if there was a problem, aborts the program */
VOID checkString (); /* Checks a string's contents; if it has an unexpected value, aborts the program */
VOID checkLimits (); /* Checks to make sure a variable is within its allowed limits */
VOID read_rewrites (); /* Reads a rewrites file */
VOID read_dict (); /* Reads a dictionary file */
VOID read_bigrams (); /* Reads a bigram file */
MODEL* findModelP (); /* Finds the model corresponding to a given model name */
INT findPhonemeX (); /* Finds the phoneme index corresponding to a given phoneme spelling */
INT findDictX (); /* Finds the dictionary index corresponding to a word spelling */
VARIANT* findVariantP (); /* Finds the variant of a word corresponding to a phonetic spelling */
VOID addDict (); /* Adds a variant phonetic spelling to the dictionary, if not already present */
VOID do_rewrites (); /* Rewrites a phonetic spelling according to the rewrite rules */

/*---------------------------------------------------------------------------------------------------------------------------
;     PRIVATE VARIABLE
;---------------------------------------------------------------------------------------------------------------------------*/

static char   cur_filename [80]; /* name of file currently opened for reading. */

/*#############################################################################################################################
 #
 #      PRIVATE DEBUGGING ROUTINES
 #
 #############################################################################################################################*/

/*===========================================================================================================================
; print_BAWL_bigrams
;
; Parameters : dictN, dictA, bigramsA
;
; History:
;    3.Dec 91   Tilo   Created.
;===========================================================================================================================*/

VOID  print_BAWL_bigrams (dictN, dictA, bigramA)

    INT      dictN;
    DICT     dictA[];
    BIGRAM   bigramA[];
{
    BIGRAM  *bigramP;
    INT     *succP;
    FLOAT   *biasP;
    INT      dictX, succX, succN, succWordX;

  printf("\n words without successors :\n\n");

  bigramP = bigramA;
  for(dictX=0; dictX<dictN; dictX++, bigramP++)
   { succN = bigramP->succN;
     if ( succN == 0) then
        printf(" %3d  %s\n", dictX, dictA[dictX].spellP);
   }
  printf("\n\n");

  bigramP = bigramA;

  for(dictX=0; dictX<dictN; dictX++, bigramP++)
   {
     succP = bigramP->succDictXA;
     biasP = bigramP->succBiasA;
     succN = bigramP->succN;

     for(succX=0; succX<succN; succX++)
      {
        succWordX = succP[succX];

        printf(" %30s (%3d) --- %8.6f ---> (%3d) %-30s\n",
                dictA[dictX].spellP, dictX, biasP[succX], succWordX, dictA[succWordX].spellP );
      }
   }
  printf(" done.\n");
}
```

```
/*=============================================================================================================================
; print_BAWL_phonemes
;
; Parameters : phonemeN, phonemeA, modelN, modelA
;
; History:
;    1.Dec 91   Tilo   Created.
;============================================================================================================================*/

VOID print_BAWL_phonemes(phonemeN, phonemeA, modelN, modelA)

    INT        phonemeN, modelN;
    PHONEME    phonemeA[];
    MODEL      modelA[];
{
    INT        i, modelX;
    PHONEME   *p;


 printf("\n  %d phonemes :\n\n  idx   spelling  char   model     modelP  modelX    min    max\n\n", phonemeN);

 for(i=0; i<phonemeN; i++)
   {
     p = &(phonemeA[i]);

     modelX = findModelX(modelN, modelA, p->modelP);

     printf("  %2d    %8s      %c   %8s    %6x      %2d      %4d   %4d\n",
            i, p->spelling, p->char1, p->modelP->nameP , p->modelP, modelX, p->minDur, p->maxDur );
   }
 printf("\n\n");
}

/*=============================================================================================================================
; print_BAWL_words
;
; Parameters :  dictN, dictA
;
; History:
;    1.Dec 91   Tilo   Created.
;============================================================================================================================*/

VOID print_BAWL_words(dictN, dictA, showIndices)

   INT       dictN;
   DICT      dictA[];
   BOOLEAN   showIndices;
{
   INT       d, p, variantX;
   DICT*     dictP;
   VARIANT*  variantP;

 printf("\n  %d words in the dictionary :\n\n", dictN);
 printf("  idx          spelling                variant  states  phonemes  phoneme-indices\n\n");

 for(d=0; d<dictN; d++)
   {
     dictP = &dictA[d];
     variantP = dictP->variantPI;
     variantX = 0;

     printf(" %3d   %30s", d, dictP->spellP);

     while (variantP != NIL)
       { if (variantX == 0)
            then printf(              "     %2d       %3d      %2d\t   ", variantX, variantP->stateN, variantP->phonN);
            else printf("\n\t\t\t\t\t     %2d       %3d      %2d\t   ", variantX, variantP->stateN, variantP->phonN);

         if(showIndices)
           then for (p=0; p<variantP->phonN; p++)
                   printf("%2d ", variantP->phonA[p]);

           else for (p=0; p<variantP->phonN; p++)
                   printf("%c ",  phonemeA[ variantP->phonA[p] ].char1);

         printf("\n");
         variantP = variantP->nextP; variantX++;
       }
   }
}

/*=============================================================================================================================
; print_BAWL_models
;
; Parameters :
;
;  modelN, modelA
;
; History:
;   25.Nov 91   Tilo   Created.
;============================================================================================================================*/

VOID  print_BAWL_models (modelN, modelA)

     INT    modelN;
```

```
    MODEL  modelA[];
{
    INT    m, s, t;
    MODEL* modelPtr;
    STATE* statePtr;
    TRANS* transPtr;


  for(m=0; m<modelN; m++)
    {
      modelPtr = &(modelA[m]);
      printf("\n model %d ;  name : \"%s\" ;  states : %d \n\n",   m, modelPtr->nameP, modelPtr->stateN);
      printf(" from  delta    penalty\tnetNr\n");

        for(s=0; s<modelPtr->stateN; s++)
          {
            statePtr = &(modelPtr->stateA[s]);
            for(t=0; t<statePtr->transN; t++)
              {
                transPtr = &(statePtr->transA[t]);
                printf("  %2d     %2d   %8.2f\n",  s, transPtr->deltaState, transPtr->penalty);
              }
            printf("\t\t\t %2d\n" , modelPtr->stateA[s].impl.netX);
          }
        printf("-----------------------------------------------\n");
    }
  printf("\n");
}

/*#############################################################################################################################
 #
 #      PUBLIC ROUTINES
 #
 #############################################################################################################################*/

/*=============================================================================================================================
; read_rewrites:  Reads a rewrites file, which describes how to rewrite phonemes from the dictionary and label file.
;
; Parameters:
;    IN:
; rewritefile = name of rewrite file
;   OUT:
; rewriteA  = table of rewrite rules
;      &rewriteN  = number of rewrite rules
;
; History:
;   6/23/91   jmt   Created, based on old LPNN routine.
;============================================================================================================================*/

