Zeitschrift: Bulletin des Schweizerischen Elektrotechnischen Vereins, des Verbandes Schweizerischer Elektrizitätsunternehmen = Bulletin de l'Association suisse des électriciens, de l'Association des entreprises électriques suisses

Herausgeber: Schweizerischer Elektrotechnischer Verein ; Verband Schweizerischer Elektrizitätsunternehmen

Band: 79 (1988)

Heft: 7

Artikel: Nora : ein Softwarepaket zur Programmierung und Analyse von parallelen Signalprozessorsystemen

Autor: Hufschmid, M. / Löffler, C.

DOI: https://doi.org/10.5169/seals-904018

Nutzungsbedingungen

Die ETH-Bibliothek ist die Anbieterin der digitalisierten Zeitschriften auf E-Periodica. Sie besitzt keine Urheberrechte an den Zeitschriften und ist nicht verantwortlich für deren Inhalte. Die Rechte liegen in der Regel bei den Herausgebern beziehungsweise den externen Rechteinhabern. Das Veröffentlichen von Bildern in Print- und Online-Publikationen sowie auf Social Media-Kanälen oder Webseiten ist nur mit vorheriger Genehmigung der Rechteinhaber erlaubt. Mehr erfahren

Conditions d'utilisation

L'ETH Library est le fournisseur des revues numérisées. Elle ne détient aucun droit d'auteur sur les revues et n'est pas responsable de leur contenu. En règle générale, les droits sont détenus par les éditeurs ou les détenteurs de droits externes. La reproduction d'images dans des publications imprimées ou en ligne ainsi que sur des canaux de médias sociaux ou des sites web n'est autorisée qu'avec l'accord préalable des détenteurs des droits. En savoir plus

Terms of use

The ETH Library is the provider of the digitised journals. It does not own any copyrights to the journals and is not responsible for their content. The rights usually lie with the publishers or the external rights holders. Publishing images in print and online publications, as well as on social media channels or websites, is only permitted with the prior consent of the rights holders. Find out more

Download PDF: 10.12.2025

ETH-Bibliothek Zürich, E-Periodica, https://www.e-periodica.ch

Nora – ein Softwarepaket zur Programmierung und Analyse von parallelen Signalprozessorsystemen

M. Hufschmid und Ch. Löffler

Im vorliegenden Artikel wird das am Institut für Signal- und Informationsverarbeitung der ETH Zürich entwickelte Programmpaket Nora (Net Optimization and Resource Allocation) vorgestellt. Nora ist ein Entwicklungswerkzeug, das die Analyse und Programmierung von parallelen Signalprozessorsystemen wesentlich vereinfacht. Einerseits besteht die Möglichkeit, einen beliebigen Signalverarbeitungs-Algorithmus nahezu optimal auf eine gegebene Hardware abzubilden, anderseits können die Leistungen verschiedener Prozessorarchitekturen untereinander verglichen werden.

Dans le présent article, l'ensemble de la software Nora (Net Optimization and Resource Allocation), développé à l'institut de traitement des signaux et de l'information de l'EPF de Zurich, est présenté. Nora a été spécialement conçu pour simplifier l'analyse et la programmation des systèmes de micro-processeurs parallèles, utilisés pour le traitement des signaux électriques. Il offre d'une part la possibilité de répartir un algorithme de traitement de signaux d'une manière presque parfaite sur un système de hardware donné, et d'autre part la possibilité d'évaluer et de comparer les rendements de différents modèles d'architecture.

Adresse der Autoren

Dipl. El.-Ing. ETH *Markus Hufschmid* und Dipl. El.-Ing. ETH *Christoph Löffler*, Institut für Signal- und Informationsverarbeitung, ETH Zentrum, 8092 Zürich.

Die digitale Signalverarbeitung hat in den letzten Jahren stark an Bedeutung gewonnen. In vielen Bereichen der Elektrotechnik werden heute in zunehmendem Masse Prozessoren zur Lösung von Signalverarbeitungsproblemen eingesetzt. Die digitale Verarbeitung von Signalen zeichnet sich durch hohe Genauigkeit, gute Reproduzierbarkeit und Flexibilität aus. Dies hat zur Folge, dass sie für gewisse Anwendungen der traditionellen Analogtechnik vorgezogen wird. Ferner erlaubt die digitale Signalverarbeitung Lösungen (z.B. adaptive Filter), welche mit herkömmlichen Methoden nicht oder nur mit ungleich höherem Aufwand zu realisieren sind.

