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: Paralleles Rechnen mit Transputern

Autor: Kropf, P. G.

DOI: https://doi.org/10.5169/seals-904016

Paralleles Rechnen mit Transputern

P.G. Kropf

Viele Probleme, die heute auf sequentiell arbeitenden Computern gelöst werden, besitzen eigentlich eine parallele Struktur. Das Transputer-Occam-Konzept erlaubt nun, derartige Probleme direkt parallel zu implementieren; im Gegensatz zu konventionellen Systemen kann die Parallelität in natürlicher Weise in ein Programm und auf die Architektur abgebildet werden. In diesem Artikel werden die Möglichkeiten der Sprache Occam und der Transputer-Architektur diskutiert. Nach einer Einführung in parallele Systeme und das Transputer-Occam-Konzept werden einige Projekte kurz vorgestellt.

La plupart des problèmes actuellement résolus à l'aide d'ordinateurs travaillant de manière séquentielle sont en fait parallèles par leur nature. Le tandem Occam-Transputer permet maintenant l'implémentation parallèle de tels problèmes, car, contrairement aux systèmes conventionnels, il permet de passer de façon naturelle du problème à un programme puis à une architecture parallèle. Dans cet article, les possibilités du langage Occam et de l'architecture du Transputer sont discutées. Après une courte introduction aux systèmes parallèles et au concept Occam-Transputer un choix de projets est présenté.

Adresse des Autors
Peter G. Kropf, Dipl. Math., Institut für Informatik und angewandte Mathematik, Universität Bern, Länggassstrasse 51, 3012 Bern.

In der Forschung und Entwicklung von parallelen Computersystemen gibt es viele verschiedene Ansätze. Die meisten davon basieren auf dem von den sequentiellen Computern her bekannten von-Neumannschen Prinzip, bei dem der zeitliche Berechnungsablauf durch die Instruktion gesteuert wird. Daneben gibt es andere Entwicklungen, die z.B. auf dem Datenflussprinzip aufbauen, bei dem die Berechnungen durch die Daten gesteuert werden. Die erste Gruppe wird oft nach den in der Tabelle I angegebenen Typen klassifiziert [1; 2]. Die heute kommerziell erhältlichen parallelen Computersysteme, insbesondere die Supercomputer wie etwa Cray oder die Connection Machine, basieren meist auf dem Single Instruction Multipe Data-Typ (SIMD-Typ)[1].

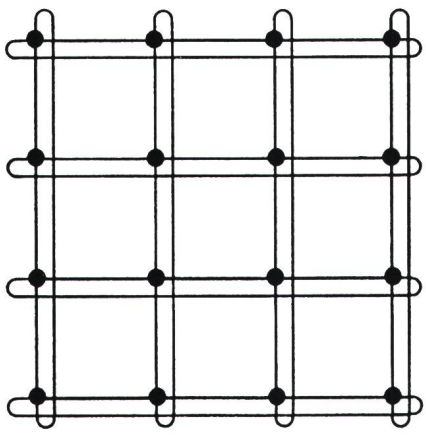
S I M D - Typen	M I M D - Typen	
	Switched	Network

Tabelle I
Klassifikation paralleler Systeme (von-Neumann-Typen)
SIMD Single Instruction Multiple Data
MIMD Multiple Instruction Multiple Data

Das Hauptgewicht bezüglich der Entwicklung von parallelen Rechnersystemen liegt bei den Multiple Instruction Multiple Data-Typen (MIMD-Typen). Die geschalteten (switched) MIMD-Maschinen enthalten gemeinsame Speicher, über die die verschiedenen Prozessoren miteinander kommunizieren und sich gegenseitig synchronisieren. Diese werden als *eng gekoppelte* parallele Systeme bezeichnet. Die zweite MIMD-Klasse, *lose gekoppelte* Netzwerk-Systeme, basieren auf dem Prinzip des Nachrichtenaustausches über ein Netzwerkmedium, das im allgemeinen eine komplexe Struktur (Topologie), wie etwa einen Torus (Fig. 1), aufweist. Jeder Prozessor verfügt über seinen eigenen, privaten Speicher, auf den kein anderer Prozessor direkten Zugriff hat, d.h. es gibt keinen gemeinsamen, globalen Speicher.

