# Die 90er Jahre: Eine neue Aera des Supercomputings*

J.E. Boillat
Institut für Informatik und angewandte Mathematik
Universität Bern
Länggassstrasse 51
3012 BERN

## 1 Einführung

Während 15 Jahren ist der Parallelrechnermarkt von Vektorrechnern beherrscht worden (CRAY, CDC/ETA, NEC, FUJITSU, HITACHI). Vektorrechner bestehen aus wenigen Prozessoren und haben alle etwa dieselbe Architektur [3]. Der Erfolg dieser Rechner lässt sich durch die einfache Anpassung der Compiler am rasanten Fortschritt der Integrationstechnologie erklären. In dieser Zeitspanne hat man auch versucht, sogenannte *automatisch parallelisierende Compiler* zu entwickeln. Die erreichten Resultate sind minim: die automatische Parallelisierung ist zur Zeit nur in ganz speziellen Fällen möglich wie z.B. Mikrotasking von Schleifen. Heute, am Anfang der 90er Jahre, werden die Vektorrechner viel schneller als erwartet durch massiv parallele Systeme abgelöst [4]:

- In 1989 hat zum ersten Mal ein massiv paralleler Rechner den berühmten *Gordon Bell Award* [5][6] für den grössten Speedup gewonnen. Die Hegemonie der Vektorrechner wurde von der *Connection Machine CM-2* mit 64K Prozessoren gebrochen. Dieser Rechner erreichte eine Leistung von 5.6 GFlops[1] gegenüber etwa 1 GFlops für den schnellsten Vektorrechner. Ein Jahr später steigerte sich die Leistung der Connection Machine auf 14 GFlops; die Vektorrechner dagegen konnten im selben Jahr nur etwa 1.5 GFlops erreichen.
- Traditionnelle Vektorrechner werden viel schneller als erwartet durch massiv parallele Rechner abgelöst. In vielen Forschungsstätten wird fast ausschliesslich mit diesen neuen Rechnern gearbeitet; wie etwa in den *Sandia National Laboratories* in New Mexico, USA [7].
- Die Finanzierung der Forschung und Entwicklung im Bereich der massiv parallelen Rechner hat in vielen Länder stark zugenommen:
  - Das *Federal High Performance Computing Program* in den vereinigten Staaten hat ein 5 Jahres-Budget von 2'000 M$.

*Dieser Artikel basiert auf der Vorlesung *Paralleles Rechnen* [1], sowie auf der Dissertation *Parallele Algorithmen für Mehrprozessorsysteme mit Verteiltem Speicher* von Peter G. Kropf [2]. Ich danke Peter Kropf für die Mitarbeit.

[1] GFlops Billion Floating Point Instructions per second

- Die japanischen Vektorrechner-Hersteller sind noch nicht in die Technologie der massiv parallelen Rechner eingestiegen. Allerdings wird zur Zeit die Forschung von der Regierung neu organisiert.
- Andere asiatische Länder haben ehrgeizige Projekte gestartet, wie etwa das *Center for Development of Advanced Computing* [8] in Indien.
- In Europa sind mehrere Projekte realisiert oder geplant: *Grand Challenge Collaboration* (UK), *Suprenum* (BRD), *Genesys-P Project* (EG), *European TeraFlop Initiative* (EC), etc. Sogar die Schweiz hat ein Forschungsprogramm im Bereich der massiv parallelen Rechner angekündigt (Schwerpunktprogramm Informatik).

Hochleistungsrechner sind überall gefragt, und die Nachfrage steigt. Im Bereich des wissenschaftlichen Rechnens warten immer noch viele Probleme, wie z.B. komplexe Klimasimulationen oder Strömungsberechnungen. Heute erreichen die schnellsten Vektorrechner und die schnellsten massiv parallelen Rechner eine Höchstleistung von etwa 20-30 GFlops. Das Rennen für den ersten TeraFlop Rechner hat aber schon angefangen [9][10][11]: Für die Simulation der Quantenchromodynamik (Simulation der Strukturen im Inneren eines Nukleons) sind sowohl in den USA (QCD TeraFlop Collaboration & Thinking Machines Corporation) als auch in Europa (European TeraFlop Initiative, Parsytec) neue spezielle Rechner in der Planungsphase. Der Parsytec Rechner soll aus 65'536 T9000 Transputer bestehen und eine theoretische maximale Leistung von 1.6 TFlops[2] erreichen. Die erste TeraFlop Maschine wird für 1993 erwartet.

Die schnellste Hardware lässt sich aber nicht ohne Software betreiben. Die Effizienz der massiv parallelen Rechner ist im wesentlichen ein Software-Problem. Der Erfolg der Vektorrechner lässt sich somit erklären, dass sich die Software Entwicklung für deren Einsatz nicht wesentlich von derjenigen für sequentielle Rechner unterscheidet. Für den erfolgreichen Einsatz der neuen Parallelrechnergeneration muss eine ganze Reihe von Software-Problemen gelöst werden [12]. Die Programmierumgebung der meisten massiv parallelen Rechner ist sehr rudimentär. Der Benutzer kann sich selten auf seine Applikation konzentrieren. Er muss sich mit Architekturdetails beschäftigen und verliert auch viel Zeit für die Datenverteilung, die Lastverteilung der parallelen Prozesse, die Routingstrategie, etc. Es wird die Aufgabe der 90er Jahre sein, Softwarewerkzeuge für einen einfachen und möglichst effizienten Einsatz der massiv parallelen Rechner zu entwickeln.

In diesem Artikel wird anhand des **occam**-Transputer Konzeptes gezeigt, wie man massiv parallele Rechner bauen und betreiben kann. Ferner werden Applikationen vorgestellt und die damit verbundenen Softwareprobleme aufgezeigt.