Als vor etwa vierzig Jahren erste Programme zur Signalverarbeitung entwickelt wurden, war eine Echtzeitverarbeitung aufgrund der damaligen bescheidenen Rechenleistungen nicht denkbar. Erst nach der Entwicklung von speziellen VLSI-Chips gelang es, die Algorithmen in Echtzeit zu berechnen. Die Leistungsfähigkeit der heute verfügbaren Signalprozessoren ist beeindruckend. Bedingt durch die steigende Komplexität der Algorithmen einerseits und den Anstieg der geforderten Abtastraten anderseits, sind jedoch auch die Anforderungen an die Rechenleistung der Prozessoren stark gestiegen. Für viele Anwendungen ist ein einzelner Prozessor nicht mehr ausreichend, es müssen mehrere, parallel arbeitende Prozessoren eingesetzt werden. Dabei stösst man leider auf Probleme, welche bis heute nicht zufriedenstellend gelöst werden konnten. Der Aufbau eines Parallelprozessorsystems ist mit den verfügbaren VLSI-Chips keine unlösbare Aufgabe, wie eine Vielzahl bereits realisierter Systeme (Cray II, Illiac IV, Empress usw.) zeigt. Hingegen stösst man auf Schwierigkeiten, wenn ein Algorithmus unter möglichst *optimaler Ausnützung* der vorhandenen *Ressourcen* auf einem solchen System implementiert werden soll. Es existieren noch kaum Werkzeuge, mit denen beliebige sequentielle Programme auch nur annähernd optimal auf eine gegebene Parallelrechnerstruktur abgebildet werden können. Dem Anwender steht zwar eine ausgereifte Hardware zur Verfügung, die Leistungsfähigkeit der Softwarewerkzeuge lässt jedoch häufig zu wünschen übrig. Während sich für allgemeine Problemstellungen diesbezüglich in naher Zukunft nicht viel ändern wird, ist für eingeschränkteAnwendungsgebiete die Möglichkeit der Entwicklung von leistungsfähigen Programmierwerkzeugen bereits absehbar. Am Institut für Signal- und Informationsverarbeitung (ISI) der ETH Zürich wurde ein Softwarepaket für die Implementation von Signalverarbeitungs-Algorithmen auf parallelen Rechnerarchitekturen entwickelt, das bei hoher Flexibilität eine sehr gute Ausnützung der Rechnerressourcen ermöglicht.

Problemstellung

Viele Echtzeit-Signalverarbeitungsanwendungen benötigen einen sehr hohen Datendurchsatz, welcher nur durch den Einsatz von parallel arbeitenden Prozessoren bewältigt werden kann. Unglücklicherweise sind solche Systeme im allgemeinen schwierig zu programmieren. Untersucht man jedoch die in der Signalverarbeitung verwendeten Algorithmen, so zeigt es sich, dass diese sehr ähnliche Eigenschaften aufweisen:

- Der Programmablauf ist weitgehend unabhängig von den zu verarbeitenden Daten. Er ist demnach schon vor Ausführung des Programms bekannt. Eine Darstellung in einem *Datenflussgraphen* ist fast immer möglich.
- Signalverarbeitungs-Algorithmen lassen sich häufig mit Hilfe von Vekto-

ren und Matrizen beschreiben, was auf das Vorhandensein von expliziter Parallelität schliessen lässt. Das wohl bekannteste Beispiel ist der in den sechziger Jahren von *Cooley* und *Tukey* veröffentlichte Fast Fourier Transform (FFT) Algorithmus. Mit diesem Algorithmus benötigt ein einzelner Prozessor zur Transformation eines N-dimensionalen Signalvektors $K \cdot N \log_2 N$ Zeiteinheiten, während N Prozessoren dieselbe Aufgabe N-mal schneller bewältigen.

- Für die Signalverarbeitung werden zwar hohe Rechenleistungen, meist jedoch nur einfache Operationen wie Addition und Multiplikation benötigt. Die Komplexität der einzelnen Prozessorelemente ist aus diesem Grund nicht sehr hoch.