VOID read_rewrites (rewritefile, /* in  */
                    rewriteA, rewriteNP) /* out */
CHAR    *rewritefile;
REWRITE rewriteA [];
INT     *rewriteNP;
{
  FILE  *r;
  CHAR  line [100];
  INT   lhsX, rhsX;
  CHAR  *strP;

  r = open_readfile (rewritefile); /* open rewrite file */
  printf (" Reading \"%s\" ...", rewritefile); fflush(stdout);
  *rewriteNP = 0; /* no rewrite rules initially */
  while (fgets (line, 100, r) != NULL) { /* for each line (rule) in rewritefile: */
      checkLimits (*rewriteNP+1, MAX_REWRITES, "MAX_REWRITES"); /*     check limits */
      lhsX = 0; /*     nothing in left hand side of rule initially */
      do { /*     repeat: */
          strP = strtok ((lhsX==0?line:NULL), " \t\n"); /*     tokenize line; get next token */
          if (strP == NULL) goto contin; /*    ignore blank lines */
          checkLimits (lhsX+1, MAX_LHS, "MAX_LHS"); /*    check limits */
          strcpy (&rewriteA[*rewriteNP].lhsA[lhsX++], strP); /*         copy into rewrite rule's lhs */
      } while (strneq (strP, "=>")); /*     until token is "=>" */
      rhsX = 0; /*     nothing in right hand side of rule initially */
      while ((strP = strtok (NULL, " \t\n")) != NULL) { /*     while there is another token: */
          checkLimits (rhsX+1, MAX_RHS, "MAX_RHS"); /*    check limits */
          strcpy (&rewriteA[*rewriteNP].rhsA[rhsX++], strP); /*         copy into rewrite rule's rhs */
      }
      rewriteA[*rewriteNP].lhsN = lhsX - 1; /*     remember number of lhs tokens (exclude "=>") */
      rewriteA[*rewriteNP].rhsN = rhsX; /*     remember number of rhs tokens */
      (*rewriteNP)++; /*     do next rule. */
  contin:;
  }
printf (" %d rewrite rules.  done.\n\n", *rewriteNP);
fclose (r); /* close rewrite file */
}

/*=============================================================================================================================
; read_dict:  Reads a dictionary file, which lists all known pronunciations of all known words.
;
; Parameters:
;    IN:
; dictfile = name of dictionary file
; rewriteA = table of rewrite rules
; rewriteN = number of rewrites rules
;       phonemeN = number of phonemes in the model file
;   OUT:
```

```
; dictA  = dictionary of all known pronunciations of all known words
;      &dictN   = number of dictionary entries
;
; History:
;   6/23/91   jmt   Created, based on old LPNN code.
;  28.Nov 91  Tilo  additional parameter phonemeN
;=============================================================================================================================

VOID read_dict (dictfile, rewriteA, rewriteN, phonemeN, /* in  */
                dictA, dictNP) /* out */
CHAR    *dictfile;
REWRITE rewriteA [];
INT     rewriteN, phonemeN;
DICT    dictA [];
INT     *dictNP;
{
  FILE *d; /* dictionary file pointer */
  INT variantN; /* number of variants */
  CHAR line [100]; /* line of text in the dictionary file */
  CHAR silStr [10]; /* silence string */
  INT i; /* temp */
  CHAR* strP; /* string pointer */
  CHAR* wSpellP; /* word spelling */
  PSPELL pSpellA [MAX_PHONESperWORD]; /* phoneme spellings, before rewrites */
  INT phoneN; /* number of phonemes, before rewrites */
  INT phonN; /* number of phonemes, after rewrites */
  INT     phonA [MAX_PHONSperWORD]; /* phonemes, after rewrites */
  INT dummyDurA [MAX_PHONSperWORD]; /* dummy durations -- not used */

  *dictNP = 0; /* no words in dictionary yet */
  variantN = 0; /* no variants in dictionary yet */
  phonA [0] = SILENCE; /* SILENCE is spelled: SILENCE */
  strcpy (silStr, "$");
  addDict (silStr, 1, phonA, dictNP, &variantN); /* add SILENCE to the dictionary */
  dictA [SILENCE].minDur = 0; /* SILENCE word has no duration constraints */
  dictA [SILENCE].maxDur = 9999;
  silenceP = dictA [SILENCE].variantPI; /* define the global SILENCE variant */

  d = open_readfile (dictfile); /* open dictionary file */
  printf (" Reading \"%s\" ...", dictfile);  fflush (stdout);
  while (fgets (line, 100, d) != NULL) { /* for each line in dictfile, eg: "ABLE  EY B AX L"... */
      checkLimits (*dictNP+1, MAX_DICT, "MAX_DICT"); /*     check limits */
      wSpellP = strtok (line, " \t\n"); /*     get first token, eg: "ABLE" */
      if (wSpellP == NULL) continue; /*     ignore blank lines */
      i = 0;
      while (wSpellP[i] != EOS) { /*     change punctuation, eg: "IT+S" ==> "IT'S" */
          if (wSpellP[i] == '+')  wSpellP[i] = '\'';
          i++;
      }
      phoneN = 0; /*     initially no phonemes in Arpabet spelling */
      while ((strP = strtok (NULL, " \t\n")) != NULL)  /*     while there are more Arpabet phonemes, */
          strcpy (&pSpellA [phoneN++], strP); /*         copy them into phones array */
      if (phoneN == 0) continue; /*     ignore words without Arpabet spellings */
      do_rewrites (phoneN, pSpellA, dummyDurA, phonemeN, /*     rewrite phonemes using rewrite rules: in  */
                   &phonN, phonA, dummyDurA); /*      out */
      addDict (wSpellP, phonN, phonA, dictNP, &variantN); /*     add variant to dictionary, if not already known */
  }
  printf (" %d words in dictionary (%d variants).  done.\n\n", *dictNP, variantN);
  fclose (d); /* close dictionary file */
}

/*=============================================================================================================================
; read_bigrams:  Reads a bigram file, which lists all the legal successors for each known word.
;
; Parameters:
;    IN:
;       dictN      = number of dictionary entries
; bigramfile = name of bigram file
;    OUT:
; bigramA    = bigrams.
;
; History:
;   6/23/91   jmt   Created, based on old LPNN code.
;  28.Nov 91  Tilo  dictN isn't global anymore.
;=============================================================================================================================

VOID read_bigrams (dictN, bigramfile, /* in  */
                   bigramA) /* out */
INT     dictN;
CHAR  *bigramfile;
BIGRAM bigramA [MAX_DICT];
{
#define MAX_CHARSperLINE  MAX_CHARSperWORD * MAX_DICT /* max number of characters on a line of the bigramfile */
CHAR line [MAX_CHARSperLINE]; /* buffer for a line of the bigramfile */
INT totalBigrams = 0; /* total number of bigrams */
FILE *f; /* bigram file pointer */
INT dictX; /* dictionary index */

for (dictX=0; dictX<dictN; dictX++) /* initialize bigrams to Unused */
    bigramA [dictX].succN = 0;

  f = open_readfile (bigramfile); /* open bigram file */
  printf (" Reading \"%s\" ...", bigramfile);  fflush (stdout);
  while (fgets (line, MAX_CHARSperLINE, f) != NULL) { /* for each line in bigram file: */
      INT   baseCount, succCount, succN, succX;
      INT   succDictX; /*     dictionary index of successor word */
```