## 2 Der Transputer

Der Transputer ist ein schneller RISC Prozessor mit einer Leistung von 10 MIPS in der Grundversion (T4xx;T2xx) und zusätzlich 1.5 MFlops[3] in der Floating Point Version (T8xx). Die Tabelle 1 zeigt eine Übersicht der zur Zeit erhältlichen Transputer Typen. Die

[2] TFlops Trillion Floating Point Instructions per second
[3] Bei einer Taktfrequenz von 20 Mhz

Architektur des Transputers kann mit derjenigen eines *Single–Chip–Computers* verglichen werden [13]. Ein entscheidender Unterschied zu herkömmlichen Mikroprozessoren sind die 4 bidirektionalen seriellen *Links*, über welche verschiedene Transputers miteinander verbunden werden können. Dabei realisiert jeder Link zwei gerichtete Kommunikationskanäle für den Datentransfer. Die Links werden über je einen eigenen DMA–Kontroller gesteuert und haben je eine wählbare Leistung von 5, 10 oder 20 Mbit/sec. Sie können auch generell für die Kommunikation mit der Aussenwelt verwendet werden, indem herkömmliche (z.B. parallele oder serielle) Schnittstellen über Linkadapters daran angeschlossen werden. Zusätzlich können über die programmierbare Speicherschnittstelle periphere Komponenten auf herkömmliche Art und Weise angeschlossen werden. Zusätzlich besitzt der Chip einen kleinen internen Speicher. Ein einfacher Interrupteingang, der wie ein Link angesprochen wird, kann im üblichen Sinn verwendet werden. Durch den eingebauten *Scheduler* (Prozessmanager) unterstützt der Transputer direkt die Realisierung von parallelen Prozessen auf einem einzigen Prozessor. Damit können parallele Prozesssysteme auf mehreren oder, ohne den üblicherweise grossen Laufzeitverlust bei konventionellen Prozessoren in Kauf nehmen zu müssen, auf einem einzelnen Prozessor ausgeführt 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.

| Typ | Datenbreite | interner Speicher | FPU | Links |
|---|---|---|---|---|
| T414 | 32 Bit | 2 KBytes | nein | 4 |
| T400 | 32 Bit | 2 KBytes | nein | 2 |
| T425 | 32 Bit | 4 KBytes | nein | 4 |
| T805 | 32 Bit | 4 KBytes | ja | 4 |
| T801 | 32/32 Bit | 4 KBytes | ja | 4 |
| T225 | 16 Bit | 4 KBytes | nein | 4 |
| M212 | 16 Bit | 2+2 KBytes | nein | 2 |
| C004 | | | | 32 |
| C012 | | | | 1 |
| C011 | | | | 1 |

Tabelle 1: Transputertypen (1991). Der T801 hat einen getrennten Daten/Adressbus, die andern arbeiten im Multiplex. Der T212 wird als Diskcontroller eingesetzt und verfügt über zusätzlich 2 KBytes internes ROM. Der C004 ist ein programmierbarer 32 x 32 Crossbarswitch und die C011/12 sind Bausteine für konventionelle Schnittstellen.

## 2.1 T9000 Transputer

Die nächste Generation von Transputern trägt den Namen **T9000** [14]. Das Konzept des Transputers, das Prozessor– und Kommunikationsleistung in einem Baustein vereinigt und das Scheduling von Prozessen in Hardware unterstützt, wird beibehalten und in wesentlichen Punkten noch verbessert. Es wird dabei auf Kompabilität mit der bestehenden Transputerfamilie geachtet. Der T9000 ist bezüglich des Instruktionssatzes und binärem

Code kompatibel mit dem T805. Die Abbildung 1 zeigt die Architektur des T9000. Die Prozessorpipeline mit 5 stages erlaubt mehrere Instruktionen in einem Zyklus zu verarbeiten, wobei die *Decoder/Grouper* Einheit versucht, die anstehenden Instruktionen in optimal ausführbaren Gruppen zu organisieren, was einer Code Optimierung gleichkommt. Die Prozessorleistung soll bei 50Mhz bis zu 150 MIPS (32 bit) und 20 MFlops (64 bit) betragen. Der 16K Byte Instruktions– und Datencache Speicher kann als eigentlicher Cache Speicher oder als schneller on-chip Speicher wie beim T805 verwendet werden. Der Cache Speicher kann bezüglich dieser beiden Funktionen partitioniert werden, und soll eine Bandbreite von 200M Worte/sec. aufweisen. Die programmierbare Speicherschnittstelle soll eine Bandbreite von 50 M Worten/sec leisten.

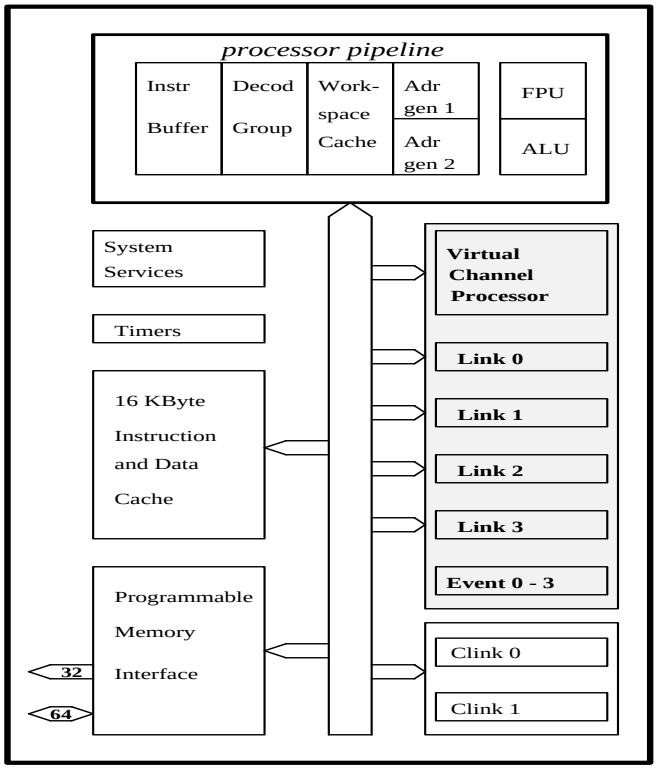


Abbildung 1: Blockdiagramm des T9000 Transputers

Das Scheduling wird wie bei der bisherigen Transputerfamilie realisiert. Jedoch gibt es zusätzlich die Möglichkeit, das Timeslicing und die Unterbrüche explizit zu kontrollieren und zu beeinflussen. Im weiteren werden auch die Prozess– Fehlerbehandlung und das Debugging besser unterstützt als bisher.