Für all die verschiedenen Architekturen gibt es auch passende Programmiersprachen. Man unterscheidet hier zwischen Sprachen, die auf dem Prinzip der gemeinsamen Variablen basieren (SIMD und switched MIMD) und denjenigen, die dem Modell des Nachrichtenaustausches zwischen unabhängigen Systemen entsprechen (MIMD-Networks).

Der im folgenden näher beschriebene Baustein für parallele Computersy-



Figur 1 Torus oder Hyperwürfel

[1] SIMD-Maschinen weisen einen gemeinsamen Speicher auf. Eine Instruktion bearbeitet jeweils gleichzeitig mehrere Daten.

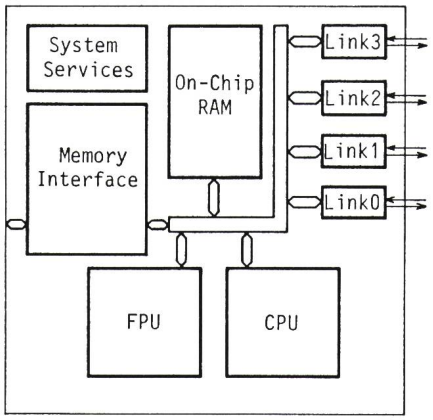
steme, der Transputer, und die parallele Programmiersprache Occam gehören in die Klasse der Netzwerk-MIMD-Systeme.

Die Sprache Occam

Die Sprache Occam [3] basiert auf der theoretischen Sprache CSP (Communicating Sequential Processes) von *C.A.R. Hoare* [4]. Occam ist speziell geeignet für die Programmierung von Prozessornetzwerken, also für parallele Systeme. Ein Occam-Programm besteht aus einer Menge von sequentiellen Teilprozessen, die zueinander parallel ablaufen. Die Teilprozesse können über Kanäle miteinander kommunizieren und Informationen austauschen. Auf diese Weise bearbeiten mehrere parallele Prozesse gemeinsam ein Problem, wobei die Prozesse genau dann kommunizieren müssen, wenn die Fortführung eines Prozesses von einem anderen Prozess abhängt. Die Kommunikation ist synchron, ungepuffert und findet immer über Punkt-zu-Punkt-Verbindungen (sog. Kanäle) statt. Parallele Prozesse können nur über Kanäle Daten austauschen (Tab. II), d.h. es gibt insbesondere keine gemeinsamen, globalen Datenbereiche (Shared Variables). Im sequentiellen Teil der Sprache findet man ausser dynamischen Strukturen etwa dieselben Sprachelemente wie in Pascal, während der parallele Teil Sprachelemente für die Kommunikation (Senden und Empfangen) (Tab. III), Kanalprotokolle und die Parallelität enthält. Im Gegensatz zu anderen Sprachen, wie Ada oder Chill, ist bei Occam die Parallelität das grundlegende Konzept und der sequentielle Fall nur ein Spezialfall [5].

Der Transputer

Der Transputer (Fig. 2) gestattet, die parallelen Occam-Programme optimal auf einer parallelen Hardware auszuführen [6; 7]. Dieser 32-Bit-Prozessor ist ein schneller RISC[2]-Rechner mit einer Leistung von 10 MIPS[3] in der Grundversion (T414) und zusätzlich 1,5 MFLOPS[3] in der Floating Point Version (T800). Im weiteren sind beim T414 2 kByte RAM und beim T800 4 kByte RAM auf dem Chip integriert. Ein entscheidender Unterschied zu anderen herkömmlichen Mikroprozessoren sind die 4 bidirektionalen seriellen *Links* (20 Mbit/s), über welche verschiedene Transputer miteinander verbunden werden können. Diese Links entsprechen den Occam-Kanälen. Softwarekanäle können damit direkt auf Hardwarekanälen (Links) abgebildet werden. Dabei realisiert jeder Link zwei gerichtete Occam-Kanäle, einen in jede Richtung. Diese Kanäle können auch generell für die Kommunikation mit der Aussenwelt verwendet werden, indem herkömmliche (parallele oder serielle) Schnittstellen daran angeschlossen werden. Dank diesem einheitlichen Hardware-Software-Konzept kann z.B. eine Prozesssteuerung ganz in einer höheren Programmiersprache implementiert werden, ohne dass auf Assembler- oder Betriebssystemebene zurückgegriffen werden muss. Die Kommunikationsfähigkeiten des Transputers machen diesen insbesondere als Komponente für den Aufbau von komplexen parallelen Systemen (Fig. 1) bis zur Leistungsfähigkeit von Supercomputern geeignet. Im Gegensatz zu klassischen