```
    INT   succDictXA [MAX_VOCAB]; /*     array of dictionary indices of successor words */
    INT   succCountA [MAX_VOCAB]; /*     array of counts for those successors */

    CHAR* strP = strtok (line, " \t\n"); /*     tokenize line; get first word */
    INT   dictX = findDictX (dictN, strP, OPTIONAL); /*     get its dictionary index */
    if (dictX == -1)  continue;
    baseCount = atoi (strP = strtok (NULL, " \t\n")); /*     get base count for this word */
    baseCount = 0; /*     naah, compute it, with exclusions.  jmt 6/19/90 */
    succN = 0;
    while ((strP = strtok (NULL, " \t\n")) != NULL) { /*     for all other words on this line: */
        succDictX = findDictX (dictN, strP, OPTIONAL); /*         get dictionary index of word = successor */
        if (succDictX == -1)  continue;
        strP = strtok (NULL, " \t\n"); /*         get count of successor */
        succCount = atoi (strP);
/*          if (dictA[succDictX].used) { /*    if this successor is in current vocabulary: */
            baseCount += succCount; /*             increase baseCount by usage */
            succDictXA [succN] = succDictX; /*        store in temp array */
            succCountA [succN] = succCount;
            if ((succN > 0) && (succDictX < succDictXA[succN-1])) /*     if bigrams are not sorted, abort. */
                {printf ("Bigram file %s is not sorted.\n", bigramfile); exit(-1);}
            succN++; /*         one more canonical successor */
  /*        }       */      /*for now, take ALL bigramms*/
    }
    bigramA [dictX].succN = succN; /*     remember number of successors */
    bigramA [dictX].succDictXA = allocTA_copy (INT, succN, succDictXA); /*     allocate & copy successors */
    bigramA [dictX].succBiasA = allocTA (FLOAT, succN); /*     allocate probability biases */
    for (succX=0; succX<succN; succX++) {
        FLOAT bigramProb = (float) succCountA[succX] / baseCount; /*     bigram probability */
        bigramA [dictX].succBiasA[succX] = -z1 * log(bigramProb); /*     convert to additive bigram bias */
    }
    totalBigrams += succN;
  }
  printf (" %d bigrams. done.\n\n", totalBigrams);
  fclose (f); /* close bigram file */
}

/*=============================================================================================================================
; read_model:  Reads a model file, which describes all the phoneme models to be built.
;
; Parameters:
;   IN:
; modelfile = name of model file
;   OUT:
; modelA   = model structure.
;      &modelN   = number of phonemes to be modeled.
; phonemeA = phoneme structure.
;      &phonemeN = number of phonemes.
;      &networkN = number of neural networks needed.
;
; History:
;   6/26/91   jmt   Created, based on old LPNN code.
;   8/28/91   jmt   AcceptsVersion 2 model file format.
;============================================================================================================================*/

VOID read_model (modelfile, /* in  */
 modelA, modelNP, phonemeA, phonemeNP, networkNP) /* out */
CHAR* modelfile;
MODEL modelA [];
INT* modelNP;
PHONEME phonemeA [];
INT* phonemeNP;
INT* networkNP;
{
INT versionX, modelX, modelN, phonemeX, phonemeN, filePhonemeN, fileNetworkN;
CHAR str [100], modelName [100];

/*----------------------------------------------------------------------------------------------------------------------------
;  The model file has two parts.  First, read all of the model descriptions.
;---------------------------------------------------------------------------------------------------------------------------*/

FILE* m = open_readfile (modelfile); /* open modelfile */
printf (" Reading \"%s\" ...", modelfile);  fflush (stdout);
readOK (fscanf (m, "%s%d%*[^\n]%*c", str, &versionX)); /* read "Version 1 model file" */
if (strneq (str, "Version")) /* prohibit old file formats */
    {printf ("  Please change this file to use the new format, or override with the -M! option.\n");  exit(-1);}
readOK (fscanf (m, "%d%s", &modelN, str)); /* read "1 model" or "2 models" */
str[5] = EOS;   /* (ignore optional "s" in "models") */
checkString (str, "model");
printf ("  %d models:\n", modelN);
if (versionX >= 2) { /* if Version 2+... */
    readOK (fscanf (m, "%d%s", &filePhonemeN, str)); /*     read "40 phonemes" */
    checkString (str, "phonemes");
    readOK (fscanf (m, "%d%s", &fileNetworkN, str)); /*     read "120 nets" */
    checkString (str, "nets");
}
*modelNP = modelN;
for (modelX=0; modelX<modelN; modelX++) { /* for each model... */
    MODEL* modelP = &modelA[modelX];
    INT    stateX, stateN, netN;
    STATE  stateA  [MAX_STATESperMODEL];
    INT    transNA [MAX_STATESperMODEL];
    TRANS  transAA [MAX_STATESperMODEL] [MAX_TRANSperSTATE];
    readOK (fscanf (m, "%*[^=]%*c%s", modelName)); /* read "-----------\n model name = 2-state" */
    modelP->nameP = allocS_copy (modelName); /* allocate & copy model name */
    readOK (fscanf (m, "%d%*s%d%*s", &stateN, &netN)); /* read "2 states, 1 net" */
    modelP->stateN = stateN;
```

```
    modelP->impl.netN = netN;
    printf ("%s\"%s\" (%d states/%d nets)",
(modelX==0?"\n ":",\n "), modelP->nameP, modelP->stateN, modelP->impl.netN);
    readOK (fscanf (m, "%*s%*s")); /* skip "state   net" */
    for (stateX=0; stateX<stateN; stateX++) { /* for each state... */
readOK (fscanf (m, "%*d%d", &stateA[stateX].impl.netX)); /* read "0    0" */
transNA [stateX] = 0; /* initialize this for later */
    }
    readOK (fscanf (m, "%*s%*s%*s")); /* skip "from   to   penalty" */
    for (;;) { /* for each transition... */
INT from, to, transN;
FLOAT penalty;
CHAR line [100];
fgets (line, 100, m); /* read next line */
if (line[0] == '\n')  continue; /* skip blank lines */
if (line[0] == '-')  break; /* if "----------", break out of loop */
readOK (sscanf (line, "%d%d%f", &from, &to, &penalty)); /* read "0    1   0.0" */
transN = (transNA [from]) ++; /* get current transN for <from> state; increment */
transAA [from][transN].deltaState = (to-from); /* store deltaState */
transAA [from][transN].penalty = penalty; /* store penalty */
    } /* end (for each transition) */
    for (stateX=0; stateX<stateN; stateX++) { /* for each state... */
STATE* stateP = &stateA [stateX];
stateP->transN = transNA [stateX]; /* store number of transitions */
stateP->transA = allocTA_copy (TRANS, transNA[stateX], transAA[stateX]);   /* ALLOCATE & COPY TRANSITIONS */
stateP->pStateX = stateX; /* identify state index */
stateP->phonemeX = -1; /* this will be instantiated later (see below) */
stateP->scoreA = NIL; /* this will be instantiated later */
    }
    modelP->stateA = allocTA_copy (STATE, stateN, stateA); /* ALLOCATE & COPY STATES */
} /* end (for each model) */