Am Institut für Signal- und Informationsverarbeitung wurde nun versucht, diese spezifischen Eigenschaften auszunützen und ein Programm zu entwickeln, das die weitgehend automatische Implementation eines Signalverarbeitungs-Algorithmus auf ein Parallelprozessorsystem gestattet [2,...,5]. Die Hauptaufgabe besteht dabei in der *Abbildung* eines Algorithmus auf eine Hardwarestruktur. In diesem Zusammenhang stellt sich vorerst die Frage, welche der vielen heute bekannten Rechnerstrukturen (Vektorrechner, Array-Prozessoren, Systolische Arrays, Multiprozessoren, Datenflusssysteme usw.) sich besonders für die Bedürfnisse der Signalverarbeitung eignen. Eine optimale Ausnützung der Parallelität wird durch Datenflusssysteme erreicht, da bei diesen der Ablauf der Rechenschritte nicht durch eine Kontrollstruktur festgelegt, sondern aus dem Vorhandensein der Daten abgeleitet wird. Leider ist der Aufwand für die Ablaufplanung (Scheduling) der Operationen im allgemeinen sehr hoch. Bei Signalverarbeitungs-Algorithmen kann die Ablaufplanung jedoch schon zur Compilationszeit vollzogen werden, da der Programmablauf grösstenteils schon vor Ausführung des Programms bekannt ist. Die wenigen datenbedingten Verzweigungen sind so einfach, dass sie mit Hilfe von einfachen Schalterfunktionen gelöst werden können.

Häufig ist nicht die minimal erreichbare Rechenzeit von Interesse, sondern der minimale Hardwareaufwand für eine vorgegebene Rechenzeit. Ein umfassendes Entwicklungspaket muss sowohl in der Lage sein, die Operationen einzuplanen, als auch die Vergabe der einzelnen Operationen an die vorhandenen Ressourcen (Prozessorelemente, Speicher usw.) vorzunehmen. Schliesslich soll als Endprodukt ein lauffähiges Mikroprogramm für das realisierte Parallelprozessorsystem erzeugt werden.

Der Datenflussübersetzer PSPL

Die Struktur des gesamten Entwicklungssystems Nora ist in Figur 1 dargestellt. Als Eingabe wird einerseits der zu realisierende Algorithmus und anderseits eine Beschreibung der Hardwarearchitektur verlangt. Das System liefert eine Aussage über die Parallelität des Algorithmus sowie ein formales Mikroprogramm, das als Vorlage zur Generierung des eigentlichen Mikrocodes für den parallelen Signalprozessor dienen kann.

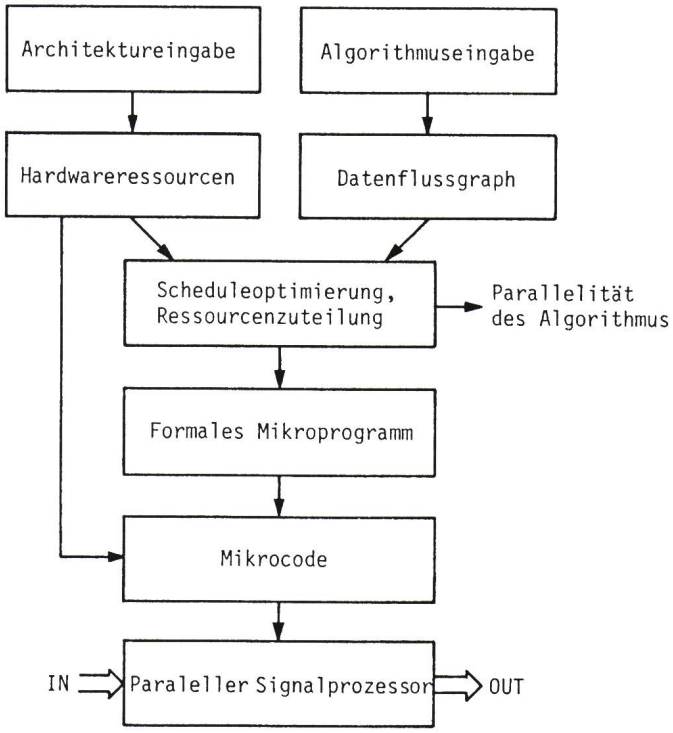
In einem ersten Schritt wird der gegebene Signalverarbeitungs-Algorithmus in einen Datenflussgraphen umgewandelt. Diese Aufgabe übernimmt der PSPL-Compiler. PSPL (Parallel Signal Processing Language) ist eine einfache Hochsprache, welche speziell für Signalverarbeitungsanwendungen am ISI entwickelt wurde. Syntax und Semantik sind weitgehend aus den bekannten Sprachen Pascal und C entlehnt. Neben den üblichen Befehlen unterstützt PSPL auch das Formulieren von Algorithmen in Vektorschreibweise sowie spezielle Funktionen (z.B. Shift, Bitreverse Adressing usw.).