Die wichtigste Neuerung neben der neuen Architektur der Prozessor Pipeline und der um Faktoren höheren Leistung betrifft die Kommunikation. Der Konnektivitätsgrad des T9000 für Netzwerke ist wiederum durch die 4 Links begrenzt. Für die Programmierebene gibt es jedoch eine wesentliche Erleichterung. Musste bisher in einem Netzwerk von Transputern aufwendige Software mit Multiplexers/Demultiplexers für die Kommunikation über Links entwickelt werden, so bietet der T9000 derartige Funktionen in Hardware an. Die Kommunikation über Links erfolgt über *virtuelle Kanäle*. Ein spezieller Prozessor für virtuelle Kanäle (*virtual channel processor*) erlaubt es, mehrere virtuelle Kanäle an

denselben physikalischen Link zu binden und damit die Multiplex/Demultiplex Funktionen zu übernehmen. Die Konfiguration von **occam** Programmen wird dadurch wesentlich vereinfacht.

Die Links weisen dieselbe Funktionalität auf wie bisher. Ein neues Protokoll und die Verwendung von 4 anstelle von bisher 2 physikalischen Leitungen erlauben eine höhere Leistung (bis zu 80 MBytes/sec) und ermöglichen die synchrone Kommunikation auf höherer Ebene: bisher war diese nur Byte–weise garantiert, beim T9000 wird dies auf der Ebene von Meldungspaketen (z.B. ein **occam** Kommunikationsprotokoll) möglich sein.

Zwei separate Kontroll–Links (*clinks*) dienen zur Kontrolle von Netzwerken bestehend aus T9000 Transputern und C104 Routing Einheiten. Damit ist es möglich, Fehler unabhängig von den Daten–Kommunikationslinks zu behandeln. Ein Netzwerk wird durch die *clinks* kontrolliert und ist über diese in einer *daisy chain* oder einen *Baum* verbunden.

Vom Standpunkt der parallelen Systeme besteht die wesentliche Neuerung in den *virtuellen Kanälen* und dem dazugehörigen *dynamischen Routing* Baustein C104 und damit zusammenhängend im neuen *Meldungs–orientierten* Kommunikationsprotokoll.

# 3 Parallele Rechner

Die heutigen parallelen Rechner [3] sind meistens Kontrollflussrechner und können grob in zwei Gruppen eingeteilt werden:

**SIMD** (Single Instruction Multiple Data) Zur SIMD Gruppe gehöhren die massiv parallelen Systeme (bezüglich der Anzahl Recheneinheiten). Ein solches System besteht aus tausenden von identischen Recheneinheiten (Arithmetisch-Logische Einheiten), welche von einer einzigen Kontrolleinheit gesteuert werden. Alle Recheneinheiten führen dieselbe Operation gleichzeitig (synchron) auf verschiedenen Daten aus. Der Kontrollfluss selber wird nicht parallelisiert. Man spricht auch von Datenparallelismus.

**MIMD** (Multiple Instruction Multiple Data) Das Hauptgewicht bezüglich der Entwicklung von parallelen Rechnern liegt heute bei MIMD Typen. Die MIMD Systeme werden in einem asynchronen Modus betrieben. Aus diesem Grund ist die Programmierung solcher Maschinen wesentlich komplizierter als im SIMD Fall. Im wesentlichen unterscheidet man zwischen zwei Hauptklassen von MIMD Maschinen:

**Geschaltete Systeme** Die geschalteten (switched) MIMD-Maschinen enthalten gemeinsame Speicher, über die die verschiedenen Prozessoren miteinander kommunizieren und sich gemeinsam synchronisieren. Diese werden als *eng gekoppelte* parallele Systeme bezeichnet. Die Prozessoren eines geschalteten MIMD-Systems können zusätzliche private Speicher besitzen. Zu einer Architektur mit einem gemeinsamen Speicher gehört ein Bussystem. Man stösst wegen des Speicher/Busflaschenhalses schnell an Grenzen. Daraus folgt, dass derartige Systeme immer nur aus einer geringen Anzahl von Prozessoren bestehen. Ausserdem sind diese nur schlecht erweiterbar.

**Netzwerke** Netzwerke basieren auf dem Prinzip des Nachrichtenaustausches (message passing) über ein Netzwerkmedium, das im allgemeinen eine komplexe Topologie (Torus, Hyperwürfel, etc.) aufweist. Sie werden auch als *lose gekoppelte* parallele Systeme bezeichnet. Jeder Prozessor besitzt seinen eigenen privaten Speicher, auf den kein anderer Prozessor Zugriff hat, das heisst es gibt keinen gemeinsamen Arbeitsspeicher. Diese Architekturen sind sehr flexibel und lassen sich bei Bedarf einfach erweitern. Man unterscheidet zwischen Maschinen mit einer statischen und solchen mit einer dynamischen Vernetzungsstruktur. Unter den statischen Maschinen wird noch unterschieden zwischen fest verdrahteten Strukturen und rekonfigurierbaren Strukturen. Fest verdrahtete Systeme werden hauptsächlich für spezifische Applikationen gebaut und verwendet. Transputernetzwerke sind besondere Beispiele dieser zweiten MIMD Klasse, denn der Transputer erlaubt es, leistungsfähige und kostengünstige paralle Systeme zu bauen.

