Parallelverarbeitung in elektronischen Systemen : eine Übersicht

Autor(en): **Kündig, A.**

Objekttyp: **Article**

Zeitschrift: **Bulletin des Schweizerischen Elektrotechnischen Vereins, des Verbandes Schweizerischer Elektrizitätsunternehmen = Bulletin de l'Association Suisse des Electriciens, de l'Association des Entreprises électriques suisses**

Band (Jahr): **79 (1988)**

Heft 7

PDF erstellt am: **24.05.2024**

Persistenter Link: https://doi.org/10.5169/seals-904013

Nutzungsbedingungen

Die ETH-Bibliothek ist Anbieterin der digitalisierten Zeitschriften. Sie besitzt keine Urheberrechte an den Inhalten der Zeitschriften. Die Rechte liegen in der Regel bei den Herausgebern.
Die auf der Plattform e-periodica veröffentlichten Dokumente stehen für nicht-kommerzielle Zwecke in Lehre und Forschung sowie für die private Nutzung frei zur Verfügung. Einzelne Dateien oder Ausdrucke aus diesem Angebot können zusammen mit diesen Nutzungsbedingungen und den korrekten Herkunftsbezeichnungen weitergegeben werden.
Das Veröffentlichen von Bildern in Print- und Online-Publikationen ist nur mit vorheriger Genehmigung der Rechteinhaber erlaubt. Die systematische Speicherung von Teilen des elektronischen Angebots auf anderen Servern bedarf ebenfalls des schriftlichen Einverständnisses der Rechteinhaber.

Haftungsausschluss

Alle Angaben erfolgen ohne Gewähr für Vollständigkeit oder Richtigkeit. Es wird keine Haftung übernommen für Schäden durch die Verwendung von Informationen aus diesem Online-Angebot oder durch das Fehlen von Informationen. Dies gilt auch für Inhalte Dritter, die über dieses Angebot zugänglich sind.

Ein Dienst der *ETH-Bibliothek*
ETH Zürich, Rämistrasse 101, 8092 Zürich, Schweiz, www.library.ethz.ch

http://www.e-periodica.ch

Parallelverarbeitung in elektronischen Systemen – eine Übersicht

A. Kündig

Methoden der Parallelverarbeitung spielen in elektronischen Systemen eine zunehmend grössere Rolle. Der Beitrag zeigt verschiedene Beweggründe für diese Tendenz auf, so den Wunsch nach Leistungssteigerung, neue technologische Möglichkeiten, aber auch «natürliche Parallelität», wie Gleichzeitigkeit der Ereignisse, Multiplizität der Daten, welche von der Umgebung her an rechnergestützte Steuer- und Regelsysteme herangetragen wird. Anschliessend werden einige wichtige Begriffe erläutert, und es wird ein Ausblick auf die nachfolgenden Beiträge gegeben.

Les méthodes de traitement parallèle jouent un rôle de plus en plus important dans les systèmes électroniques. L'article met en évidence les diverses raisons de cette tendance, par exemple le désir d'augmentation de puissance, de nouvelles possibilités technologiques, mais aussi la «parallélité naturelle» (simultanéité des événements, multiplicité des données) apportée par l'environnement aux systèmes de commande et de réglage assistés par calculateur. Puis suit l'explication de quelques termes importants et une vue sur les articles consécutifs.

Adresse des Autors
Prof. Dr. *Albert Kündig*, Institut für Elektronik, ETH-Zentrum, 8092 Zürich

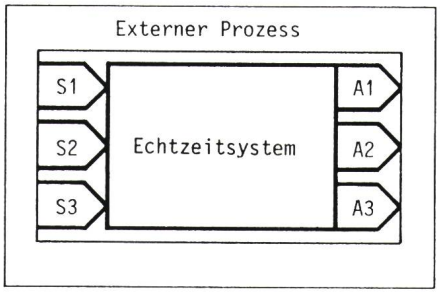
Motive für die Parallelverarbeitung

Anlässlich der Verleihung des prestigereichen *Turing Award* an den Computerpionier *J. Backus* hielt der Geehrte 1978 einen inzwischen vielzitierten Vortrag mit dem Titel «Can Computer Programming be liberated from the von Neumann-Style?» [1]. Backus zeigte, dass die klassische, *von Neumann* zugeschriebene Rechnerarchitektur mit einem Rechenwerk, einem gemeinsamen Daten- und Programmspeicher, einem Steuerwerk sowie einer Einheit für die Ein- und Ausgabe von Daten in seiner Leistungsfähigkeit grundsätzlich beschränkt ist; namentlich der Transfer von Befehlen und Daten zwischen dem Arbeitsspeicher und den übrigen Einheiten – zum Beispiel über einen Bus – legt letztlich die Arbeitsgeschwindigkeit des ganzen Rechners fest. So setzt sich zum Beispiel die Zeit für die Ausführung einer arithmetischen oder logischen Operation aus den folgenden Anteilen zusammen:

- Abrufen des nächsten Befehls (Instruction Fetch),
- Befehlsdecodierung,
- Abrufen der Operanden (Operand Fetch),
- Ausführung der Operation,
- Abspeichern der Resultate.