/*-------------------------------------------------------------------------------------------------------------------------------
;  We have finished reading all the model descriptions.  Now read the phonemes and their model assignments.
;-------------------------------------------------------------------------------------------------------------------------------*/

phonemeN = 0; /* no phonemes yet */
while (fscanf (m, "%s%s%s", /* while model file is not empty... */
       phonemeA[phonemeN].spelling, /* read, eg: "EH  e  6-state" */
      &phonemeA[phonemeN].char1,
       modelName) != EOF) {
    MODEL* modelP = findModelP (modelN, modelName, REQUIRED); /* find model of "6-state" */
    MODEL* newModelP = allocT_copy (MODEL, modelP); /* allocate & copy the model */
    INT    stateN = modelP->stateN; /* get number of states in model */
    STATE  stateA [MAX_STATESperPHON];
    INT    stateX, netX;
    for (stateX=0; stateX<stateN; stateX++) { /* for each state of the model... */
stateA[stateX] = modelP->stateA[stateX]; /*     make a local copy of the state */
stateA[stateX].phonemeX = phonemeN; /*     instantiate its phonemeX */
netX = stateA[stateX].impl.netX;
/* stateA[stateX].scoreA = &scoreM [phonemeN][netX][0]; /*     instantiate its scoreA, pointing to scoreM frames */
    }
    newModelP->stateA = allocTA_copy (STATE, stateN, stateA); /* allocate & copy the instantiated states into new model */
    phonemeA[phonemeN].modelP = newModelP; /* store new model */
    phonemeA[phonemeN].minDur = 9999; /* initialize phoneme duration statistics... */
    phonemeA[phonemeN].maxDur = 0;
    phonemeN ++; /* one more phoneme */
}
phonemeA [SILENCE].minDur = 0; /* SILENCE has no duration constraints... */
phonemeA [SILENCE].maxDur = 9999;
printf (".\n\n  %d phonemes.", phonemeN);
*phonemeNP = phonemeN;
fclose (m); /* close modelfile */

/*-------------------------------------------------------------------------------------------------------------------------------
;  Count how many nets there are altogether.  Check limits and agreements of values.
;-------------------------------------------------------------------------------------------------------------------------------*/

*networkNP = 0;
for (phonemeX=0; phonemeX<phonemeN; phonemeX++) /* count networks... */
    *networkNP += phonemeA[phonemeX].modelP->impl.netN;
printf ("  %d nets.\n\n ... done.\n\n", *networkNP);

checkLimits (modelN, MAX_MODELS, "MAX_MODELS");
checkLimits (phonemeN, MAX_PHONEMES, "MAX_PHONEMES");
for (modelX=0; modelX<modelN; modelX++) {
    checkLimits (modelA[modelX].stateN, MAX_STATESperMODEL, "MAX_STATESperMODEL");
    checkLimits (modelA[modelX].impl.netN, MAX_NETSperMODEL, "MAX_NETSperMODEL");
}
if (versionX >= 2) {
    if (*phonemeNP != filePhonemeN)  {printf ("Error: number of phonemes is wrong.\n");  exit(-1);}
    if (*networkNP != fileNetworkN)  {printf ("Error: number of networks is wrong.\n");  exit(-1);}
}
}

/*===============================================================================================================================
; findModelP:  Finds the model corresponding to a given model name.
;
; Parameters:
;   modelN    = number of models in the model file
;   modelName = name of the model, eg: "6 state".
;   required  = is this model required to be known?
;
; Returns:  modelP for that modelName, or NIL if not found.  If a required model is not found, aborts with an error message.
;
; History:
```

```
;   6/26/91   jmt   Created.
;  28.Nov 91  Tilo  modelN isn't global anymore.
;=============================================================================================================================

MODEL* findModelP (modelN, modelName, required)
INT   modelN;
CHAR* modelName;
INT   required;
{
INT modelX;
for (modelX=0; modelX<modelN; modelX++)
    if (streq (modelName, modelA[modelX].nameP))
        return (&modelA[modelX]);
if (required) {
    printf ("Modelfile %s does not recognize model name %s.\n", modelfile, modelName);
    exit (-1);
} else return (NIL);
}

/*============================================================================================================================
; findModelX:  Finds the model index corresponding to a given model pointer.
;
; Parameters:
;   modelN    = number of models in the model file
;   modelA    = the array of models.
;   modelP    = pointer to the model.
;
; Returns:  modelX for that modelPointer, or -1 if not found.
;
; History:
;   1.Dec 91  Tilo  created
;============================================================================================================================

INT  findModelX (modelN, modelA, modelP)
   INT    modelN;
   MODEL  modelA[];
   MODEL* modelP;
{
   INT    modelX;

 for (modelX=0; modelX<modelN; modelX++)
   if (streq (modelP->nameP, modelA[modelX].nameP))
      then return (modelX);

 return (-1);
}


/*============================================================================================================================
; findPhonemeX:  Finds the phoneme index corresponding to a given phoneme spelling.
;
; Parameters:
;   phonemeN = number of phonemes in the model file
;   spelling = spelling of a phoneme.
;   required = is this phoneme required to be known?
;
; Returns:  phonemeX for that phoneme, or -1 if not found.  If a required phoneme is not found, aborts with an error message.
;
; History:
;   6/26/91   jmt   Created, based on old LPNN code.
;  28.Nov 91  Tilo  phonemeN isn't global anymore.
;============================================================================================================================

INT findPhonemeX (phonemeN, spelling, required)
INT   phonemeN;
CHAR *spelling;
INT   required;
{
INT phonemeX;

for (phonemeX=0; phonemeX<phonemeN; phonemeX++) /* check all phoneme... */
    if (streq (phonemeA[phonemeX].spelling, spelling)) /*  if found the given phoneme spelling, */
        return (phonemeX); /* return its index. */
if (required) {
    printf ("Modelfile %s does not recognize phoneme %s.\n", modelfile, spelling); /* failure => abort. */
    exit (-1);
} else return (-1);
}


/*============================================================================================================================
; findDictX:  Finds the dictionary index that corresponds to a spelling.
;
; Parameters:
;   dictN    = number of dictionary entries
;   spelling = spelling of the word.
;   required = is this word required to be in the dictionary?
;
; Returns:  dictX for that word, or -1 if not found.  If a required word is not found, aborts with an error message.
;
; History:
;   6/26/91   jmt   Created, based on old LPNN code.
;  28.Nov 91  Tilo  dictN isn't global anymore.
;============================================================================================================================