Die folgende Liste zeigt einige Transputer–basierte Systeme, wovon eines genauer beschrieben wird.

- Supercluster; Parsytec (BRD)
- Meiko Computing Surface; Meiko (GB)
- ATW Workstation; Atari (GB)
- Supernode; Telmat (F) und Parsys (GB)
- Cogent XTM Workstation; Cogent Research (USA)

Für Ende 1992 ist eine neue Transputerbasierte Maschine von der Firma Parsytec angekündigt. Sie soll mit bis zu 16'384 Prozessoren vom Typ T9000 ausgestattet sein und eine maximale Leistung von 400 GigaFlops haben.

## 3.1 Die Supernode Architektur

Die Supernode–Maschine [15] ist im Rahmen eines ESPRIT–Projektes entwickelt worden. Der Grundbaustein (im folgenden *Node* genannt) besteht aus 18 T800 Transputer. Davon besitzen 16 je 4 MByte Speicher und bilden die rechnende Komponente der Maschine. Ein weiterer T800 verwaltet einen 16 MB grossen gemeinsamen Speicher[4]. Der letzte Transputer wird zusammen mit einem M212 Disktransputer als Diskverwalter eingesetzt. Alle Transputer Links sind an einen Konfigurationsbaustein angeschlossen. Dieser Baustein funktioniert wie eine Telefonzentrale und kann innerhalb von $2\mu$s zwei beliebige freie Links miteinander verbinden. Er ermöglicht damit eine dynamische Rekonfiguration des Nodes. Zur Steuerung des Konfigurationsbausteins wird ein T212 Transputer (Kontrolltransputer) eingesetzt. Zusätzlich sind alle Transputer an einen sogenannten Kontrollbus angeschlossen. Der Kontrollbus dient

[4]Der gemeinsame Speicher ist mit Linkverbindungen über den Kontrolltransputer ansprechbar

- zur Übertragung von Systemsignalen zwischen den rechnenden Komponenten und dem Kontrolltransputer
- zur Überwachung der einzelnen Linkaktivitäten
- zur dynamischen Rekonfiguration, indem die freien Links dem Kontrolltransputer übermittelt werden.

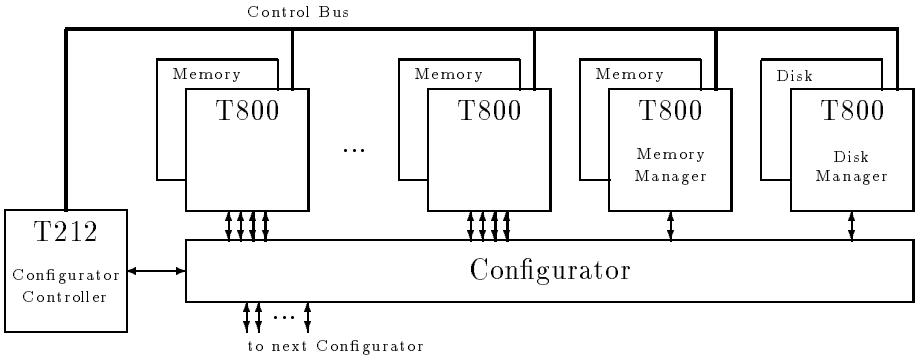


Abbildung 2: Supernode

Die Supernode Architektur ist völlig modular. Verschiedene Nodes können, wie die T800 innerhalb eines Nodes, zusammengesetzt werden, mit der Einschränkung, dass die Anzahl aktiver Links, welche für die interne Kommunikation jedes Nodes benutzt werden, dementsprechend reduziert wird. Der Kontrolltransputer jedes Nodes wird dann zum Sklave des Kontrolltransputers des Supernodes.

In einer solchen Konfiguration könnte ein Node die I/O Funktionen übernehmen. Theoretisch erreicht damit ein Node mit 16 Transputern und 64 Links eine Bandbreite von $\frac{1}{2}$GBit pro Sekunde für die Datenübertragung.

# 4 Occam

## 4.1 Einführung: CSP und occam

Das Transputerkonzept und die Sprache **occam** basieren auf der Theorie der sequentiellen kommunizierenden Prozesse (*CSP : Communicating Sequential Processes*) [16]. CSP bildet ein mathematisch fundiertes Programmiermodell für parallele Systeme. Das Modell definiert Begriffe wie *Prozess*, *Parallelität* oder *Nebenläufigkeit* und *Kommunikation* und veranschaulicht deren Bedeutung für die Spezifikation, den Entwurf, die Implementierung und die Verifikation von parallelen Systemen :

- *Prozesse* beschreiben das Zusammenwirken eines Systems mit seiner Umgebung und können parallel oder zeitlich überlappend ablaufen.

- Die Grundidee von CSP liegt darin, dass Systeme auf natürliche Art und Weise in Subsysteme zerlegt werden können, die parallel arbeiten und untereinander sowie mit der Umgebung in Wechselwirkung stehen.
- Die *Kommunikation* zwischen zwei Prozessen wird als Spezialfall der Prozesswechselwirkung verstanden, in welchem der eine Prozess eine Meldung zur gleichen Zeit abschickt, wie der andere sie empfängt.