Figur 2 Der Inmos-Transputer
FPU Floating Point Unit
CPU Central Processing Unit

```
CHAN OF INT from.A.to.B :  -- Kanal Deklaration --

PAR                        -- Parallele Ausfuehrung --

  -- Prozess 'A'
  INT x :                  -- Variablen Deklaration --
  SEQ
    --compute
    kanal ! x              -- sende 'x' ueber Kanal
    --compute                  'from.A.to.B' zu Prozess 'B' --

  -- Prozess 'B'
  INT y :
  SEQ
    --compute
    kanal ? y              -- empfange von Prozess 'A' ueber
    --compute                  Kanal 'from.A.to.B einen Wert
                               und speichere ihn in 'y'      --
```

Tabelle II Kommunizierende Prozesse
Das Einrücken bestimmt in Occam die Blockbildung und die Gültigkeitsbereiche von Objekten.

Datentypen	BOOL BYTE INT REAL 32	Wert TRUE,FALSE Wert 0 bis 255 Integer Floating Point Zahlen
Konstrukte	SEQ PAR IF ALT WHILE	Sequenz parallel Verzweigung Input Alternative Schleife
Kanäle, Kommunikation	CHAN OF protocol keyboard? char screen! char	Kanaltyp Kanalinput Kanaloutput

Tabelle III Elemente der Sprache Occam

[2] RISC Reduced Instruction Set Computer

[3] MIPS Millionen Instruktionen pro Sekunde, MFLOPS Millionen Gleitkommaoperationen pro Sekunde

Supercomputern sind derartige Systeme allerdings wesentlich kleiner und kostengünstiger.

Occam-Transputer-Programmierung

Zur Entwicklung von Occam-Programmen auf Transputern steht dem Benutzer ein integriertes System mit Editor, Compiler, Configurator und Bibliotheken zur Verfügung. Dieses Entwicklungssystem enthält auch Hilfsmittel zur separaten Kompilation von Prozeduren, zum Laden von Transputernetzwerken über Hostrechner (z.B. IBM-PC, Sun, Vax), zur Einbindung von Programmteilen, die in anderen Sprachen geschrieben sind (C, Pascal, Fortran) und zur Erstellung von Eprom. Für die Fehlersuche steht ein symbolischer Debugger zur Verfügung. Dieses von Inmos entwikkelte Programmiersystem, das Transputer Development System (TDS), hat sich in unseren Projekten bewährt. Es gibt jedoch auch Stand-alone-Occam-Compiler, bei denen die Umgebung von Hostrechnern (Editoren usw.) verwendet werden kann. Das TDS erfordert einen relativ hohen Einarbeitungsaufwand, bietet aber gegenüber den Stand-alone-Compilern einen wesentlich höheren Komfort (insbesondere beim Editieren, bei der separaten Kompilation und der Fehlersuche).

Weitere Hilfsmittel sind an der Universität Bern in Entwicklung, u.a. ein Werkzeug zur automatischen Konfiguration von Occam-Programmen für ein Netzwerk von Transputern unter Berücksichtigung minimaler Kommunikationskosten und optimaler Prozessorauslastung [8], ein grafisches Werkzeug zur Entwicklung von Occam-Programmen. Compiler für andere Sprachen wie Prolog, Lisp oder ADA existieren bereits oder sind in Entwicklung.

Projekte

Seit Beginn des Einsatzes von Transputern und Occam besteht eine enge Zusammenarbeit zwischen dem Institut für Informatik und angewandte Mathematik (IAM), der Ingenieurschule Bern (ISB) und der Software-Schule Schweiz in Bern (SWS). Am IAM sowie auch an der ISB/SWS wird Occam schon seit längerem in der Ausbildung eingesetzt. In den nachfolgenden Kapiteln werden einige der bisher durchgeführten und laufenden Projekte kurz beschrieben.

1. Prozesssteuerungen

Seit Beginn der Aktivitäten mit Transputern und Occam am IAM werden Praktika für Studenten in den Grundsemestern durchgeführt. In Gruppen (5 bis 7 Studenten) werden kleine Projekte über ein Semester lang realisiert. Das Ziel dieser Arbeiten ist die Entwicklung einer Prozesssteuerung vom Design bis zur Implementierung. Die Aufgabe besteht darin, eine Modelleisenbahnanlage oder einen Modellift zu steuern. Den verschiedenen Gruppen stehen als Steuerungssysteme wahlweise Transputer, AIM-65-Systeme oder IBM-PCs zur Verfügung.