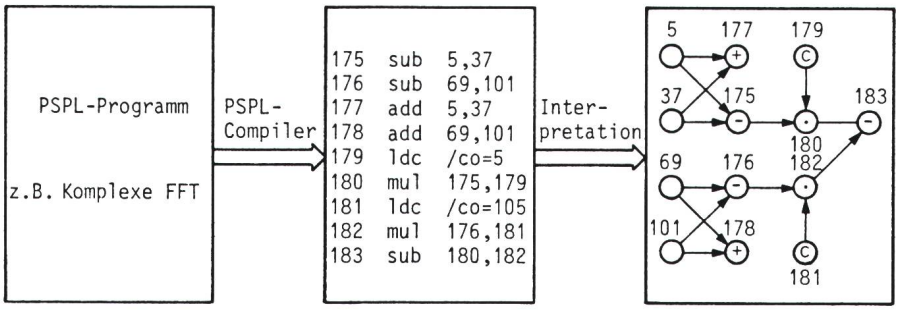
Der Compiler besteht aus drei Teilen: dem Preprozessor, der den Quellencode vorverarbeitet (u.a. die Schleifen auflöst und die lokalen Variablen beseitigt), dem eigentlichen Übersetzer (Compiler) und dem Outputgenerator. Als Ausgabe liefert der Compiler eine Rückwärtsabhängigkeitstabelle, welche einer tabellenartigen Darstellung des Datenflussgraphen des eingegebenen Algorithmus entspricht. Die Figur 2 zeigt, wie aus einem PSPL-Programm vom Compiler die Rückwärtsabhängigkeitstabelle generiert wird. Zum besseren Verständnis ist gleichzeitig auch die graphische Interpretation (Datenflussgraph) wiedergegeben.

Hardwaremodell

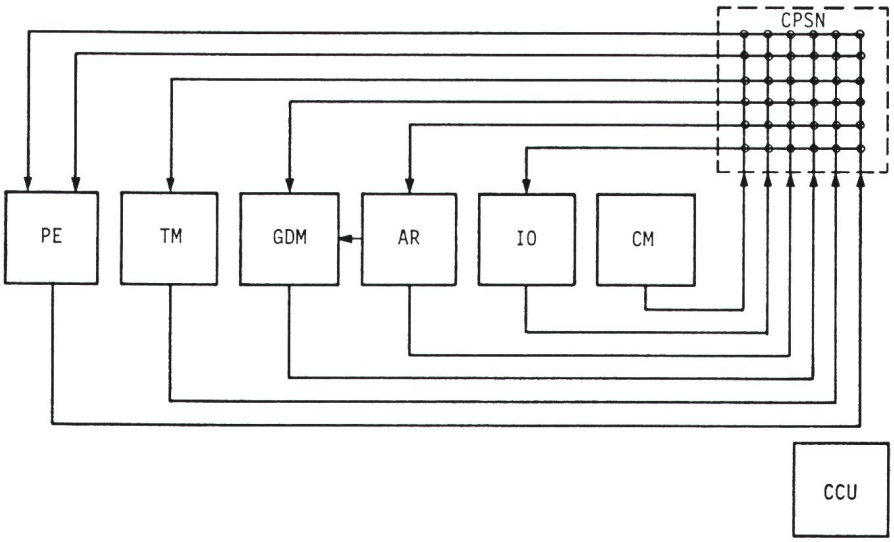
Damit das Entwicklungspaket Nora die Zuteilung der Operationen an die vorhandenen Ressourcen vornehmen kann, wird eine Beschreibung der Hardwarearchitektur benötigt. Ausgehend von den Bedürfnissen der digitalen Signalverarbeitung wurde das in Figur 3 dargestellte Hardwaremodell entworfen. Die einzelnen Baublöcke sind jeweils einmal gezeichnet worden, können jedoch auch mehrmals auftreten.

Figur 1
Übersicht über das Softwarepaket Nora





Figur 2 PSPL-Compiler



Figur 3 Hardwaremodell

PE Das Processing Element führt einfache Operationen und Funktionen aus.

GDM Das Global Data Memory wird für die Speicherung von global zugänglichen Daten, d.h. etwa für Verzögerungselemente in Filteralgorithmen oder für Eingangsdaten einer FFT verwendet.

AR Das Address Register wird für die Adressierung des GDM verwendet.

CM Das Coefficient Memory speichert Konstanten, beispielsweise für Filterkoeffizienten.

IO Das Input/Output Device, z.B. ein AD/DA-Konverter, kommuniziert mit der Aussenwelt.

TM Das Temporary Memory speichert Zwischenresultate.