Die Kritik von Backus ist sachlich ohne Zweifel berechtigt, und sie mag den Anstrengungen, leistungsfähigere neue Rechner zu entwickeln, neuen Auftrieb gegeben haben. Anderseits fasste Backus in bezug auf die Rechnerarchitektur Erkenntnisse prägnant zusammen, die an sich schon lange Zeit vorher bekannt waren und denn auch schon viel früher zu unzähligen Ansätzen für Computerstrukturen führten, deren interne Abläufe einen höheren Grad an Parallelität aufweisen. Wenn darüber hinaus der Titel des Backus-Vortrages zur Assoziation führen würde, dass sozusagen von Neumann mit seinem Rechnerkonzept eine Entwicklung in erfolgversprechendere Richtungen verhindert hätte, so wäre dies völlig fehl am Platz. Erstens hat sich der klassische Rechner bekanntlich gerade wegen seines bestechend einfachen Konzepts millionenfach bewährt – in Applikationen, welche vom Taschenrechner und vom elektronisch gesteuerten Alltagsgerät bis zum Grosscomputer reichen. Und zweitens hat gerade von Neumann vor mehr als dreissig Jahren erste Überlegungen zu dem wieder hochaktuellen Thema von *Neuronennetzen* publiziert [2], einem Ansatz für informationsverarbeitende Maschinen mit hochgradiger Parallelität! Allerdings müssen wir auch den eigentlichen Verdiensten von Backus gerecht werden. Sein provozierender Vortrag zielte weniger auf die Architektur der von-Neumann-Maschine ab, sondern prangerte mit gewissem Recht den dadurch geprägten Programmierstil an. Backus hat daraus die Konsequenzen gezogen und eine neuartige funktionale Programmiersprache (FP) vorgeschlagen.

Das Stichwort Neuronennetze weist auf das wohl natürlichste Motiv für unser Suchen nach parallelen Methoden der Informationsverarbeitung hin: Wo immer wir auch um uns blicken, stellt sich die Natur als ein ausserordentlich komplexes System mit einer Unzahl von gleichzeitigen, meistens wohlkoordinierten Prozessen dar. Was liegt näher, als uns die Natur in der Gestaltung von informationsverarbeitenden Maschinen zum Vorbild zu nehmen! Ganz besonders eindrückliche Beispiele sind bekanntlich im Bereich der Sinnesorgane von Lebewesen zu suchen – man denke nur an das menschliche Auge und den menschlichen Gesichtssinn. Verwandt mit diesen Vorstellungen ist nun aber auch die Problematik, mit welcher sich der Entwickler von rechnergestützten Steuer- und Regelsystemen konfron-



Figur 1 Eingebettetes System
S Sensor
A Aktor

tiert sieht: Auch wenn er seine Anlage oder sein Gerät auf einem klassischen Rechner – also einem Einprozessorsystem – implementiert, werden die umgebenden Prozesse im allgemeinen zu nicht voraussehbaren Zeiten Interaktionen verlangen. Wie die Figur 1 zeigt, muss bei einem solchen *eingebetteten System* der interne Rechenprozess mit den umgebenden (physikalischen, chemischen ...) Prozessen synchronisiert werden. In diesem Falle liegt es nahe, für die Verwirklichung der Steuerung mehrere Rechner vorzusehen, indem zum Beispiel mit peripheren Prozessoren bestimmte *Echtzeitanforderungen* abgedeckt und weniger zeitkritische Aufgaben im Bereiche Bedienung und Datenverwaltung durch separate Prozessoren getragen werden.

Ein eingebettetes System stellt also ein Beispiel für natürliche Parallelität dar, und zwar für Parallelität auf der Ebene von *Prozessen*. Das bereits erwähnte Vorbild des menschlichen Auges aber beruht auf einer anderen Art der natürlichen Parallelität: Parallelität auf der Ebene der *Daten* (Operationen können auf gleichartige, in ein räumliches Gitter eingebundene Datenelemente gleichzeitig angewendet werden).

Neben diesen natürlichen Beweggründen für das Studium der parallelen Informationsverarbeitung gibt es aber auch durchaus handfeste wirtschaftliche und technische Argumente:

- Die immer höheren Ansprüche an die Leistung von datenverarbeitenden Systemen im wissenschaftlichen und technischen Bereich lassen schon lange an Lösungen denken, bei welchen durch eine geeignete Kopplung von *k*-Rechnern eine Verarbeitungsleistung erbracht werden kann, welche möglichst nahe an das k-fache der Leistung eines einzelnen Rechners herankommt (die tatsächlich gemessene Leistungssteigerung wird als Speed-up bezeichnet). Darüber hinaus erhofft man sich von solchen Lösungen, dass bei wachsenden Ansprüchen ein System um zusätzliche Rechner erweitert werden kann (Modularität).
- Bei der Regelung oder Steuerung industrieller Systeme sieht man sich neben Echtzeitproblemen oft mit hohen Anforderungen an die Verfügbarkeit dieser Systeme konfrontiert. Auch in diesem Falle besteht ein naheliegender Ansatz darin, die Verarbeitungsleistung in einer Weise auf parallel arbeitende Subsysteme aufzuteilen, dass ein Ausfall eines dieser Teile höchstens zu einer partiellen Degradation führt. Besonders interessant sind Lösungsansätze, bei welchen die replizierten Subsysteme nicht redundant sind, sondern im Normalbetrieb zur Funktionalität und Leistung des Gesamtsystems einen wesentlichen Beitrag erbringen. Diese Technik der hochverfügbaren Rechnersysteme hat ihrerseits viele wertvolle Anstösse zur Entwicklung auf dem Gebiet der Parallelverarbeitung gegeben.