Um die Modelle direkt von einem Transputer ansprechen zu können, wurden parallele und serielle Schnittstellen entwickelt, die an einen Transputer-Link angeschlossen werden können. Damit wird die Applikation vom Hostsystem unabhängig.

Als Design-Hilfsmittel werden die State-Event-Technik und Petri-Netze verwendet. Die Umsetzung des Designs in Occam gestaltet sich wesentlich einfacher als bei anderen Sprachen (Macro Assembler, Pascal), da die bei diesen Aufgaben vorhandene Parallelität direkt auf die Software abgebildet werden kann. Die Verwendung von Occam erlaubt, alle Funktionen der Steuerung in einer Hochsprache zu formulieren, ohne dass auf Systemroutinen oder Assembler zurückgegriffen werden muss. Beispielsweise sind die Eingabefunktionen der Schnittstellen als Prozesse formuliert, die einen Kanal abfragen, ob Daten anliegen (Tab. IV). Dabei werden aber keine kontinuierlichen Lese-Instruktionen ausgeführt, wie es beim Polling[4] üblich ist, sondern der Prozess, der die Abfrage auf dem entsprechenden Kanal macht, wartet passiv, bis ein Datum anliegt. Die Geschwindigkeit des Transputers und sein interner Scheduling-Algorithmus mit einer Context Switching Time von unter 1 µs garantieren, dass keine anliegenden Daten unberücksichtigt bleiben. Anstelle der üblichen Unterbrechungssteuerung tritt also ein Pseudo-Polling.

In Tabelle V ist die äusserste Ebene eines Steuerungsprogramms dargestellt. Der Prozess *process.control* bearbeitet die Steuerung, und der Prozess *input.output* ist verantwortlich für das Pseudo-Polling auf den Kanälen *to.interface* und *from.interface*, die durch die PLACE-Anweisungen an einen Transputer-Link gebunden sind. Dieser Link ist über eine Schnittstelle mit der Anlage verbunden.

2. Monte-Carlo-Simulationen

Seit dem Frühling 1987 ist am IAM ein Forschungsprojekt im Gange, welches sich mit Anwendungen auf neuartigen parallelen und vektoriellen Architekuren befasst. Der anwendungsorientierte Aspekt steht dabei im Vordergrund, wobei insbesondere naturwissenschaftliche Probleme behandelt werden.

Das Projekt umfasst die Entwicklung und Untersuchung von parallelen Algorithmen für naturwissenschaftliche Probleme und die Bildverarbeitung, die Implementierung dieser Algorithmen auf verschiedenen Architekturen und die Entwicklung von Werkzeugen zum effizienten Einsatz von parallelen Systemen. Bisher wurden vektorielle Rechner (Cray) sowie kleine Transputernetzwerke benutzt.

```
WHILE TRUE
  interface.in ? data
  -- send data to other process
```

Tabelle IV Kanalabfrage, ein Pseudo-Polling-Prozess

[4] Abfragen einer Eingabestelle zu definierten Zeitpunkten

Tabelle V Struktur der Steuerung

```
CHAN OF ANY in,out :
CHAN OF BYTE to.interface, from.interface :
PLACE to.interface AT link.adress.0.out :
PLACE from.interface AT link.adress.0.in :
PAR
  process.control(in,out)
  input.output(in,out,to.interface,from.interface)
```

Künftig sollen auch andere Architekturen einbezogen werden (z.B. Suprenum). Ausser der kleinen Transputerinstallation (20 Prozessoren) und Entwicklungsstationen (Sun, IBM-PC) stehen dem Projekt keine eigenen Ressourcen zur Verfügung. Herkömmliche Supercomputer wie die Cray sind jedoch mittels Telekommunikationsverbindungen (Telepac) bei anderen Institutionen verfügbar.

Bisher wurden insbesondere Algorithmen für Monte-Carlo-Simulationen in der Gittereichtheorie betrachtet. Verschiedene vektorielle und parallele Implementierungen wurden vorgenommen und auf ihre Effizienz hin untersucht [9; 10].