CPSN Das Cross-Point Switch Network verbindet die einzelnen Baublöcke.

CCU Die Central Control Unit ist die zentrale Steuerung des Systems.

Für die Verbindung der Baublöcke wurde im Modell ein Cross-Point-Switch-Netzwerk gewählt, da es die allgemeinste Lösung für ein Verbindungsnetzwerk darstellt. Es ist aber nicht beabsichtigt, ein solches Netzwerk auch für die Hardwarerealisierung zu verwenden. Da nicht alle Verbindungsknoten einen Schalter enthalten müssen, wird definiert, ob ein Cross-Point geschaltet, immer offen oder dauernd geschlossen ist. Auf diese Art lassen sich *beliebige* Prozessorstrukturen beschreiben. Das hier beschriebene Modell deckt alle Bedürfnisse der Signalverarbeitung bestens ab. Es lässt sich beliebig erweitern, indem die Anzahl der einzelnen Baublöcke erhöht wird. Im weiteren ist eine hierarchische Modellierung der Hardwarestruktur möglich, d.h. es können Subsysteme formuliert werden, die wiederum zu Gesamtsystemen zusammengefasst werden können.

Ablaufoptimierung und Ressourcenzuteilung

Das zentrale Problem bei der Programmierung eines Datenflussrechners besteht in der Optimierung der Ablaufplanung (Scheduleoptimierung) und in der Ressourcenzuteilung. Die erstere arbeitet den «Fahrplan» der Operationen (welche Operation wird *wann* im Gesamtablauf ausgeführt) so aus, dass die totale Ausführungszeit bei einer gegebenen Anzahl Rechenelemente minimal wird. Es wird vorerst angenommen, dass die Speicher- und Datentransferoperationen ideal sind. Die Zuweisung der Operationen an eine bestimmte Ressource sowie das Finden eines geeigneten Speicherplatzes und Datenpfades ist Aufgabe der Ressourcenzuteilung. Die Scheduleoptimierung und die Ressourcenzuteilung bilden den vom PSPL-Compiler gelieferten Datenflussgraphen auf das oben beschriebene Hardwaremodell ab. Im Gegensatz zu früheren Arbeiten [2;5;6] sind im System Nora die Ablaufoptimierung und die Ressourcenzuteilung kombiniert. Dies erlaubt die Modellierung von beliebigen Hardwarekonfigurationen unter Einbezug der Hardwareeinschränkungen, welche durch die Speicher- und Datentransferoperationen gegeben sind. Diese Einschränkungen haben einen direkten Einfluss auf die minimal erreichbare Abarbeitungszeit des Algorithmus.

Zum Bestimmen des optimalen Scheduling wurde ein heuristisches Verfahren gewählt, da kein Algorithmus bekannt ist, der dieses Problem in polynomialer Zeit[1] löst (NP-komplettes Problem). Die verwendete Methode wurde im Grundsatz von *Gewald et al.* [7] vorgeschlagen und beinhaltet die folgenden Schritte:

1. Feststellen der einplanbaren Operationen: Es sind dies die Operationen, deren Vorgänger schon eingeplant sind oder die selber keinen Vorgänger haben.

2. Ordnen der Operationen nach Prioritäten: Diese bestimmen sich aus einer geschickten Kombination verschiedener Kriterien, von welchen hier nur einige aufgezählt werden:

- Speicherbelegung,
- Pufferzeit (die freie bzw. gesamte Pufferzeit bestimmt, um wieviel ein Task verschoben werden kann, ohne dass der früheste bzw. späteste Anfangszeitpunkt eines Nachfolgers verschoben wird),
- graphtopologische Kriterien wie «Joint-Priorität» (Erhöhung der Priorität für Tasks, die den gleichen Nachfolger besitzen wie ein soeben eingeplanter) oder Anzahl Nachfolger,
- mögliche Ressourcen für die Ausführung eines Tasks und mögliche Ausgangsressourcen (Verbindun-

[1] Die Rechenzeit nimmt in Funktion der zu verarbeitenden Informationsmenge schneller als jede beliebige Polynomfunktion zu.

gen, Zwischenspeicher für Resultate),
- benötigte Verbindungen.

3. Einplanen der Operationen gemäss Priorität, bis keine weiteren Operationen mehr zum gegebenen Zeitpunkt möglich sind.

4. Zeitintervall um eine Zeiteinheit erhöhen.

5. Repetition des Vorgehens ab Punkt 1 bis alle Operationen eingeplant sind.

Trotz des heuristischen Vorgehens zeigt es sich, dass die erhaltenen Resultate meist sehr nahe am Optimum liegen. Dies wird etwas später anhand eines Beispiels verdeutlicht werden.