Argumente gegen die Parallelverarbeitung

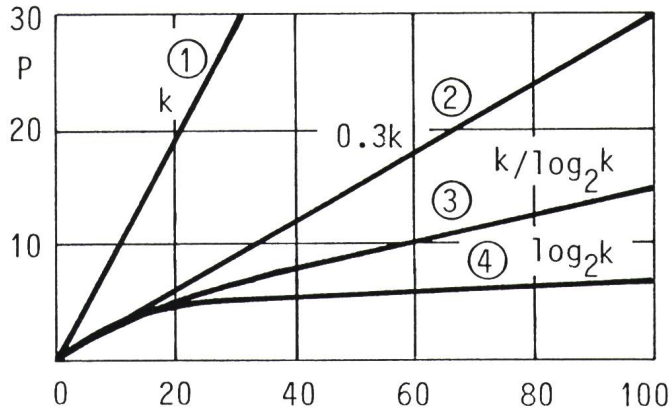
Wir wollen nicht verschweigen, dass die Idee einer Leistungssteigerung in der Informationsverarbeitung dank parallelem Rechnen auch immer wieder der Kritik ausgesetzt war. Die am häufigsten vorgebrachten Argumente können etwa wie folgt zusammengefasst werden:

- Man befürchtet, dass der zusätzliche Aufwand für die Koordination paralleler Rechner den Leistungsgewinn ganz oder teilweise kompensieren könnte.
- Systeme mit parallelen Prozessen sind wesentlich komplexer als solche mit rein sequentieller Datenverarbeitung. Es treten neue Phänomene wie Verklemmungen (Deadlocks) auf; die Synchronisation paralleler Prozesse sowie die wirkungsvolle Verwaltung gemeinsamer Betriebsmittel sind schwierig zu lösende Probleme.
- Die rasche und nach wie vor ungebremste technologische Entwicklung zieht fortlaufende Leistungssteigerungen bei den konventionellen Einprozessorsystemen nach sich. Diese Leistungssteigerungen könnten den Verbesserungen bei den Multiprozessorsystemen noch lange voraneilen.

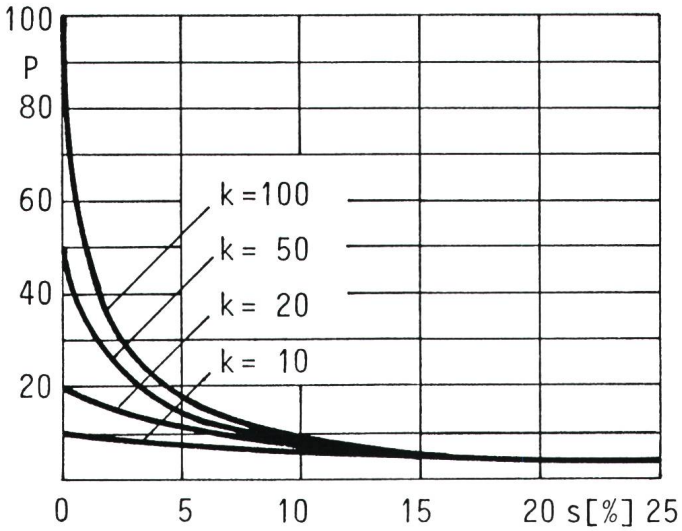
Bekannte Autoren glaubten in der Vergangenheit sogar, diese Kritik in eigentliche Gesetzmässigkeiten zusammenfassen zu können. Wohl das bekannteste dieser Gesetze ist dasjenige von *Minsky* [3], welches wir zusammen mit dem sogenannten 1. Gesetz von *Amdahl* [4] in Figur 2 dem Idealfall eines linearen Speed-up ($\sim k$) sowie praktischen Ergebnissen von *Kuck* bei der Parallelisierung von Fortran gegenüberstellen. Es bleibt dem Leser überlassen, diese Voraussagen anhand der weiteren Beiträge im vorliegenden Heft einer kritischen Würdigung zu unterziehen. Besonders ernst zu nehmen ist allerdings das von Amdahl vorgebrachte Argument, dass in den meisten praktischen Anwendungsfällen ein Teil der Aufgaben gar nicht parallelisierbar ist. Die Figur 3 zeigt sein 2. Gesetz, welches den erzielbaren Gewinn (Speed-up) als Funktion des Anteils *s* von nur sequentiell ausführbaren Programmteilen ausweist.