Diese Monte-Carlo-Simulationen wurden auch sequentiell in Fortran 77 und Occam implementiert und auf verschiedenen Maschinen getestet. Ein Vergleich der Laufzeit-Grössenordnungen für diese Programme ist in Tabelle VI wiedergegeben. Die parallele Version für ein Transputernetzwerk erbrachte einen linearen Speedup, d.h. mit 3 Transputern wurde auch eine etwa 3mal schnellere Ausführungszeit gemessen. Die bisherigen Untersuchungen deuten darauf hin, dass mit einem wesentlich grösseren Transputernetzwerk ebenfalls ein linearer Speedup erreicht werden kann.

3. Paralleler Lisp-Interpreter

Ein funktionales Programm entspricht einem Algorithmus mit einem Input, und seine Ausführung besteht in der Auswertung eines Ausdruckes, der sukzessive bis zur Normalform reduziert wird. Die Auswertungen von Teilausdrücken können kausal unabhängig voneinander sein und damit parallel ausgeführt werden. In einer funktionalen Sprache geschriebene Programme weisen oft implizite Parallelität auf. Die Sprache Lisp ist die am weitesten verbreitete funktionale Programmiersprache.

Im Rahmen einer Lizentiatsarbeit wird am IAM ein Lisp-Interpreter für ein paralleles Transputernetzwerk realisiert. Vorrangiges Ziel dieser Arbeit ist, ein geeignetes *Speicherverwaltungskonzept* und einen *Garbage-Collection-Algorithmus* zu finden, welche das Einbinden eines Multiprozess-Lisp-Interpreters für die parallele Auswertung von Ausdrücken ermöglichen sollen. Zu diesem Zweck stehen zurzeit ein Transputer-Entwicklungssystem (PC-AT mit Transputerboard und 2 MByte Memory) sowie ein externes Board (4 Transputer mit je 256 KByte) zur Verfügung. Zweckmässigerweise wird ein Konzept mit Speichersegmentierung verwendet, um den lokalen Speicher der Transputer-Umgebung möglichst gut zu nutzen. Der Garbage-Collection-Algorithmus wird diesem Konzept angepasst und auf einem eigenen Transputer ausgeführt. Schliesslich wird ein Interpreter mit minimalem Befehlssatz zu Testzwekken implementiert.

Aufbauend auf dieser Umgebung kann ein Interpreter mit grösserem Funktionsumfang (beispielsweise Common Lisp) oder ein Multiprozess-Interpreter für ein Transputer-Netzwerk entwickelt werden. Schliesslich besteht auch die Möglichkeit, einen Lisp-Compiler in diese Umgebung zu integrieren. Als Basis für den zu Testzwecken benötigten Interpreter wurde XLisp gewählt. Da ein XLisp-Interpreter öffentlich zur Verfügung steht[5], wurde dieser zu Vergleichszwecken auf einen Transputer portiert.

4. X.25-Implementation

Das X.25-Protokoll ist einer der am weitesten verbreiteten Standards für die digitale Kommunikation in der Industrie und in den öffentlichen Netzen. In der kurzen Entwicklungszeit von einem halben Jahr wurde das X.25-Kommunikationsprotokoll in einer möglichst natürlichen Art und Weise implementiert [11]. Die in seiner Definition vorhandenen parallelen Strukturen und der State-Event-Charakter wurden direkt als System von kommunizierenden Prozessen realisiert. Die OSI-Schichten im X.25-Protokoll werden in der Implementierung direkt durch die Occam-Prozess- und Kanalstrukturen reflektiert (Fig. 3). Dadurch wird der übliche grosse Graben zwischen Spezifikation und Implementation wesentlich verringert.

Um aussagekräftige Tests machen zu können, wurde eine passende Benutzerschnittstelle (OSI-Schichten 4 bis 7) realisiert. Teil des Projektes war auch die Untersuchung der Anwendbarkeit der besonderen Kommunikationsfähigkeiten des Transputers (4 Links, die von separaten, unabhängigen DMA-Controllern getrieben werden) sowie der Möglichkeiten, die Software gleichmässig bezüglich Belastung auf verschiedene, miteinander verbundene Prozessoren zu verteilen. Beim letzteren hat sich gezeigt, dass die Definition des Protokolls selbst eine Leistungssteigerung verunmöglicht, da dieses mehrheitlich einen streng sequentiellen Ablauf vorsieht. Der grosse Vorteil, den die Verwendung der parallelen Occam-Prozesse bringt, liegt mehr in den guten Formulierungsmöglichkeiten für die Abhandlung des Protokolls als in den Möglichkeiten der echt parallelen Ausführung.