Der Rechner

Um die vom Softwarepaket Nora gelieferten Ergebnisse auch in der Praxis überprüfen zu können, wird zurzeit am Institut für Signal- und Informationsverarbeitung ein Parallelrechnersystem aufgebaut. Besonderes Gewicht wurde dabei auf eine möglichst vollständige Modularität gelegt. Wie die Figur 4 zeigt, besteht der Rechner aus einem Bussystem und einer Steuereinheit (Controller). Letztere besitzt eine Schnittstelle zu einem Hostsystem, welches zum Laden und Debuggen von Programmen dient. Hinzu kommt eine nahezu beliebige Anzahl (paralleler) Arithmetik-, Speicher-, Input/Output- und eventuell Verbindungsmodule. Die einzelnen Module lassen sich gut mit Hilfe des oben eingeführten Hardwaremodells beschreiben; als Beispiel ist in Figur 5 die Modellierung des Arithmetikmoduls dargestellt. Um volle Modularität zu gewährleisten, besitzt jedes Modul seinen eigenen Mikrocodespeicher, in dem das Mikroprogramm für das Modul abgelegt wird. Denkt man sich die Mikrocodespeicher der einzelnen Module aneinandergereiht, so ergibt sich ein sehr langes Mikrocodewort, man spricht von einem VLIW-Rechner (Very Long Instruction Word). Um die Anzahl der für die Verbindung zwischen den Modulen benötigten Busse möglichst klein zu halten, wurden einige zusätzliche Massnahmen ergriffen:

- Beim Zugriff auf den Temporärspeicher arbeitet der Bus im Multiplexbetrieb. Während der ersten Hälfte des Basiszyklus kann der Speicher beschrieben, während der zweiten Hälfte kann er gelesen werden.
- Es wurden Dual-Port-Speicher eingesetzt. Diese Bausteine können als Busschalter verwendet werden, da sie zwei Eingänge und zwei Ausgänge besitzen.
- Durch den Einsatz von Tri-State-Ausgangsregistern wurde die Mehrfachbelegung von Bussen ermöglicht. Selbstverständlich muss die Mehrfachbelegung eines Busses bei der Ressourcenzuteilung berücksichtigt werden.

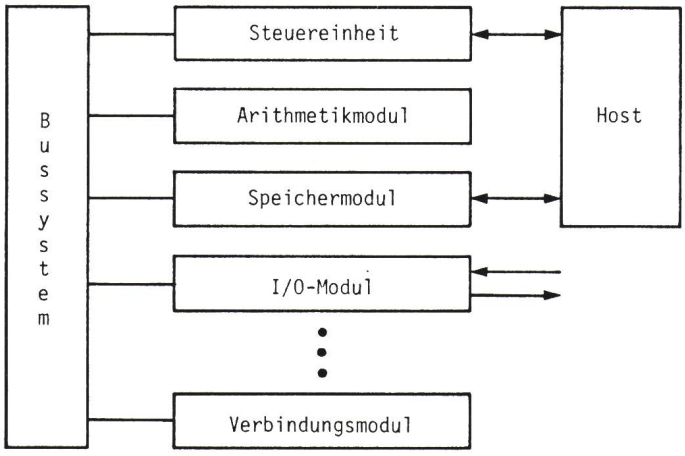
Bis heute wurden die Controllerkarte, ein Arithmetik-, ein Speicher- sowie ein Input-Output-Modul aufgebaut. Das realisierte System ist durch folgende Grössen charakterisiert:

Datenformat:	32 Bit Floating Point
Zykluszeit:	100 ns
Verwendete Bausteine:	Serie AMD 293xx
Hostrechner:	System 68000
Aufbau:	3fach-Europakarten, Wire-wrap
Bussystem:	gemeinsamer Adress- und Steuerbus 6 Datenbusse à 32 Bit, unterteilbar
Dauer der Operationen:	Addition, Subtraktion, Multiplikation: je 1 Zyklus