INT findDictX (dictN, spelling, required)
```

```
INT   dictN;
CHAR *spelling;
INT   required;
{
INT dictX;

for (dictX=0; dictX<dictN; dictX++)
    if (streq (spelling, dictA[dictX].spellP))
return (dictX);
if (required) {
    printf ("%s is not in the dictionary.\n", spelling);
    exit(-1);
} else return (-1);
}


/*============================================================================================================================
; findVariantP:  Finds a requested variant of a word, if it currently exists in the dictionary.
;
; Parameters:
;   dictX    = dictionary index of word.
;   phonN    = number of phonemes in the word's phonetic pronunciation.
;   phonA    = phonetic pronunciation of word.
;   required = must this variant exist?
;
; Returns:  variantP for that word, or NIL if not found.  If a required variantP is not found, aborts with an error message.
;
; History:
;   6/25/91   jmt   Created.
;============================================================================================================================*/

VARIANT* findVariantP (dictX, phonN, phonA, required)
INT dictX, phonN, phonA[], required;
{
VARIANT* variantP;
for (variantP=dictA[dictX].variantPI; variantP!=NIL; variantP=variantP->nextP) { /* search all current variants... */
    if (variantP->phonN == phonN) { /* if variant has same number of phonemes, */
INT phonX;
for (phonX=0; phonX<phonN; phonX++) /* and all phonemes match... */
    if (variantP->phonA[phonX] != phonA[phonX])
        goto nextVariant;
return (variantP); /* then return matching variantP. */
    }
nextVariant: ;
}
if (required) {
    printf ("Variant for %s is not in the dictionary.\n", dictA[dictX].spellP);
    exit (-1);
} else return (NIL);
}

/*============================================================================================================================
; addDict:  Adds a variant phonetic spelling to the dictionary, if not already present.
;
; Parameters:
;   IN:
; spellP   = lexical spelling of word.
; phonN  = number of phonemes in variant.
; phonA  = phonetic spelling of variant.
;   IN-OUT:
;      &dictN    = number of words currently in the dictionary.
;      &variantN = number of variants currently in the dictionary.
;
; History:
;   6/25/91   jmt   Created.
;============================================================================================================================*/

VOID addDict (spellP, phonN, phonA, dictNP, variantNP)
CHAR   *spellP;
INT phonN, phonA[], *dictNP, *variantNP;
{
INT dictX = findDictX (*dictNP, spellP, OPTIONAL); /* find dictionary index of spelling */
if (dictX == -1) {              /* if not yet in the dictionary: */
    dictX = *dictNP; /* initialize a new dictionary entry... */
    dictA[dictX].spellP = allocS_copy (spellP);
    dictA[dictX].used = FALSE;
    dictA[dictX].minDur = 9999;
    dictA[dictX].maxDur = 0;
    dictA[dictX].labelCount = 0;
    dictA[dictX].labelTotalDur = 0;
    dictA[dictX].variantPI = NIL;
    (*dictNP)++; /* one more dictionary entry */
}
if (findVariantP (dictX, phonN, phonA, OPTIONAL) == NIL) { /* if variant does not yet exist... */
    INT wStateN, phonX;
    PTR wStatePA [MAX_STATESperWORD];
    VARIANT* variantP = allocT (VARIANT); /* allocate a new variant pronunciation */
    variantP->dictX = dictX; /* store dictionary index */
    variantP->vocabX = -1; /* vocabX will be patched up in read_vocab */
    variantP->phonN = phonN; /* store number of phonemes */
    variantP->phonA = allocTA_copy (INT, phonN, phonA); /* allocate & copy phonemes */
    wStateN = 0;
    for (phonX=0; phonX<phonN; phonX++) { /* for each phoneme... */
INT phonemeX = phonA[phonX];
MODEL* pModelP = phonemeA[phonemeX].modelP;
INT pStateN = pModelP->stateN;
```

```
INT pStateX;
for (pStateX=0; pStateX<pStateN; pStateX++) /* for each state in phoneme model: */
    wStatePA [wStateN++] = (PTR) &(pModelP->stateA[pStateX]); /*     append state pointer to big list */
    }
    variantP->stateN = wStateN; /* store number of states in word */
    variantP->statePA = allocTA_copy (STATE*, wStateN, wStatePA); /* allocate & copy array of state pointers */
    variantP->nextP = dictA[dictX].variantPI; /* insert at beginning of linked list of variants */
    dictA[dictX].variantPI = variantP;
    (*variantNP)++; /* one more variant in the dictionary */
}
}

/*=============================================================================================================================
; match:  Tells if a given rewrite rule can apply at a given phoneme position.
;
; Parameters:
;   rewriteX = rewrite index: this is the rule to test.
;   phoneX   = phoneme index: begin comparisons at spellA [phoneX].
;   phoneN   = number of phonemes currently in spellA.
;   spellA   = spelling of each phoneme.
;
; Returns:
;   TRUE if the rewrite rule applies, FALSE if it doesn't.
;
; History:
;   6/26/91   jmt   Created.
;============================================================================================================================*/

INT match (rewriteX, phoneX, phoneN, spellA)
INT rewriteX, phoneX, phoneN;
PSPELL spellA [];
{
INT lhsX;
if (phoneX + rewriteA[rewriteX].lhsN > phoneN)  return (FALSE); /* if left hand side of rule is too long, fail. */
for (lhsX=0; lhsX<rewriteA[rewriteX].lhsN; lhsX++) /* for each token in left hand side: */
    if (strneq (&rewriteA[rewriteX].lhsA[lhsX], &spellA[phoneX+lhsX])) /*     if doesn't match corresponding phoneme, */
        return (FALSE); /*    fail. */
return (TRUE); /* otherwise, succeed. */
}

/*=============================================================================================================================
; apply:  Applies a given rewrite rule at a given position of the phonetic spelling.
;
; Parameters:
;   IN:
; rewriteX = rewrite index: this is the rule to apply.
; phoneX   = phoneme index: apply rule starting at spellA[phoneX].
;   IN/OUT:
;     &phoneN   = number of phonemes in spellA.
; spellA   = spelling of each phoneme.
; durA     = duration of each phoneme.
;
; History:
;   6/26/91   jmt   Created.
;============================================================================================================================*/