Die Tests in einer simulierten Umgebung, d.h. ohne Verbindung zur Aussenwelt, jedoch mit zwei durch einen Link verbundenen Transputersystemen, sind erfolgreich abgeschlossen worden. Mit einem 15-MHz-Transputer, einem externen RAM mit 200 ns Zugriffszeit und 10 Mbit/s-Links wurde eine Übertragungsleistung von 12 Paketen zu 128 Byte pro Sekunde erreicht. Diese Leistung erscheint auf den ersten Blick nicht überragend zu sein. Es ist aber zu betonen, dass die gesamte X.25-Protokollabhandlung softwaremässig implementiert ist und auf *einem* Transputer abläuft. Die zeitaufwendigen Funktionen des Layers 2, die in den üblichen Implementationen immer hardwaremässig realisiert sind, werden hier vollständig von der Software durchgeführt.

Der Einsatz eines schnelleren Transputers und Speichern sowie die Programmoptimierungen werden noch eine wesentliche Leistungssteigerung bringen. Ausserdem ist zu berücksichtigen, dass für Testzwecke der Bit-Strom auf Schicht 1 durch eine byteweise Übertragung realisiert wurde, d.h. für jedes einzelne Bit in einem Paket wird ein Byte über den Transputer-Link übertragen. Gegenwärtig wird an den Schnittstellen zur Aussenwelt gearbeitet, um das X.25-System mit und ohne Verwendung eines dazwischengeschalteten Modems testen zu können.

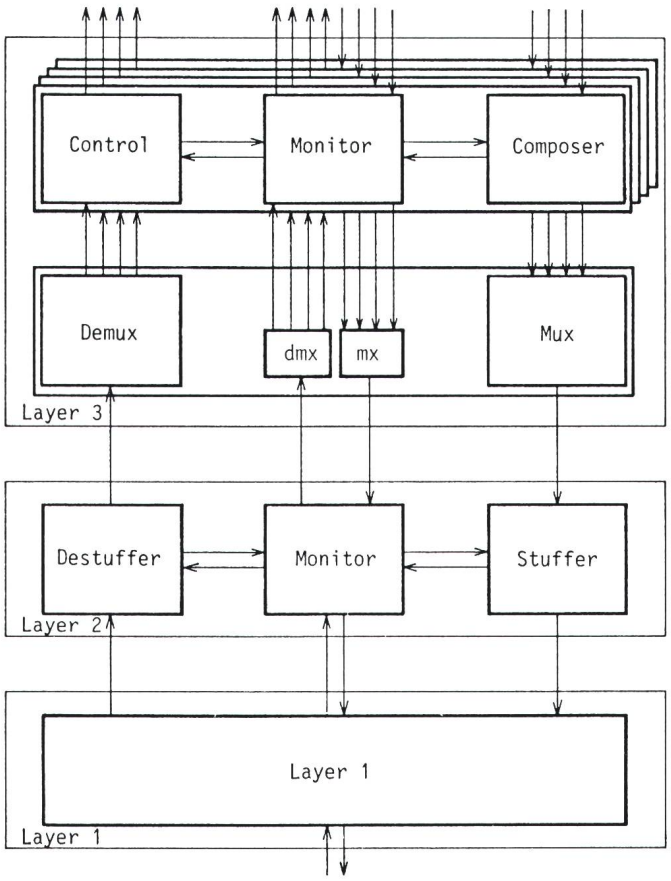
[5] In der Sprache C geschriebene Public Domain Software

Cray X-MP/416	T800	3 T800	Sun-3/FPA	VAX 8530
1	142	46	130	87

Tabelle VI
Leistungsvergleich

5. Kürzeste Wege in einem Graphen

Der Algorithmus von *Dijkstra* [12] zum Auffinden des kürzesten Weges



Figur 3
X.25: OSI-Schichten und dazugehörige Occam-Prozesse

Layer 1, physikalische Bitübertragungsschicht

Layer 2, Sicherungsschicht: Die Prozesse Destuffer und Stuffer entfernen bzw. fügen den Paketen Kontrollinformationen für die Übertragung hinzu. Im Prozess Monitor werden Statusinformationen verwaltet und Fehlerkorrekturen vorgenommen.