Einige Begriffe

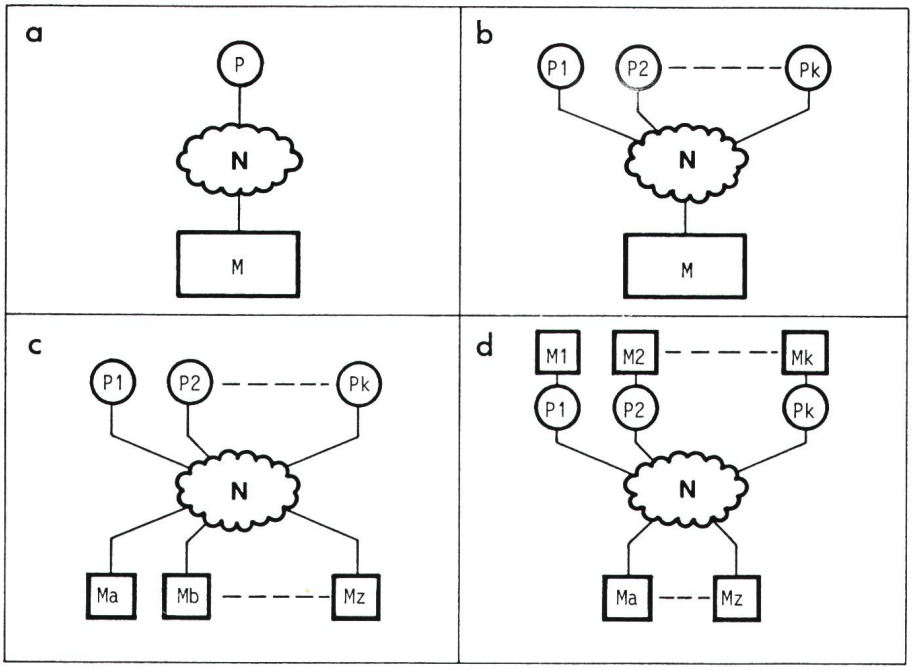
Im folgenden sollen einige bereits erwähnte Begriffe besser definiert und zum Teil auch illustriert werden. Zunächst vergleicht die Figur 4 einige Varianten von Rechnerarchitekturen mit dem klassischen von-Neumann-Rechner, welcher stark vereinfacht in Teil



Figur 2 Systemleistung *P* bei *k* Prozessoren
Verschiedene Prognosen für die Leistungssteigerung durch Parallelisierung. Bezugsgrösse ist die Leistung eines Einprozessorsystems.
① Linearer Speed-up
② Praktisch erreichte Resultate von Kuck
③ 1. Gesetz von Amdahl
④ Gesetz von Minsky



Figur 3
2. Gesetz von Amdahl
$P = [s + (1-s)/k]^{-1}$
s Anteil der nur sequentiell ausführbaren Programmteile
P Systemleistung
k Anzahl Prozessoren



Figur 4 Einige Varianten von Rechnerarchitekturen

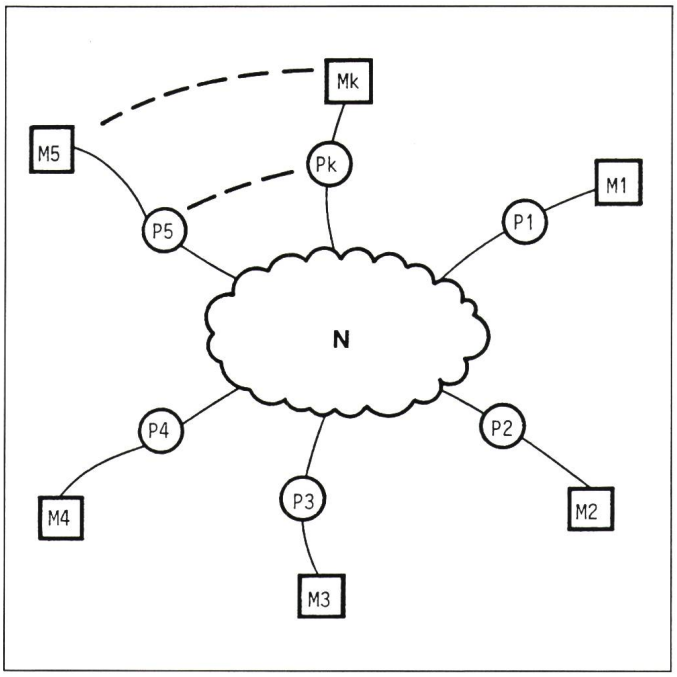
a	Einprozessorsystem
b, c, d	Varianten von Mehrprozessorsystemen mit zunehmender Autonomie der Subsysteme
P1...Pk	Prozessoren
M1...Mk	Lokale Arbeitsspeicher
Ma...Mz	Segmente des zentralen Speichers
N	Kommunikationsnetzwerk

(4a) dargestellt wird. Steuerwerk und Rechenwerk werden in dieser Figur zum Prozessor P zusammengefasst; Programme und Daten befinden sich im Arbeitsspeicher M, welcher vom Prozessor aus über ein «Netzwerk» N zugänglich ist. Netzwerk steht dabei stellvertretend für irgendeine rechnerinterne Kommunikations-Infrastruktur – in den meisten einfachen Fällen wird es sich um ein Bussystem handeln. Die Figur 4b zeigt den klassischen Multiprozessor, bei welchem k-Prozessoren über ein Netzwerk Zugang zu einem gemeinsamen Speicher M besitzen. Damit muss hardware- und softwaremässig sichergestellt werden, dass eine konfliktfreie Benützung dieses Arbeitsspeichers möglich ist; anderseits kann über gemeinsame Speicherbereiche die Kommunikation verschiedener Prozesse mittels globaler Variablen (Shared Variables) bewerkstelligt werden. In Figur 4c ist die Struktur von 4b in dem Sinne weiterentwickelt worden, als durch Aufteilung des Arbeitsspeichers in die Segmente Ma ... Mz ein gleichzeitiger Zugriff auf physisch getrennte Speicherbereiche ermöglicht wird (sofern auch das Netzwerk dies unterstützt). In Figur 4d schliesslich ist jeder Prozessor mit einem eigenen lokalen Arbeitsspeicher ausgerüstet worden; darüber hinaus stehen aber sämtlichen Prozessoren über das Netzwerk die gemeinsam benützbaren Speichersegmente Ma ... Mz zur Verfügung.