Damit kann CSP als Entwurfshilfsmittel für parallele Systeme eingesetzt werden. Es verlangt aber vom Programmierer vertiefte mathematische Kenntnisse. Allerdings ist die Betrachtungsweise von parallelen Systemen in CSP auch für den mathematischen Laien von grossem Wert für die Progammentwicklung[5]. Deshalb finden sich die Ideen von CSP in anderen Entwurfsmethoden wieder. Zu den effizientesten gehören etwa SDL [17] für Telekommunikations–Anwendungen oder Petri Netze [18],[19] für eingebettete Systeme.

Die Sprache **occam** [20] ist speziell geeignet für die Programmierung von Netzwerken von Prozessoren, also für parallele Systeme [21]. Ein **occam** Programm besteht aus einer Menge von parallelen sequentiellen Teilprozessen. 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 das Weiterfahren von einem anderen Prozess abhängig ist. Die Kommunikation ist synchron, ungepuffert und findet immer über Punkt-zu-Punkt Verbindungen (Kanäle) statt. Parallele Prozesse können nur über Kanäle Daten austauschen, 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. Für den parallelen Teil gibt es Sprachelemente für die Kommunikation (Senden und Empfangen), Kanalprotokolle und die Parallelität. Im Gegensatz zu anderen Sprachen, wie ADA oder CHILL ist bei **occam** die Parallelität das grundlegende Konzept und der sequentielle Fall nur ein Spezialfall [22]. Die Sprache ist im **occam 2** Reference Manual [23] definiert. Der Zusammenhang von **occam** und *CSP* wird in [24] und [25] dargelegt.

## 4.2 Occam Sprachaufbau

### 4.2.1 Primitive Prozesse

**occam** Prozesse bestehen aus Kombinationen von fünf primitiven Prozesse. Jeder primitive Prozess muss auf eine eigene Zeile geschrieben werden, d.h. die Sprache ist sensitiv auf das textuelle Einrücken (*indentation sensitive*). Die Tabelle 2 zeigt die primitiven Prozesse und ihrer Bedeutung.

### 4.2.2 Konstruktoren

Konstruktoren erlauben es, neue Prozesse durch wiederholte Zusammensetzung von schon existierenden zu definieren. Die drei fundamentalen Konstrukte sind die *Sequenz*, die

[5] Da **occam** auf CSP basiert, kann es selbst auch als Spezifikationsmittel eingesetzt werden

| Zuweisung | a := 1 | Zuweisung des Wertes eines Ausdruckes an eine Variable. |
|---|---|---|
| Eingabe | c ? x | Der Prozess liest einen Wert über einen Kanal in eine Variable ein (Empfangsprozess). |
| Ausgabe | c ! x | Der Prozess sendet einen Wert über einen Kanal an einen anderen Prozess (Sendeprozess). |
| Skip | SKIP | Der Prozess startet und terminiert sogleich. |
| Stop | STOP | Der Prozess startet, terminiert aber nie. |

Tabelle 2: Primitive Prozesse in **occam**

*Parallelität* und die *Alternative*. Zusätzliche Konstruktoren betreffen die *Auswahl* und die *Wiederholung*. Ausser der letzteren können alle Konstruktoren zu Familien von ähnlichen Prozessen repliziert werden. Die Tabelle 3 zeigt die **occam** Konstruktoren.

| Sequenz | SEQ | Sequentielle Zusammensetzung von Prozessen. |
|---|---|---|
| Parallelität | PAR | Alle Komponenten des PAR Konstruktes werden parallel ausgeführt. Der durch PAR definierte parallele Prozess terminiert, falls alle seine Komponenten terminiert haben. |
| Alternative | ALT | Die Komponenten von ALT sind bewachte Eingabeprozesse (*guarded processes*) gefolgt von je einem Prozess. |
| Auswahl | IF | Die Komponenten eines IF bestehen aus Bool'schen Bedingungen gefolgt von je einem Prozess. |
| Auswahl | CASE | Dies ermöglicht die Auswahl von Prozessen bezüglich bestimmter Werte eines Ausdruckes. |
| Wiederholung | WHILE | Die Komponente dieses Prozesses wird solange wiederholt, als die dazugehörige Bedingung erfüllt ist. |

Tabelle 3: Konstruktoren in **occam**

Eine Replikation erzeugt einen Array von Prozessen gleichen Typs. Die Konstruktoren `SEQ, PAR, ALT, IF` können auf diese Weise vervielfacht werden:

```
REP index = base FOR count
  -- REP steht fuer SEQ, PAR, ALT oder IF
```

Mit dieser Replikation werden `count` ähnliche Prozesse als Array erzeugt. Die Identifikation der einzelnen Komponenten erfolgt über die Variable `index`.

### 4.2.3 Deklarationen und Ausdrücke

Jede Deklaration in **occam** ist lokal zu einem Prozess und wird in der Verschachtelungshierarchie nach innen vererbt. Es können Variablen und Konstanten verschiedener Datentypen vereinbart werden. Kanaldeklarationen definieren die Verbindungen zwischen Prozessen. Die Information, die über einen Kanal transferiert wird, ist durch ein Protokoll definiert. Prozeduren und Funktionen erlauben die Definition von mehrmals aufrufbaren Prozessen.

Alle Variablen und die Kanäle können zu *Arrays* beliebiger Dimension kombiniert werden. Die Komponenten eines Arrays können einzeln angesprochen werden, oder auch als ganze Teilstücke (*slices*).

Ein Ausdruck in **occam** ist entweder ein monadischer Operator und dessen Operand, oder ein dyadischer Operator mit zwei Operanden, oder eine Datentypkonversion, oder ein Operand. Alle Operatoren besitzen die gleiche Priorität. Komplexe Ausdrücke müssen deshalb geklammert werden. **occam** stellt arithmetische Operatoren, Vergleichsoperatoren, Bool'sche Operatoren, logische Operatoren, shift Operatoren und Verzögerungsoperatoren zur Verfügung.