Layer 3, Netzwerkschicht: Mx, Mux, dmx, Demux sind Multiplexer bzw. Demultiplexerprozesse. Die anderen Prozesse des Layers 3 sind jedem Benutzer (logischen Kanal) einzeln zugeordnet: Die Prozesse Control senden Pakete mit Benutzerdaten an die oberen Schichten. In den Prozessen Composer werden die Pakete zusammengestellt und gekennzeichnet. Die Monitor-Prozesse sind für den Kontrollfluss verantwortlich.

```
[2]CHAN OF INT master.to slave, slave.to.master :
PAR
  SEQ      -- master process
    PAR
      master.to.slave[0] ! nodes
      master.to.slave[1] ! nodes
    WHILE not.finished
      SEQ
        PAR
          slave.to.master[0] ? data
          slave.to.master[1] ? data
        -- compute

  SEQ     -- slave.0
    master.to.slave[0] ? nodes
    WHILE not.finished
      SEQ
        -- compute
        slave.to.master[0] ! result

  SEQ  -- slave.1
    master.to.slave[1] ? nodes
    WHILE not.finished
      SEQ
        -- compute
        slave.to.master[1] ! result
```

Tabelle VII
Master-Slave-Programm

Die hier ausformulierten Prozesse können auch als Prozeduren definiert werden und dann parallel aufgerufen werden. Die Slave-Prozesse werden damit nur einmal (parametrisiert) definiert.

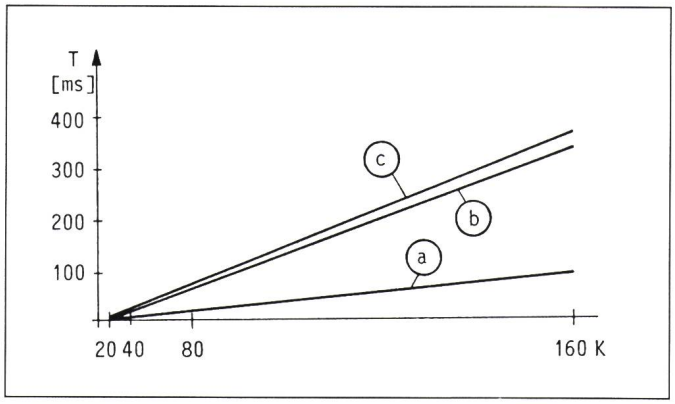
von irgendeinem Knoten zu allen anderen Knoten in einem gerichteten Graphen ist für eine Parallelisierung gut geeignet. Der Aufwand des Algorithmus sinkt linear im Verhältnis der eingesetzten Prozessoren, falls die Anzahl der Prozessoren viel kleiner als die der Knoten ist. Der parallele Algorithmus basiert auf einem einfachen *Master-Slave-Prinzip:* jeder Slaveprozess bearbeitet einen Teil der Knoten des Graphen, und der Masterprozess liest dann jeweils aus der Menge der von den Slaveprozessen vorgeschlagenen Knoten denjenigen aus, für welchen der kürzeste Weg gefunden worden ist. In einem Initialisierungsschritt werden die benötigten Daten den einzelnen Prozessen zugesandt, d.h. der Datenbereich wird über das Netzwerk verteilt, so dass jeder Prozess direkten Zugriff zu den Daten hat, die er bearbeitet. Während des Programmablaufs kommunizieren die Prozesse miteinander, um sich gegenseitig abzustimmen (Tab. VII).

Da ein Transputer nur vier Links besitzt, kann die Master-Slave-Struktur nicht um beliebig viele Slaves erweitert werden. Um mehr Slaves bei der Bearbeitung mithelfen zu lassen, kann z.B. eine ternäre Baumstruktur eingeführt werden, bei der die Slaves über die dazwischenliegenden Knoten mit dem Master kommunizieren.

Zeitmessungen haben den theoretischen Geschwindigkeitsgewinn (Speed-up) bestätigt (Fig. 4). Da bei diesem Programm, je nach Anzahl Knoten im Graphen, grosse Datenmengen transferiert werden, kommt den Kommunikationsmöglichkeiten des Transputers besondere Bedeutung zu. Die Daten können parallel transferiert werden, weil jeder Transputer-Link von einem separaten DMA-Kontroller betrieben wird. Dies bedeutet, dass für eine Konfiguration mit einem Master und drei Slaves für die Verteilung von 160 Knoten (etwa 100 KByte) nur die Zeit für einen vergleichbaren Datentransfer in der Grösse von ungefähr 35 KByte gebraucht wird.