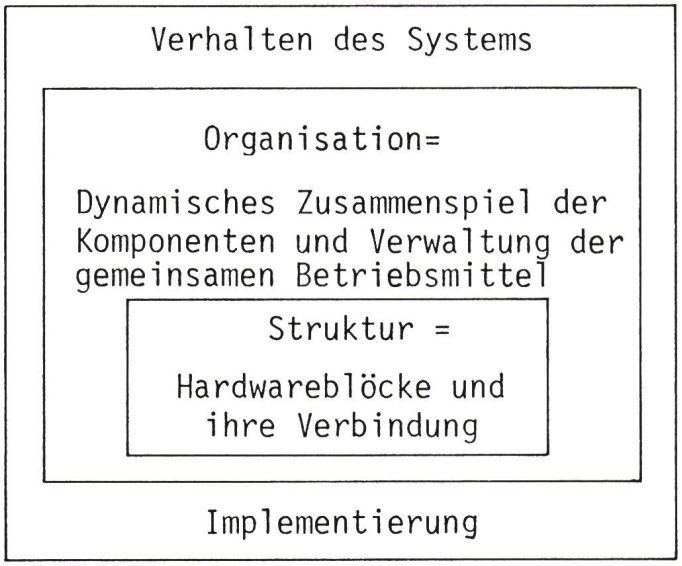
An dieser Stelle muss unbedingt auch der Begriff *Prozess* definiert werden. Wir wollen darunter die schrittweise Abarbeitung eines sequentiellen Programmes verstehen, dabei aber die Möglichkeit vorsehen, dass zwischen verschiedenen Prozessen mittels geeigneter Mechanismen Information ausgetauscht werden kann. Dieser Informationsaustausch kann über gemeinsame Variablen erfolgen; es ist aber auch denkbar, dafür Meldungen (im Sinne von transient vorhandener Information) zu benutzen (sogenanntes *Message Passing*). Dieses zweite Verfahren stellt gerade eines der wesentlichen Merkmale einer weiteren Systemarchitektur dar, des sogenannten *verteilten Systems* (Distributed System).

Die Figur 5 versucht, ein derartiges System zu illustrieren. Von der Hardware her gesehen fällt auf, dass nun kein gemeinsamer Speicher mehr vorhanden ist; die Prozesskommunikation muss also notwendigerweise auf Meldungen abgestützt werden, welche durch das Netzwerk N zu übermitteln sind. Darüber hinaus kann für solche Systeme angenommen werden, dass die Übermittlungszeit für eine Meldung wesentlich grösser ist als ein elementarer Arbeitsschritt auf einem der beteiligten Prozessoren; mithin wird es nicht sinnvoll sein, das Programm so auf die verschiedenen Prozessoren aufzuteilen, dass ein zu häufiger Informationsaustausch notwendig wird. Wir streben also eine Lösung mit *loser Kopplung* der parallelen Prozesse an. Lose Kopplung wird hier im Sinne seltener Interaktionen gebraucht; aus der Sicht zuverlässiger Systeme kann aber lose Kopplung auch bedeuten, dass ein fehlerhafter Prozess keine Störungen bei den andern, korrekt ablaufenden Prozessen provoziert.

Der Vollständigkeit halber wollen



Figur 5
Verteiltes System



Figur 6
Rechnerarchitektur im engeren und weiteren Sinne

wir noch vermerken, dass in deutschen Texten anstelle des Begriffes Parallelität oft der Ausdruck *Nebenläufigkeit* verwendet wird; statt von parallelen Prozessen spricht man dann von nebenläufigen Prozessen.

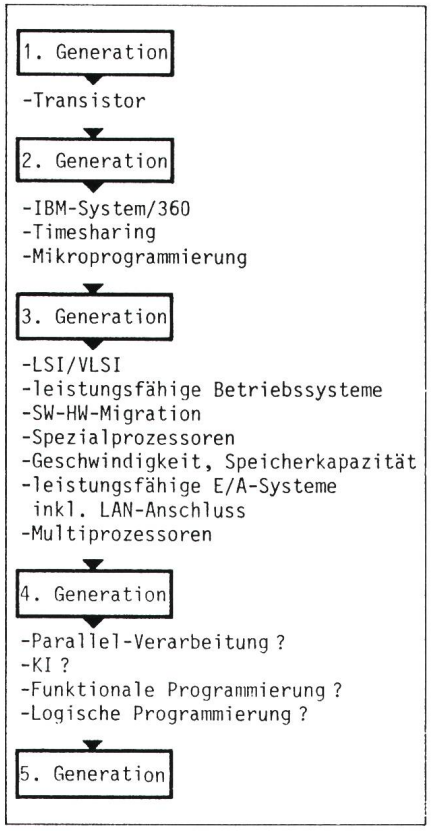
Die bisher gezeigten Systemkonzepte entsprechen natürlicherweise einer Ausnützung von Parallelität auf der Ebene von Prozessen. Typische Vertreter werden im Beitrag [5] dieses Heftes vorgestellt. Demgegenüber sind die Beiträge [6] und [7] eher den Architekturen zuzuordnen, welche die Parallelität der zu verarbeitenden Daten ausnutzen.