### 4.2.4 Kommunikation

Der primitive Eingabe und der Ausgabe Prozess definieren zusammen eine *synchrone*, *ungepufferte Punkt–zu–Punkt* Kommunikation. Eine Kommunikation kann also nur dann stattfinden, wenn zwei sich entsprechende Prozesse zu einer Interaktion über einen gemeinsamen Kanal bereit sind. Ein solcher Kanal gehört zu genau einem Paar von Ein–und Ausgabe Prozessen und ist gerichtet. Bei jeder Kommunikation wird eine Meldung vom Sender zum Empfänger transferiert. Der Typ der Meldung ist durch das Kanalprotokoll bestimmt (z.B. ein Integer oder eine Folge von beliebigen Datentypen).

Das `ALT` Konstrukt ermöglicht die Auswahl einer Kommunikation aus einer Menge von beliebig vielen möglichen Kommunikationen. Besitzt ein Prozess mehrere Eingabekanäle, dann kann es vorkommen, dass mehrere der dazugehörigen Ausgabeprozesse gleichzeitig zu einer Kommunkation bereit sind. In diesem Fall wird in der Alternative nicht–deterministisch eine Kommunikation ausgewählt. Die Komponenten einer Alternative können zusätzlich durch Bool'sche Ausdrücke bewacht werden. In der Alternative wird dann aus allen möglichen Kommunikationen, d.h. aus denjenigen Komponenten, bei denen der Sendeprozess bereit ist und ein allfälliger Kontrollausdruck erfüllt ist, genau eine nicht–deterministisch ausgewählt. Das Konstrukt wartet bis mindestens eine seiner Komponenten bereit ist, und terminiert nach der ausgewählten Kommunikation und der Ausführung des dazugehörigen Prozesses[6]. Das `ALT` Konstrukt erlaubt also auf der Empfangsseite auf verschiedene Kommunikationen zu warten und dann auszuwählen. Ein analoges Konstrukt auf der Senderseite ist im Gegensatz zu CSP in **occam** nicht vorhanden.

Das Konzept der Kommunikation in **occam** erlaubt es auch mit Prozessen der Umgebung, d.h. externen Prozessen zu kommunizieren. Zusätzlich sind spezielle Prozesse vorgesehen,

[6] In den meisten Implementationen ist die nicht–deterministische Auswahl nicht realsiert, und es wird die textuell erste Komponente gewählt.

welche die Echtzeit repräsentieren. Eine Kommunikation mit diesen Prozessen erfolgt über spezielle Kanäle, die `TIMER`. Damit lassen sich in Kombination mit `ALT` *timeouts* auf der Empfangsseite, sowie Echtzeitverzögerungen realisieren.

Grundsätzlich erlauben die **occam** Sprachkonstrukte nur die *einfache Kommunikation*. Alle anderen Kommunikationsarten lassen sich aber durch Komposition verschiedener Konstrukte realisieren.

#### 4.2.5 Parallelität

Die Parallelität in **occam** ist grundsätzlich *statisch*. Dies bedeutet, dass Prozesse in einem **occam** Programm nicht dynamisch kreiiert werden können. Vor Ablauf eines Programms ist also der maximale Grad der Parallelität bestimmt. **occam** weist im Gegensatz zu CSP ebenfalls keine Rekursion auf. Diese kann zwar mit replizierten parallelen Prozessen simuliert werden, wobei aber die maximale Rekursionstiefe (Anzahl Komponenten des replizierten `PAR`) bekannt sein muss. Das Fehlen dieser dynamischer Elemente hat einen Grund in der Architektur des Transputers. Die Sprache **occam** und der Transputer wurden nämlich gemeinsam entwickelt. Die Philosophie von **occam** geht auch davon aus, dass es keine Rolle spielt, ob ein paralleles Programm auf einem oder mehreren Prozessoren verteilt abläuft. Darin liegt nun ein weiterer Grund für die statische Parallelität, denn es wäre schwer vorstellbar, dass bei der dynamischen Kreierung von Prozessen dann auch dynamisch Prozessoren generiert werden könnten.

### 4.3 Occam auf Transputernetzwerken

Die Architektur und der Instruktionsatz des Transputers unterstützen die Sprache **occam** optimal. Für die Sprachkonstrukte, die Kommunikationen und die Parallelität betreffen, gibt es spezielle Instruktionen, die bei gewöhnlichen Mikroprozessoren nicht anzutreffen sind. Der mikrocodierte Scheduler unterstützt mit seinem schnellen Context–switch die simulierte Ausführung von parallelen Prozessen bestens. Für deren Ausführung stellt der Transputer zwei Prioritäten zur Verfügung. Die tiefe Priorität (Standard) benutzt ein *time slicing* Verfahren für das Scheduling, während bei hoher Priorität der Prozessor einem Prozess, der ausführungsbereit wird, *sofort* und so lange wie nötig zugeteilt wird.

#### 4.3.1 Kommunikation und Parallelität

Die Links des Transputers realisieren die **occam** Kommunikation, wenn mehrere Prozessoren miteinander verbunden sind. Jeder Link implementiert zwei **occam** Kanäle, in jede Richtung einen. Die Parallelität wird durch die Vernetzbarkeit über die Links gewährleistet. Dabei ist zu beachten, dass sechs prinzipielle funktionale Einheiten des Transputers (CPU, FPU, 4 Links) untereinander parallel arbeiten können. Dies bedeutet insbesondere, dass Berechnungen und Kommunikationen auf jedem Transputer selbst parallel ausgeführt werden können. Um diese interne Priorität optimal ausnutzen zu können, werden Kommunikationsprozesse, die die Links ansprechen als hoch priorisierte Prozesse definiert, und alle anderen Prozesse mit der Standard Priorität. Eine falsche Priorisierung, also wenn

Berechnungen gegenüber Link–Kommunikationen höher priorisiert sind, kann zu beträchtlichen Leistungseinbussen führen. Messungen haben eine Verlangsamung von bis zu einem Faktor 6 ergeben.