VOID apply (rewriteX, phoneX,   phoneNP, spellA, durA)
INT rewriteX, phoneX, *phoneNP, durA[];
PSPELL spellA [];
{
INT lhsN = rewriteA[rewriteX].lhsN; /* get lengths of lhs and rhs */
INT rhsN = rewriteA[rewriteX].rhsN;
INT incr = rhsN - lhsN; /* compute increase/decrease in length */
INT avgDur, lhsX, rhsX, pX;
INT totalDur = 0;
for (lhsX=0; lhsX<lhsN; lhsX++) /* compute total duration for lhs */
    totalDur += durA [phoneX+lhsX];
avgDur = totalDur / rhsN; /* redistribute durations over rhs */
if (incr < 0) /* if decrease in length: */
    for (pX=phoneX+lhsN; pX<=(*phoneNP)-1; pX++) { /*     pull unaffected tokens to the left... */
        spellA [pX+incr] = spellA [pX];
durA [pX+incr] = durA [pX];
    }
if (incr > 0) /* if increase in length: */
    for (pX=(*phoneNP)-1; pX>=phoneX+lhsN; pX--) { /*     push unaffected tokens to the right... */
spellA [pX+incr] = spellA [pX];
durA [pX+incr] = durA [pX];
    }
for (rhsX=0; rhsX<rhsN; rhsX++) { /* for each rhs token: */
    strcpy (&spellA[phoneX+rhsX], &rewriteA[rewriteX].rhsA[rhsX]); /*     copy new phoneme spelling */
    durA [phoneX+rhsX] = (rhsX<rhsN-1 ? avgDur :  /*     store adjusted average duration */
              totalDur-((rhsN-1)*avgDur)); /*    (note: integer division may be inexact) */
}
*phoneNP += incr; /* adjust number of phonemes in current word */
}

/*=============================================================================================================================
; do_rewrites:  Rewrites a phonetic spelling according to the rewrite rules.
;
; Parameters:
;   IN:
; inPhoneN = number of phonemes in the following array, before rewrite rules are applied.
; inSpellA = spelling of each phoneme.
```

```
; inDurA   = duration of each phoneme.
;       phonemeN = number of phonemes in the model file
;   OUT:
;      &outPhonN = number of phonemes, after rewrite rules are applied.
; outPhonA = array of rewritten phonemes, as absolute phoneme indices.
; outDurA  = duration of each rewritten phoneme.
;
; History:
;   6/26/91   jmt   Created.
;=============================================================================================================

VOID do_rewrites (inPhoneN, inSpellA, inDurA, phonemeN,
  outPhonNP, outPhonA, outDurA)
PSPELL inSpellA [];
INT inPhoneN, inDurA[], phonemeN, *outPhonNP, outPhonA[], outDurA[];
{
INT phonX, phoneX, rewriteX;
PSPELL spellA [MAX_PHONSperWORD];
INT durA   [MAX_PHONSperWORD];
INT phoneN = inPhoneN; /* make a local copy of input parameters... */
for (phoneX=0; phoneX<phoneN; phoneX++) { /*     (not strictly necessary, but good practice.) */
    strcpy (&spellA[phoneX], &inSpellA[phoneX]);
    durA[phoneX] = inDurA[phoneX];
}
STARTOVER:
    for (phoneX=0; phoneX<phoneN; phoneX++) /* for each position in phonetic spelling: */
        for (rewriteX=0; rewriteX<rewriteN; rewriteX++) /*     for each rewrite rule: */
    if (match (rewriteX, phoneX, phoneN, spellA)) { /*    if rewrite rule applies here, */
apply (rewriteX, phoneX, /*         apply the rule:  in  */
       &phoneN, spellA, durA); /*          in/out */
goto STARTOVER; /*         and start over. */
    }
*outPhonNP = phoneN; /* return results... */
for (phonX=0; phonX<phoneN; phonX++) {
    outPhonA [phonX] = findPhonemeX (phonemeN, &spellA[phonX], REQUIRED);
    outDurA [phonX] = durA [phonX];
}
}

/*=============================================================================================================
; allocCA:  Allocates an array of characters from private dynamic memory.
;
; Parameters:
;   numChars = number of characters (bytes) to allocate.
;   message  = message describing what is being allocated, in case of failure.  Example: "1000 floats".
;
; Returns:
;   Pointer to the allocated block of memory.  The allocated memory is cleared with zeros.

; History:
;   6/13/91   jmt   Created, based on Patrick Haffner's "challoc_store".
;=============================================================================================================

PTR allocCA (numChars, message)
INT numChars;
CHAR *message;
{
   if (numChars <= 0)  return (NIL);
   else {
      INT i;
      PTR p = (PTR) (dynamicA + dynamicN); /* point to next available dynamic memory location */
      INT numInts = (numChars-1)/sizeof(INT) + 1; /* convert numChars to numInts, to guarantee word alignment */
      if ((numInts MOD 2) == 1)  numInts++; /* avoid word alignment problems */
      if (verbose & MEMORY)  printf ("allocCA: %d %d\n", numInts, dynamicN);
      dynamicN += numInts; /* allocate memory */
      checkLimits (dynamicN, MAX_DYNAMIC_INTS, "MAX_DYNAMIC_INTS");
      for (i=0; i<numInts; i++)  p[i] = 0; /* clear allocated memory */
      return (p); /* return pointer */
   }
}


/*=============================================================================================================
; allocCA_copy:  Allocates an array of characters from private dynamic memory, then copies another array into it.
;
; Parameters:
;   numChars = number of characters (bytes) to allocate.
;   sourceA  = array to be copied into the allocated memory.
;   message  = message describing what is being allocated, in case of failure.  Example: "1000 floats".
;
; Returns:
;   Pointer to the allocated block of memory.  The allocated memory is initialized with the contents of sourceA.

; History:
;   6/13/91   jmt   Created, based on Patrick Haffner's "challoc_store".
;=============================================================================================================

PTR allocCA_copy (numChars, sourceA, message)
INT numChars;
INT *sourceA;
CHAR *message;
{
   if (numChars <= 0)  return (NIL);
   else {
      INT i;
      PTR p = (PTR) (dynamicA + dynamicN); /* point to next available dynamic memory location */
```

```
      INT numInts = (numChars-1)/sizeof(INT) + 1; /* convert numChars to numInts, to guarantee word alignment */
      if ((numInts MOD 2) == 1)  numInts++; /* avoid word alignment problems */
      dynamicN += numInts; /* allocate memory */
      if (verbose & MEMORY)  printf ("allocCA_copy: %d %d\n", numInts, dynamicN);
      checkLimits (dynamicN, MAX_DYNAMIC_INTS, "MAX_DYNAMIC_INTS");
      for (i=0; i<numInts; i++)  p[i] = sourceA[i]; /* initialize allocated memory */
      return (p); /* return pointer */
   }
}

/*=============================================================================================================================
; allocS_copy:  Allocates a copy of a string.
;
; Parameter:  str = source string.
;
; Returns:  Pointer to the allocated block of memory.  The allocated memory is initialized with the contents of str.