Schliesslich scheint auch ein Kommentar zum Begriff *Rechnerarchitektur* angebracht. Wie in Figur 6 gezeigt wird, kann es dabei im engsten Sinne nur gerade um die Hardwarearchitektur gehen, wie sie ja auch den Klassierungsversuchen in den Figuren 4 und 5 zugrunde lag. Diese statische Beschreibung eines Rechnersystems genügt aber bei weitem nicht, um dessen Eigenschaften befriedigend vorauszusagen. Dazu muss auch das darüberliegende *Betriebssystem* bekannt sein, welches unter anderem für die Verwaltung der gemeinsamen Betriebsmittel wie Speicher, Netzwerk und Prozessoren verantwortlich ist. Den Anwender schliesslich interessiert eigentlich nur das Verhalten nach aussen, welches zusätzlich durch die Applikationsprogramme sowie durch die Implementierung in einer bestimten Technologie gegeben wird.

Klassische Formen der parallelen Informationsverarbeitung

Einleitend zeigen wir in Figur 7 eine implizite Definition verschiedener Computergenerationen, indem wir jene neuen Merkmale angeben, welche zum Übergang von einer Generation zu einer nächsten Generation geführt haben. Schon in der 1. Computergeneration wurde versucht, einen Leistungsgewinn durch Parallelverarbeitung zu erzielen. Allerdings handelte es sich um eine sehr primitive Form von Parallelität, bei der Daten- und Adresspfade eine parallele Übermittlung der Bits eines Wortes erlaubten, und auch Steuer- und Rechenwerke für die wortparallele Verarbeitung ausgerüstet wurden. Bereits in der 2. Generation wurde dann versucht, durch geschickte zeitliche Überlappung der bereits früher erwähnten Phasen der Ausführung einer Instruktion einen Zeitgewinn zu realisieren (sogenanntes *Pipelining* auf der Ebene elementarer Operationen). Aber auch vom Benutzer her gesehen ist bereits in den sechziger Jahren Parallelität sichtbar geworden: Sogenannte *Timesharing*-Betriebssysteme erlauben die quasigleichzeitige Benutzung eines Rechners durch mehrere Anwender, indem durch Multiplexierung sozusagen verschiedene virtuelle Maschinen geschaffen werden; jede virtuelle Maschine wird zum Träger eines separaten Anwenderprozesses. Schliesslich zeigen sich erste Ansätze zu eigentlichen Mehrprozessorsystemen, indem die Steuerung und Verwaltung externer Speichermedien (Magnetband und -platte) auf spezielle Prozessoren ausgelagert wird. Wieder andere, heute bei den meisten Rechnersystemen anzutreffende Formen von paralleler oder quasiparalleler Ausführung von Arbeitsschritten betreffen den Einsatz von *virtuellen Speichern* und *Caches*[1], die Verwendung von spezialisierten, dem Rechenwerk beigefügten Verarbeitungseinheiten (zum Beispiel für

[1] Schneller Arbeitsspeicher, welcher jene Programmteile und Daten enthält, auf die am häufigsten zugegriffen wird.



Figur 7
Computergenerationen
Die Fragezeichen beim Übergang von der 4. zur 5. Generation sollen andeuten, dass noch keine allgemein anerkannten Kriterien für einen Computer der 5. Generation existieren.

die Ausführung von Gleitkommaoperationen), den Einsatz bestimmter Verfahren des direkten Speicherzugriffs (Cycle Stealing) usw.

Zwar können alle diese Massnahmen zu einer wesentlichen Leistungssteigerung von Rechnersystemen beitragen, dennoch führen sie aber im Prinzip nicht weg vom Flaschenhals des klassischen Einprozessorsystems – dem «von-Neumann-Bottleneck» von Backus.

Neue Formen der Parallelverarbeitung

Ansätze, welche zum Teil wesentlich weitergehen als die bereits nahezu klassischen Mehrprozessorlösungen von Figur 3, wurden in den letzten 15 Jahren vermehrt zur Diskussion gestellt. Letztlich stehen dahinter 3 verschiedene Beweggründe:

- Es wird nach Architekturen gesucht, welche eine entscheidende Leistungssteigerung versprechen.
- Man möchte die Möglichkeiten der modernen VLSI-Technik möglichst gut ausnutzen.
- Schliesslich – und dies könnte langfristig entscheidend werden – möchte man Rechnerarchitekturen schaffen, die eine besonders effiziente Implementierung von Programmen in neuartigen Programmiersprachen erlauben.

Zu beachten ist, dass die heutigen Technologien es durchaus gestatten, Parallelverarbeitungsmethoden sowohl in Mehrzweckrechnern wie auch in sehr anwendungsspezifischen Systemen (bis hin zu reinen Hardwareimplementationen, beispielsweise im Bereiche der Signalverarbeitung) einzusetzen.