### 4.3.2 Konfigurationssprache

Die Implementierung von **occam** auf dem Transputer stellt eine Konfigurationssprache zur Verfügung [23]. Die wichtigsten Konstrukte sind in Tabelle 4 zusammengestellt.

| | | |
|---|---|---|
| Par Priorisierung | `PRI PAR` | Die erste Komponente des Konstrukts ist gegenüber den anderen priorisiert. |
| Alt Priorisierung | `PRI ALT` | Die erste Komponente ist bei der Auswahl immer priorisiert. |
| Allokation | `PLACE x AT a` | Die Variable, der Kanal oder Timer `x` wird zu der Adresse `a` alloziert. Jeder Link hat eine vordefinierte Adresse. |
| Konfiguration | `PLACED PAR` | In der Konfiguration werden den Transputern **occam** Prozesse zugewiesen. |

Tabelle 4: Konfiguration von **occam** Programmen für Transputernetzwerke

Jedem Transputer wird genau ein **occam** Prozess zugewiesen, wobei es sich auch um verschiedene Inkarnationen derselben Prozessdefinition handeln kann. Ein solcher Prozess hat höchstens vier Eingabekanäle und vier Ausgabekanäle, entsprechend den vier bidirektionalen Links des Transputers. Die Allozierung der Kanäle ist *statisch*. Die Nummerierung der Transputer ist völlig frei und ohne Bedeutung. Die Topologie des Programms ist durch die Kanäle eindeutig bestimmt und muss mit derjenigen des Zielsystems übereinstimmen.

# 5 Parallelisierung in occam

Im folgenden wird anhand eines einfachen Beispiels gezeigt, wie sequentielle Programme parallelisiert werden können.

Die Fakultät, als äusserst einfache Berechnung, lässt sich leicht als iteratives Programm formulieren. Da es zwischen den einzelnen Berechnungsschritten auch Unabhängigkeiten bezüglich der Daten gibt, ist es auch möglich die Berechnung der Fakultät zu parallelisieren und damit also ein paralleles Programm zu schreiben. Dieses Beispiel demonstriert wie ein komplettes **occam** Programm aussieht.

Die Fakultät ist wie folgt definiert:

$$\begin{aligned} 0! &= 1 \\ n! &= n \times (n-1)! \end{aligned}$$

Die gewöhnliche iterative und sequentielle Berechnung der Fakultät wird als **occam** Programm wie folgt formuliert

```
INT x, z:
SEQ
  z := 1
  WHILE x > 0
    SEQ
      z := z * x
      x := x - 1
```

Die beiden Berechnungen innerhalb des Programms, die Multiplikation `z := z * x` und die Substraktion `x := x - 1` können bezüglich der Daten kausal unabhängig gemacht werden. Dies geschieht mit der Einführung einer Hilfsvariable `y`. Damit werden die Berechnungen erweitert zu

```
SEQ
  y := x
  z := z * y
  x := x - 1
```

Damit lässt sich nun die Fakultät als paralleles **occam** Programm formulieren

```
INT x, y, z:
SEQ
  z := 1
  WHILE x > 0
    SEQ
      x := y
      PAR
        z := z * y
        x := x - 1
```

Die beiden relevanten Berechnungen `z := z * y` und `x := x - 1` erfolgen jetzt parallel.

Im obigen Programm sind einzelne Aktionen als parallel auszuführend spezifiziert. Das Beispiel der Fakultät kann aber auch als System kooperierender Teilprozesse formuliert werden. Dabei wird hier auch der Ein- und Ausgabeprozess einbezogen, womit ein vollständiges lauffähiges paralleles **occam** Programm ergibt.

```
CHAN OF INT input, result, factor :

PROC input.output(CHAN OF INT in, out)
  INT n, fac :
  SEQ
    out ! n
    in ? result
:
```

```
PROC subst(CHAN OF INT in, out)
  INT x, y :
  SEQ
    in ? x
    WHILE x > 0
      SEQ
        y := y
        PAR
          out ! y
          x := x - 1
:

PROC mult(CHAN OF INT in, out)
  INT y, z :
  SEQ
    y := 2
    z := 1
    WHILE y > 1
      SEQ
        in ? y
        z := z * y
    out ! z
:

PAR
  input.output(result,input)
  subst(input,comm)
  mult(comm,result)
```

Die Fakultätsberechnung ist sequentiell ebenso leicht wie iterativ auch rekursiv formulierbar. In **occam** ist die Rekursion nicht vorgesehen im üblichen Sinne. Mit dem Konzept der parallelen Prozesse kann jedoch auch in **occam** Rekursion simuliert werden.

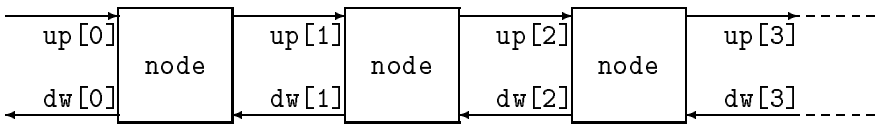


Abbildung 3: Simulation der Rekursion in **occam**

Eine lineare Rekursion lässt sich als eine Pipeline von Parallelen Prozessen formulieren. Jeder Prozess stellt dabei eine Rekursionsstufe dar

```
VAL max.lenght IS 20 :
[max.lenght+1]CHAN OF INT up, down :

PROC node(CHAN OF INT in.left, out.left, in.right, out.right)
  INT x, y :
  SEQ
    in.left ? x
    IF
      x > 0
        SEQ
          out.right ! x - 1
          in.right ? y
          out.left ! x * y
      x = 0
        out.left ! 1
      TRUE
        SKIP
:

PROC input.output(CHAN OF INT in, out)
  INT n, fac :
  SEQ
    out ! n
    in ? result
:

PAR
  input.output(down[0],up[0])
  PAR i = 1 FOR max.length
    node(up[i-1],down[i-1],down[i],up[i])
```

# 6 Anwendungsspektrum

Der Transputer kann überall eingesetzt werden, wo gewöhnliche Mikroprozessoren zum Zug kommen. Seine überragende Stärke liegt in seinen Vernetzungs- und Kommunikationsfähigkeiten. Dies ermöglicht es, leistungsfähige und sichere Systeme zu einem sehr günstigen Preis aufzubauen. Die folgende Auflistung zeigt einige typische Anwendungsbereiche für parallele Transputersysteme.