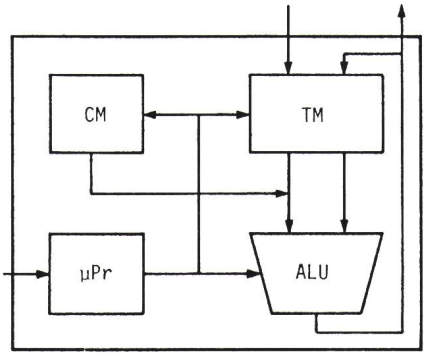
Beispiel

Im folgenden soll die Anwendung des Softwarepakets Nora anhand eines Beispiels demonstriert werden. Wir nehmen an, ein Parallelprozessorsystem mit der in Figur 5 dargestellten Hardwarearchitektur sei gegeben. Es besteht aus einer Steuereinheit, einem Speicher- sowie drei Arithmetikmodulen. Die drei Arithmetikmodule seien untereinander im Ring und mit dem Speichermodul über drei bidirektionale Busschalter verbunden. Auf diesem System soll eine komplexe 64-Punkt-Fast-Fourier-Transformation implementiert werden. Es handelt sich dabei um einen typischen Signalverarbeitungs-Algorithmus mit 2332 Operationen (Additionen, Subtraktionen, Multiplikationen, Laden und Speichern der Signale). Das erste Ergebnis, das man nach Eingabe des Algorithmus und der Hardwarestruktur erhält, ist die minimale Abarbeitungsdauer des Algorithmus unter der Annahme, dass die Ressourcen in unbeschränktem Masse zur Verfügung stünden. Es ist dies ein direktes *Mass für* die im Algorithmus vorhandene *Parallelität*. In unserem Beispiel könnte die FFT in minimal 18 Zeiteinheiten gerechnet werden. Dazu würden jedoch 128 Arithmetikeinheiten benötigt.

Nora erlaubt nun, für den gewählten Algorithmus verschiedene Architekturvarianten zu untersuchen und untereinander zu vergleichen. In Tabelle 1 sind die Ergebnisse einiger Untersuchungen zusammengestellt. Die erste Kolonne zeigt die Anzahl benötigter Rechenschritte und den Ausnüt-



Figur 4 Blockschaltbild des parallelen Signalprozessorsystems



Figur 5 Arithmetikmodul

µPr Mikroprogramm
CM Koeffizientenspeicher
TM Zwischenresultatspeicher
ALU Arithmetik-Logic-Einheit

Anzahl Arithmetikmodule	I Processing-Elemente beschränkt	II Processing-Elemente und Speicher beschränkt	III Bei Architektur der Figur 5
2	**776** 99%	**811** 95%	**817** 94%
3	**512** 99%	**557** 92%	**606** 85%
4	**396** 97%	**439** 88%	**450** 86%

Die Tabelle zeigt die Anzahl benötigter Rechenschritte, und zwar unter der Bedingung,

I dass Speicherressourcen und Verbindungen unbeschränkt zur Verfügung stehen,
II dass Verbindungen unbeschränkt zur Verfügung stehen,
III dass Processing-Elemente, Speicherressourcen und Verbindungen beschränkt zur Verfügung stehen.

Tabelle 1 Resultate zum FFT-Beispiel

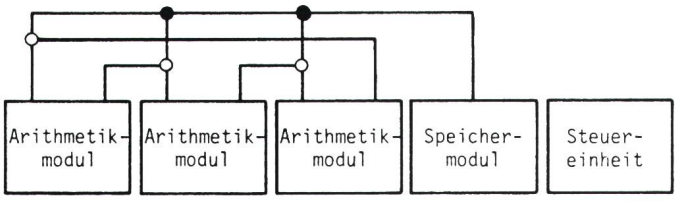
zungsgrad der Rechenmodule falls die Speicherressourcen und die Verbindungen in unbeschränktem Masse zur Verfügung stünden. Es wird also lediglich die Einschränkung durch die Processing-Elemente (PE) betrachtet. In Kolonne 2 wurden die Zwischenspeicher modelliert, hingegen wurde angenommen, dass die Elemente über ein vollständiges Cross-Point-Switch-Netzwerk verbunden sind. Die letzte Kolonne zeigt schliesslich die erreichbaren Rechenzeiten bei einer Hardwarearchitektur, die derjenigen von Figur 5 entspricht. Bemerkenswert ist der durchwegs hohe Ausnutzungsgrad von 85 bis 99%, welcher mit dem heuristischen Optimierungsverfahren erreicht wurde. Wie anzunehmen war, nimmt die Rechenzeit mit zunehmender Beschränkung der Ressourcen (in der Tabelle von links nach rechts) zu. Verglichen mit dem deutlich höheren Hardwareaufwand der idealen Variante, ist der Gewinn an Rechenleistung jedoch bescheiden. Dies ist ein Hinweis darauf, dass eine Vereinfachung der Hardwarestruktur bis zu einem gewissen Grad keinen entscheidenden Einfluss auf die Leistung des Systems haben muss.