Es stellt sich als recht schwierig heraus, die verschiedenen neuen Rechnerarchitekturen nach einigen wenigen prägnanten Merkmalen zu klassieren. Vor allem zeigt sich, dass an sich bewährte und weitverbreitete Schemas zu wenig ausdrucksfähig sind. Immerhin müssen wir die klassischen Bezeichnungen von *Flynn* [8] erwähnen, welche von den bereits erwähnten Unterschieden hinsichtlich Parallelität der Prozesse und Daten ausgehen. Flynn prägte die folgenden Rechnerklassen:

- SISD: Single Instruction Stream, Single Data Stream
- SIMD: Single Instruction Stream, Multiple Data Stream
- MISD: Multiple Instruction Stream, Single Data Stream
- MIMD: Multiple Instruction Stream, Multiple Data Stream

Wie man unschwer errät, handelt es sich beim SISD-Typ um den klassischen von-Neumann-Rechner, während Rechner im Sinne der Figuren 4b, 4c und 4d sowie Figur 5 als MIMD-Anwärter identifiziert werden können. Rechner des SIMD-Typs findet man im Bereiche der Signalverarbeitung und der Mustererkennung, wo eine einzelne Instruktion auf eine Menge von Datenelementen angewendet wird. Über die praktische Bedeutung der MISD-Klasse gehen die Meinungen hingegen auseinander, da die gleichzeitige Anwendung verschiedener Instruktionen auf das gleiche Datenelement gar nicht konfliktfrei möglich ist. Wenn allerdings eine losere Interpretation zugelassen wird, so könnten die sogenannten Pipeline-Computer dieser Klasse zugerechnet werden. Bei einem Pipeline-Rechner werden mehrere Prozessoren in Serie geschaltet, und ein Datenstrom «fliesst» durch diese Kette von Verarbeitungseinheiten.

Treleaven führte 1982 die folgenden Unterscheidungsmerkmale für die Beschreibung neuartiger Rechnerarchitekturen ein [9]:

- *Control Flow:* Maschinen, bei welchen das Arbeitsprinzip im wesentlichen auf der klassischen sequentiellen Abarbeitung einer Instruktionsfolge mit zentraler Steuerung beruht.
- *Data Driven:* Maschinen mit mehreren elementaren Verarbeitungseinheiten, welche immer dann eine Operation ausführen, sobald alle dafür notwendigen Operanden bereit sind. Es handelt sich also um eine dezentrale Form der Ablaufsteuerung.
- *Demand Driven:* Maschinen, deren Architektur direkt die sogenannte Reduktion von mathematischen Ausdrücken unterstützt.

Maschinen der 2. Kategorie werden normalerweise *Datenfluss-Rechner* genannt (Data Flow Machines). Da einer der folgenden Beiträge [10] eine Einführung in die Funktionsweise solcher Rechner enthält, verzichten wir hier auf weitere Erklärungen. Die Maschinen der dritten Kategorie werden sinngemäss als Reduktionsmaschinen (*Reduction Machines*) bezeichnet. Treleaven hat dafür das Merkmal *Demand Driven* geprägt, weil bei der systematischen Reduktion mathematischer Ausdrücke schrittweise komplexere Terme durch einfachere ersetzt werden – es werden entsprechende Operanden angefordert (demanded) –, bis schliesslich eigentliche Werte verknüpft werden können. In diesem Sinne besteht eine starke Verwandtschaft zur Technik der tabellengesteuerten Syntaxanalyse bei Compilern. Während mit verschiedenen Typen von Datenflussrechnern bereits praktische Erfahrungen gesammelt werden konnten, ist ähnliches über Reduktionsmaschinen kaum bekannt.

Es würde den Rahmen dieser Einführung sprengen, wenn wir auch noch auf die sogenannt *massiv parallelen Rechner* eingehen würden, Rechner also, bei welchen unter Umständen Tausende von Verarbeitungseinheiten in ein spezielles Kommunikationsnetz eingebunden werden. Solche Konzepte sind unter anderem unter den Begriffen *Systolic Arrays* [11] oder Wavefront Processors bekanntgeworden. Spezielle Beachtung findet auch die

bereits kommerzialisierte *Connection Machine* [12] mit 65 536 Prozessoren für Anwendungen im Gebiet der künstlichen Intelligenz.

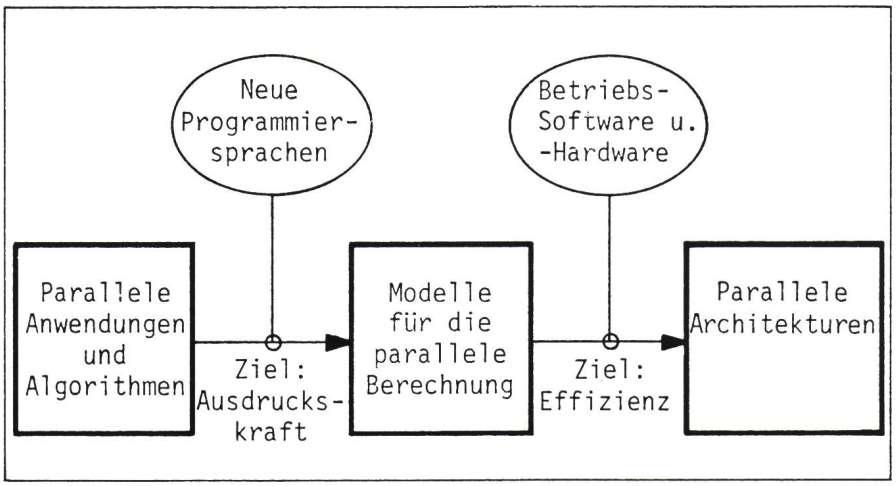
Wo stehen wir heute?