Zusammenfassung

Das Transputer-Occam-Konzept hat sich bisher in allen Projekten des IAM sowohl in der Ausbildung als auch in der Forschung bewährt. Die Modularität des Systems ermöglicht einen sehr flexiblen Einsatz der (bis jetzt) 20 Prozessoren. Für die Programmentwicklung werden Transputer-Einschubkarten für PCs verwen-



Figur 4
Laufzeitvergleich

T Rechenzeit
K Anzahl Knoten des zu verrechnenden Graphen
ⓐ Paralleles Programm (1 Master-, 3 Slave-Prozesse), Ausführung auf 4 Prozessoren
ⓑ Sequentielles Programm (1 Prozess), Ausführung auf 1 Prozessor
ⓒ Paralleles Programm (1 Master-, 3 Slave-Prozesse), Ausführung auf 1 Prozessor

det. Es steht damit ein äusserst kostengünstiges paralleles System zur Verfügung, das erlaubt, parallele Programme nicht nur zu simulieren, sondern auch echt parallel auszuführen. Wegen der freien Konfigurierbarkeit eines Transputernetzwerkes können für verschiedene Applikationen jeweils die am besten passenden Topologien zusammengestellt werden. Neben der Sprache Occam wurden auch Programme in Pascal, C und Fortran 77 für einzelne und mehrere Transputer geschrieben. Optimal lässt sich die Parallelität jedoch nur mit Occam nutzen. Diese Sprache verfügt über alle üblichen sequentiellen Kontroll- und Datenstrukturen ausser den dynamischen Konstrukten (Rekursion und Pointers). Rekursionen lassen sich allerdings einfach mittels Pipelines von parallelen Prozessen nachbilden. Als weiteres häufiges Anwendungsgebiet für Transputer, das hier nicht diskutiert wurde, sind die Bildverarbeitung und die grafische Datenverarbeitung zu erwähnen. Beim IAM ist z.B. ein Projekt zur Implementierung eines Grafikpakets im Gang, bei dem ein Transputer in einem Netzwerk als Grafikprozessor eingesetzt wird.

Zusammenfassend kann gesagt werden, dass das Tandem Occam-Transputer ein mächtiges Mittel bei der Entwicklung paralleler Systeme ist, das für die Zukunft noch ungeahnte Möglichkeiten beinhaltet.

Literatur

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

[2] *R.W. Hockney* and *C.R. Jesshope:* Parallel computers. Architecture, programming and algorithms. Bristol, A. Hilger, 1981.

[3] Occam 2 reference manual. London a.o., Prentice-Hall, 1988.

[4] *C.A. Hoare:* Communicating sequential processes. EnglewoodCliffs/N.J., Prentice-Hall, 1985.

[5] *P.G. Kropf:* A comparison between the language Chill and Occam. Proceedings of the 4th Chill Conference, Munich, October 1986; p. 145...152.

[6] *Transputer reference manual. Bristol, Inmos Ltd., October 1986.*

[7a] *P. Eckelmann:* Transputer der 2. Generation. 1. Teil: Architektur und Merkmale. Elektronik 36(1987)18, S. 61...70.

[7b] *P. Eckelmann:* Transputer der 2. Generation. 2. Teil: Leistungsuntersuchungen und Benchmark-Programme. Elektronik 36(1987)19, S. 129...136.

[7c] *P. Eckelmann:* Transputer der 2. Generation. 3. Teil: Hardware- und Software-Hilfsmittel für die Anwendung. Elektronik 36(1987)20, S. 86...93.

[8] *J.E. Boillat* a.o.: An analysis and reconfiguration tool for mapping parallel programs onto transputer networks. Proceedings of the 7th Occam User Group Meeting (OUG), Grenoble, September 1987.

[9] *K. Decker* and *P.G. Kropf:* Vectorized and parallelized Monte Carlo algorithms for lattice gauge theory problems. Parcom Project Preprint. Berne, University of Berne, January 1988.

[10] *K. Decker:* Monte Carlo simulations for lattice gauge theory. Parcom Project Preprint. Berne, University of Berne, February 1988.

[11] *D. Bärtschi* a.o.: Communication protocols and concurrency: An Occam implementation of X.25. Proceedings of the 1988 International Zürich Seminar on Digital Communication Systems. Zurich, March 1988.

[12] *E.W. Dijkstra:* A note on two problems in connexion with graphs. Numerische Mathematik 1(1959)-, p. 269...271.