; History:
;   6/13/91   jmt   Created, based on Patrick Haffner's "challoc_store".
;============================================================================================================================*/

CHAR* allocS_copy (str)
CHAR* str;
{
return ((CHAR*) allocCA_copy (strlen(str)+1, str, "STRING")); /* allocate and copy string */
}

/*=============================================================================================================================
; open_readfile:  Opens a file for read access; remembers the name of the file.
;
; Parameters:
;   filename = name of the file being opened.
;
; Returns:
;   Pointer to the opened file.  If file cannot be opened, program is aborted.
;
; History:
;   7/3/89  jmt  Created.
;============================================================================================================================*/

FILE* open_readfile (filename)
CHAR *filename;
{
  FILE *fp;

  if ((fp = fopen (filename, "r")) == NULL) { /* try to open filename            */
          printf ("Cannot open %s.\n", filename);   /* failure: give error            */
          exit (-1);} /*      and abort program.     */
  else {
          strcpy (cur_filename, filename); /* success: remember the filename  */
          return (fp); } /*       and return the handle. */
}

/*=============================================================================================================================
; readOK:  Checks the result of fscanf; if there was a problem, abort the program.
;
; Parameter:
;   result = result of a prior call to fscanf.
;
; History:
;   7/3/89  jmt  Created.
;============================================================================================================================*/

VOID readOK (result)
INT result;
{
  if (result == EOF) {
      printf ("Unexpected EOF in %s.\n", cur_filename);
      exit (-1);
  }
}

/*=============================================================================================================================
; checkString:  Checks a string's contents; if it has an unexpected value, abort the program.
;
; Parameters:
;   actual   = string read from a file.
;   expected = what it should equal.
;
; History:
;   7/3/89  jmt  Created.
;============================================================================================================================*/

VOID checkString (actual, expected)
CHAR *actual, *expected;
{
  if (strneq (actual, expected)) {
          printf ("Bad command in %s: expected %s, read %s.\n", cur_filename, expected, actual);
          exit (-1);}
}

/*=============================================================================================================================
; checkLimits:  Checks to make sure a variable is within its allowed limits.
;
; Parameters:
;   value = current value of a variable.
```

```
;   limit = maximum limit for that variable.
;   message = string name of the constant to increase, in the event of failure.
;
; History:
;   6/13/91   jmt   Created.
;=============================================================================================================================