Es fehlt also nicht an vielversprechenden Vorschlägen für neuartige Rechnerarchitekturen. Darüber hinaus ist es namentlich im Bereich anwendungsspezifischer Rechner (z.B. für die Signalverarbeitung) gelungen, einigen der neuen Konzepte auch wirtschaftlich zum Durchbruch zu verhelfen. Dennoch können wir nicht verschweigen, dass noch viel in der Forschung und Entwicklung zu tun bleibt, wenn die radikaleren der neuen Konzepte voll verstanden und beherrscht werden sollen. Dabei genügt es nicht, die neuen Rechner auf der Ebene der Hardwarearchitektur einigermassen zu verstehen; vielmehr muss es um die Beherrschung aller Entwicklungsschritte, von der Formulierung einer neuen Applikation bis zur Implementierung, gehen. In Anlehnung an die Ausführungen in [13] versuchen wir, diese Situation in Figur 8 zu illustrieren:

- Noch ganz wenig beackert ist das ganze Gebiet paralleler Algorithmen. Der klassischen Sammlung von *Knuth* steht in diesem Bereich noch nichts gegenüber!
- Neuartige Algorithmen werden ihrerseits in neuartigen Programmiersprachen mit möglichst hoher Ausdruckskraft formuliert werden müssen. Interessante Vertreter sind zum Beispiel funktionale Programmiersprachen, welche es besonders einfach machen, Parallelität darzustellen (die Summe von zwei Funktionen kann parallel evaluiert werden). Für diese neuen Programmiersprachen sind entsprechende Compiler zu entwickeln.
- Schliesslich wird es auch – abgesehen von sozusagen fest verdrahteten Spezialmaschinen – bei den neuartigen Rechnern eines Betriebssystems bedürfen, welches eine effiziente Ausnutzung der vorhandenen Betriebsmittel für eine breite Klasse von Anwendungsprogrammen erlaubt.

Schlussbemerkungen

Neue Konzepte für die Parallelverarbeitung in elektronischen Systemen haben vor allem im Zusammenhang mit sogenannten Superrechnern Beachtung gefunden. In jüngster Zeit wurde sogar der Begriff «Mini-Superrechner» für eine neue Klasse von preiswerten, kommerziell erhältlichen Parallelrechnern mit bislang unerreichtem Preis-Leistungs-Verhältnis geprägt. Von den Publikationen her könnte man versucht sein, hier einmal mehr die Schweiz in einem technologischen Rückstand zu sehen. Um so erfreulicher ist es, dass im Rahmen der ITG-Informationstagung, welche Gegenstand dieses Heftes ist, einige Forschungs- und Entwicklungsprojekte an Schweizer Hochschulen vorgestellt werden konnten. Besonders interessant scheinen uns die Projekte auch deshalb zu sein, weil von allem Anfang an Anwendungen im industriellen Bereich im Vordergrund stehen.



Figur 8 Forschungsthemen im Bereich Parallelverarbeitung

- Ausdruckskraft: Vermögen einer Sprache, leistungsfähige Konzepte kompakt und dennoch präzis und anwenderfreundlich darzustellen.
- Effizienz: Möglichst gute Ausnutzung der Verarbeitungs- und Speicherelemente.

Literatur

[1] *J. Backus:* Can programming be liberated from the Von Neumann style? Communications of the ACM 21(1978)8, p. 613...641.

[2] *J. Von Neumann:* Probabilistic logics and the synthesis of reliable organisms from unreliable components. Automata studies. Princeton, Princeton University Press, 1956.

[3] *M. Minsky* and *S. Papert:* On some associative, parallel and analog computations. In: Associative information techniques – New York, Elsevier, 1971.

[4] *G.M. Amdahl:* Validity of the single processor approach to achieving large scale computing capabilities. Proceedings AFIPS 30(1967)–, p. 483...485.

[5] *P.G. Kropf:* Paralleles Rechnen mit Transputers. Bull. SEV/VSE 78(1988)7, S. 356...361.

[6] *M. Hufschmid* und *Ch. Löffler:* Modularer Parallelrechner für die Signalverarbeitung. Bull. SEV/VSE 78(1988)7, S. 368...372.

[7] *A. Gunzinger, S. Mathis* und *W. Guggenbühl:* Synchroner Datenflussrechner zur Echtzeitbildverarbeitung. Bull. SEV/VSE 78(1988)7, S. 362...367.

[8] *M.J. Flynn:* Some computer organizations and their effectiveness. IEEE Trans. C-21(1972)9, p. 948...960.

[9] *P.C. Treleaven* a.o.: Data-driven and demand-driven computer architecture. ACM Computing Surveys 14(1982)1, p. 93...143.

[10] *R. Bührer:* Datenflussrechner – Konzepte und Anwendungen. Bull. SEV/VSE 78(1988)7, S. 344...350.

[11a] *J.A.B. Fortes* and *B.W. Wah:* Systolic arrays—from concept to implementation. IEEE Computer 20(1987)7, p. 12...17.

[11b] *J.A.B. Fortes* a.o.: Systolic arrays. A survey of seven projects. IEEE Computer 20(1987)7, p. 91...103.

[12] *D.L. Waltz:* Applications of the connection machine. IEEE Computer 20(1987)1, p. 85...97.

[13] *P.C. Patton:* Multiprocessors—Architecture and applications. IEEE Computer 18(1985)6, p. 29...40.