Das Softwarepaket Nora eignet sich gut dazu, für ein gegebenes Problem (in unserem Beispiel die Bestimmung der diskreten Fourier-Transformation) verschiedene Algorithmen und Architekturen auf ihre Eignung hin zu untersuchen. Hat man sich für eine Lösung entschieden, liefert Nora die notwendige Rechenzeit und die Auslastung der diversen Hardwareelemente; schliesslich besteht noch die Möglichkeit, ein lauffähiges Mikroprogramm zu generieren. Dadurch wird die Analyse von parallelen Signalprozessorarchitekturen und ihre Programmierung wesentlich erleichtert.

Zusammenfassung und Ausblick

Im vorliegenden Artikel wurde ein Softwarepaket zur Analyse, Synthese und Programmierung von parallelen Signalprozessorarchitekturen vorgestellt. Wir haben uns bewusst auf Signalverarbeitungsprobleme beschränkt, um die besonderen Eigenschaften von derartigen Algorithmen ausnützen zu können. Das Softwarepaket Nora kann eingesetzt werden, um verschiedene Algorithmen zu analysieren und untereinander zu vergleichen. Insbesondere wird die dem Algorithmus inhärente Parallelität bestimmt. Nora gestattet ferner die Modellierung, Simulation und den Vergleich von Hardwarearchitekturen. Es ist somit möglich, einen guten Kompromiss zwischen Hardwareaufwand und Rechenzeit zu finden. Da die Auslastung der einzelnen Komponenten angegeben wird, kann die Hardware gezielt an den gewählten Algorithmus angepasst werden. Um Hinweise auf noch zu lösende Problempunkte zu erhalten, wird zurzeit am Institut für Signal- und Informationsverarbeitung ein paralleles Signalprozessorsystem realisiert. Die gewählte Hardware besitzt einen hohen Grad an Modularität, ist gut realisierbar und zeichnet sich trotz ihrer strukturellen Einfachheit durch hohe Effizienz aus.

Obwohl das bestehende Softwarepaket Nora für viele Beispiele schon gute Resultate liefert, sind noch Verbesserungen geplant. So wäre es von Vorteil, wenn der vom PSPL-Compiler gelieferte Datenflussgraph auch nachträglich vom Benutzer modifiziert werden könnte. Die Parallelität kann dadurch häufig noch gesteigert werden. Im weiteren sollen neben Signalprozessoren auch VLSI-Chips als Zielsysteme unterstützt werden, was die Attraktivität des Systems für gewisse Anwender erhöhen dürfte. Neu sollen als Architekturelemente in Zukunft auch ganze Blöcke und etwas speziellere Bausteine wie beispielsweise Rotatoren (Digitales Phasendrehglied) eingesetzt werden können. Schliesslich muss noch ein Konzept zur effizienten Ausführung und Programmierung von «If then else» und Schleifenstrukturen gefunden und implementiert werden.



Figur 6 Hardwarearchitektur des im Beispiel verwendeten FFT-Prozessors

● Festgeschaltete Cross-Point-Switch-Verbindung
○ schaltbare Cross-Point-Switch-Verbindung

Literatur

[1] *J.W. Cooley* and *J.W. Tukey:* An algorithm for the machine calculation of complex Fourier series. Mathematics of Computation 19(1965)2. p. 297...301.
[2] *J.V. Zeman:* Synthese und praktische Realisation von Systemen und Algorithmen für digitale Signalverarbeitung. Dissertation Nr. 7857 der ETH Zürich, 1985.
[3] *M. Thaler, C. Loeffler* and *G.S. Moschytz:* Programming, analysis and synthesis of parallel signal processors. Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), Philadelphia, May 4...7, 1987. Vol. 2, p. 378...381.
[4] *M. Thaler:* Analyse und Synthese von parallelen Signalprozessor-Architekturen. Dissertation Nr. 8240 der ETH Zürich, 1987.
[5] *J. Zeman* and *G.S. Moschytz:* Systematic design and programming of signal processors, using project management techniques. IEEE Trans. ASSP 31(1983)6, p. 1536...1549.
[6] *B.R. Rau, C.D. Glaeser* and *R.L. Picard:* Conference Proceedings of the 9th IEEE/ACM Symposium of Computer Architecture 1982; p. 131...139.
[7] *K. Gewald, K. Kasper* und *H. Schelle:* Netzplantechnik. Band 2: Kapazitätsoptimierung. München, Oldenbourg-Verlag, 1972.