VOID checkLimits (value, limit, message)
INT value, limit;
CHAR *message;
{
     if (value > limit) {
          printf ("Must increase %s.\n", message);
          exit (-1);
      }
}
/*=============================================================================================================================
```

Literaturverzeichnis

[1] Selim G. Akl. *The Design and Analysis of Parallel Algorithms.* Prentice Hall, Englewood Cliffs, NJ., 1989. ISBN 0-132-00056-3.

[2] S. Austin, R. Schwartz, and P. Placeway. The Forward–Backward Search Algorithm. In *International Conference on Acoustics, Speech and Signal Processing*, volume 1, pages 697–700. IEEE, May 1991.

[3] L.R. Bahl, F. Jelinek, and R. Mercer. A Maximum Likelihood Approach to Continous Speech Recognition. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume PAMI–5(2), pages 179–190. IEEE, March 1983. Pages 308–319 in [45].

[4] Tom Blank. The MasPar MP-1 Architecture. In *Proc. of the COMPCON Spring 1990 – The 35th IEEE Computer Society International Conf.*, pages 20–24, San Francisco, California, February 26 – March 2, 1990.

[5] R. Bracewell. *Schnelle Hartley–Transformation: eine reellwertige Alternative zur FFT.* Oldenbourg Verlag, München, Wien, 1990. ISBN 3-486-21079-3.

[6] E. O. Brigham. *FFT: schnelle Fourier–Transformation.* Oldenbourg Verlag, München, Wien, 1989. 4.te Auflage. ISBN 3-486-21332-6.

[7] Gradient Technology Inc., 95B Connecticut Drive, Burlington, NJ 08016. *DeskLab User Manual, Version V9110*, October 1991.

[8] Kamil A. Grajski. Neurocomputing using the MasPar MP-1. Technical Report 90–010, Ford Aerospace Advanced Development Department, MSX–22, San Jose, CA 95161, October 1990.

[9] P. Haffner, M. Franzini, and A. Waibel. Integrating Time Alignment and Neural Networks for High Performance Continuous Speech Recognition. In *International Conference on Acoustics, Speech and Signal Processing*, volume 1, pages 105–108. IEEE, May 1991.

[10] P. Haffner and A. Waibel. Multi-State Time Delay Neural Networks for Continuous Speech Recognition. In J. Moody, S. Hanson, and R. Lippmann, editors, *Advances in Neural Information Processing Systems*, San Mateo, CA, 1992. Morgan Kaufmann.

[11] X.D. Huang, Y. Ariki, and M.A. Jack. *Hidden Markov Models for Speech Recognition.* Edinburgh University Press, Edinburgh, 1990.

[12] F. Jelinek. Self-Organized Language Modeling for Speech Recognition. In A. Waibel and K.-F. Lee, editors, *Readings in Speech Recognition*, chapter 8, pages 450–506. Morgan Kaufmann, 1990. [45].

[13] Fil Alleva Kai-Fu Lee. Continuous Speech Recognition. In M. Mohan Sohnhi and Sadaoki Furi, editors, *Advances in Speech Signal Processing.* Pub. Marcel Dekker, 1991.

[14] T. Kohonen, G. Barna, and R. Chrisley. Statistical Pattern Recognition with Neural Networks: Benchmarking Studies. In *International Conference on Neural Networks*, volume 1, pages 61–68. IEEE, July 1988.

[15] Kai-Fu Lee. Context–Dependent Phonetic Hidden Markov Models for Speaker–Independent Continous Speech Recognition. In *IEEE Transactions on Acoustics, Speech and Signal Processing.* IEEE, April 1990. Pages 347–365 in [45].

[16] MasPar Computer Corporation, Sunnyvale, California. MasPar *MP-1 Principles of Operation, Part Number 9300-5001-A1*, July 1990.

[17] MasPar Computer Corporation, Sunnyvale, California. MasPar *MP-1 Standard Programming Manuals, Part Number 9300-9001-00-A3*, March 1991.

[18] MasPar Computer Corporation, Sunnyvale, California. MasPar *Parallel Application Language (MPL) Reference Manual, Part Number 9302-0000-A4*, March 1991. Part of [17].

[19] MasPar Computer Corporation, Sunnyvale, California. MasPar *Parallel Disk Array (MPDA) Manual, Part Number 9300-5005-A0*, January 1991. BETA RELEASE.

[20] E. McDermott, H. Iwamida, S. Katagiri, and Y. Tohkura. Shift-Tolerant LVQ and Hybrid LVQ-HMM for Phoneme Recognition. In A. Waibel and K.-F. Lee, editors, *Readings in Speech Recognition*, chapter 7, pages 425–438. Morgan Kaufmann, 1990. [45].

[21] E. McDermott and S. Katagiri. Phoneme Recognition Using Kohonen's Learning Vector Quantization. Technical Report TR-I-00??, ATR Interpreting Telephony Research Laboratories, January 1989.

[22] E. McDermott and S. Katagiri. Shift-Invariant, Multi-Category Phoneme Recognition Using Kohonen's LVQ2. In *International Conference on Acoustics, Speech and Signal Processing*, pages 9, S3.1. IEEE, May 1989.

[23] H. Ney. The Use of a One–Stage Dynamic Programming Algorithm for Connected Word Recognition. In *IEEE Transactions on Acoustics, Speech and Signal Processing*, volume ASSP–32(2), pages 263–271. IEEE, April 1984. Pages 188–196 in [45].

[24] John Nickolls. The Design of the MasPar MP-1, A Cost-Effective Massively Parallel Computer. In *Proc. of the COMPCON Spring 1990 – The 35th IEEE Computer Society International Conf.*, pages 25–28, San Francisco, California, February 26 – March 2, 1990.

[25] Michael Philippsen, Thomas Warschko, Walter F. Tichy, and Christian Herter. Projekt Triton: Beiträge zur Verbesserung der Programmierbarkeit hochparalleler Rechensysteme. *Informatik—Forschung und Entwicklung*, 7, January 1992.

[26] L.R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. In *Proceedings of the IEEE*. IEEE, 1989. Pages 267–296 in [45].

[27] L.R. Rabiner and S.E. Levinson. Isolated and Connected Word Recognition — Theory and Selected Applications. In *IEEE Transactions on Communications*, volume COM–29(5), pages 621–659. IEEE, May 1981. Pages 115–153 in [45].

[28] L.R. Rabiner and Schafer R.W., editors. *Digital Processing of Speech Signals*. Prentice Hall, Englewood Cliffs, NJ., 1978. ISBN 0-132-13603-1.

[29] R. Ramirez. *The FFT, Fundamentals and Concepts*. Prentice Hall, Englewood Cliffs, NJ., 1985. ISBN 0-133-14386-4.

[30] H. Sakoe. Two–Level DP–Matching — A Dynamic Programming–Based Pattern Matching Algorithm for Connected Word Recognition. In *IEEE Transactions on Acoustics, Speech and Signal Processing*, volume ASSP–27(6), pages 588–595. IEEE, December 1979. Pages 180–187 in [45].

[31] H. Sakoe and S. Chiba. Dynamic Programming Algorithm Optimization for Spoken Word Recognition. In *IEEE Transactions on Acoustics, Speech and Signal Processing*, volume ASSP–26(1), pages 43–49. IEEE, February 1978. Pages 159–165 in [45].

[32] D. Sanner and A. Waibel. Performance Measures for LPNN–Forward Passes using the iWarp. Unpublished comments. DARPA Neural Network review meeting, August 1991.

[33] O. Schmidbauer and J. Tebelskis. An LVQ based Reference Model for Speaker–Adaptive Speech Recognition. To be published in ICASSP, March 1992.

[34] Otto Schmidbauer. An LVQ based Reference Model for Speaker–Independent and –Adaptive Speech Recognition. Internal technical report, Siemens AG, ZFE IS KOM3, 8 Muenchen 83, Germany, 1991.

[35] R. Schwartz and S. Austin. A Comparison of Several Approximate Algorithms for Finding Multiple (N-Best) Sentence Hypotheses. In *International Conference on Acoustics, Speech and Signal Processing*, volume 1, pages 701–704. IEEE, May 1991.

[36] R. Schwartz and Y.-L Chow. The N–best Algorithm: An Efficient and Exact Procedure for Finding the N Most Likely Sentence Hypotheses. In *International Conference on Acoustics, Speech and Signal Processing*, volume 1, pages 81–84. IEEE, April 1990.

[37] V. Steinbiss. Sentence–Hypotheses Generation in a Continous–Speech Recognition System. In *European Conference on Speech, Communication and Technology*, volume 2, pages 51–54. Philips GmbH Forschungslaboratorium Hamburg, September 1989.

[38] Bernhard Suhm. Pipelining bei der Datenvorverarbeitung zur Spracherkennung. Universität Karlsruhe, Studienarbeit am Lehrstuhl Waibel, March 1992.

[39] J. Tebelskis and A. Waibel. Large Vocabulary Recognition Using Linked Predictive Neural Networks. In *International Conference on Acoustics, Speech and Signal Processing*, volume 1, pages 437–440. IEEE, April 1990.

[40] J. Tebelskis, A. Waibel, B. Petek, and O. Schmidbauer. Continuous Speech Recognition by Linked Predictive Neural Networks. In R. Lippmann, J. Moody, and D. Touretzky, editors, *Advances in Neural Information Processing Systems*, San Mateo, CA, 1991. Morgan Kaufmann.

[41] J. Tebelskis, A. Waibel, B. Petek, and O. Schmidbauer. Continuous Speech Recognition Using Linked Predictive Neural Networks. In *International Conference on Acoustics, Speech and Signal Processing*, volume 1, pages 61–64. IEEE, May 1991.

[42] Walter F. Tichy and Christian G. Herter. Modula-2*: An extension of Modula-2 for highly parallel, portable programs. Technical Report No. 4/90, University of Karlsruhe, Department of Informatics, January 1990.

[43] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and Lang K. Phoneme Recognition Using Time-Delay Neural Networks. *IEEE, Transactions on Acoustics, Speech and Signal Processing*, March 1989.

[44] A. Waibel, A. Jain, A. McNair, H. Saito, A. Hauptmann, and J. Tebelskis. JANUS: A Speech-to-Speech Translation System Using Connectionist and Symbolic Processing Strategies. In *International Conference on Acoustics, Speech and Signal Processing*, volume 2, pages 793–796. IEEE, May 1991.

[45] A. Waibel and K.-F. Lee, editors. *Readings in Speech Recognition*. Morgan Kaufmann, San Mateo, CA., 1990. ISBN 1-558-60124-4.

[46] A. Waibel and B. Yegnanarayana. Comparative Study of Nonlinear Time Warping Techniques in Isolated Word Speech Recognition Systems. Technical Report CMU-CS-81-125, Carnegie-Mellon University, Department of Computer Science, June 1981.

[47] Monika Woszczyna. LVQ–Erkenner für die deutsche Sprache. Universität Karlsruhe, Interner Bericht des Lehrstuhls Waibel, January 1992.

[48] Xiru Zhang, Michael McKenna, Jill P. Mesirov, and David Waltz. An Efficient Implementation of the Backpropagation Algorithm on the Connection Machine CM–2. Technical Report RL89–1, Thinking Machines Corporation, Cambridge, MA 02142–1214, August 1989.

Danksagung

Abschließend möchte ich mich bei allen bedanken, die zum Entstehen dieser Diplomarbeit beigetragen haben. Für das Korrekturlesen und für die dadurch entstandenen Anregungen möchte ich mich bei Michael Phillipsen und Lutz Prechelt bedanken. Mein besonderer Dank gilt Monika Woszczyna für das Korrekturlesen und für Ihre Unterstützung bei der Einbindung der entstandenen Implementationen in JANUS. Ebenso möchte ich mich bei Joe Tebelskis bedanken, der sich bei Diskussionen per Email immer besondere Mühe gab und unermüdlich für Fragen zur Verfügung stand.