**Medizin:** Der Transputer ist besonders geeignet für die Verarbeitung von Computertomographie–Daten. Dreidimensionale Bilder können dank dem Einsatz von Transputernetzwerken in Echtzeit manipuliert (Ray Tracing) und graphisch dargestellt werden.

**Aeronautik:** Das Flugverhalten von neuen Flugzeugen kann dank komplexen Simulationsprogrammen (Fluid Dynamics) schon im Entwurfsstadium untersucht und ver-

bessert werden.

**Chemie:** Transputer werden hier eingesetzt für die Modellierung und Simulation von Molekeln. Insbesondere wird die Synthese von neuen Medikamenten wesentlich vereinfacht.

**Kryptologie, Signalverarbeitung:** Dank dem Transputer ist der Entwurf von komplexen und sicheren Kryptologiesystemen wesentlich vereinfacht worden. Der Einsatz von Transputern ist in diesem Gebiet viel einfacher und flexibler als entsprechende Signalprozessor-basierte Systeme.

**Telekommunikation:** In neuen Telekommunikationssystemen (Bildtelephon, etc) müssen immer mehr Daten in Echtzeit verarbeitet werden. Der Transputer ist besonders geeignet für den Aufbau von komplexen Kommunikationsschnittstellen (Breitbandschnittstellen, ISDN etc).

**NC Steuerung:** Der Transputer wird immer mehr für die NC Steuerung von Werkzeugmaschinen eingesetzt (Schnelle Spline Interpolation, parallele Steuerung).

**Bildverarbeitung:** Nicht nur im medizinischen Bereich, sondern überall, wo Bilder verarbeitet werden, kann der Transputer leistungssteigernd eingesetzt werden: Bilderkennung, zum Beispiel das Fingerabdruck–Erkennungssystem von Scotland Yard, Echtzeit Animation, Ray Tracing, Video, usw.

**Wirtschaft:** Simulationen von betriebswirtschaftlichen Systemen ermöglichen es, genauere Prognosen herzustellen.

Zur Illustration folgt nun ein Beispiel einer parallelen Lösung einer Differentialgleichung.

**Zweidimensionale Poisson Gleichung:**

Gesucht ist ein paralleler Algorithmus zur numerischen Lösung der zweidimensionalen Poisson Gleichung

$$\Delta u(x,y) = f(x,y)$$

im rechteckigen Gebiet $G = [0,1] \times [0,1]$ mit Randbedingung

$$u(x,y) = f(x,y) \text{ auf } \partial G$$

Die Methode der finiten Differenzen führt zur folgenden Diskretisierung des Laplace Operators

$$\Delta u_{i,j} = u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}$$

und zu folgendem iterativen Relaxationsalgorithmus

$$u_{i,j}^{k+1} = \frac{u_{i+1,j}^{k} + u_{i-1,j}^{k} + u_{i,j+1}^{k} + u_{i,j-1}^{k} - f_{i,j}}{4}$$

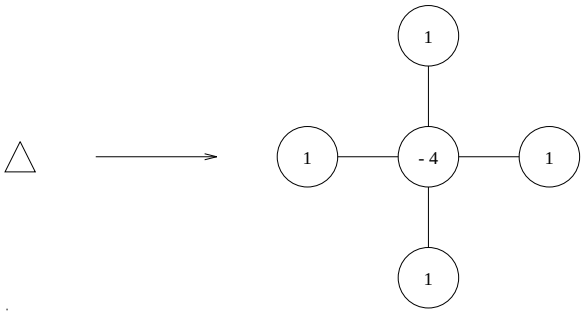


Abbildung 4: Laplace Operator

In jedem Iterationsschritt müssen die Werte von jedem Gitterpunkt $u_{i,j}$ geändert werden. Im diesem Falle ist die geometrische Parallelisierung die am besten geeignete Methode. Prozessoren werden gitterartig angeordnet. Kann jedem Gitterpunkt ein Prozessor zugeordnet werden, so sieht jeder Relaxationsschritt lokal wie folgt aus:

```
REAL32 site,north,south,east,west :
SEQ
  PAR
    to.north ! site
    to.south ! site
    to.east ! site
    to.west ! site
    from.north ? north
    from.south ? south
    from.east ? east
    from.west ? west
  site := (((north + south) + (east + west)) - f) / 4.0
```

In der Praxis gibt es weniger Prozessoren als Gitterpunkte. Aus diesem Grund werden jedem Prozessor rechteckige Teile des Integrationsgebiets zugeordnet:

Folgender paralleler Algorithmus ist dem Transputer speziell zugeschnitten. Er verwendet die Eigenschaft, dass Datentransfer über Transputer Links parallel zur Berechnung erfolgen kann:

```
SEQ
  PAR
    ... update interior sites
    ... transfer boundary data
  ... update boundary sites
```

Es muss dabei beachtet werden, dass das Verhältniss

$$\frac{\text{Kommunikationskosten}}{\text{Berechnungskosten}}$$

möglichst günstig ausfällt, das heisst möglichst klein ist.

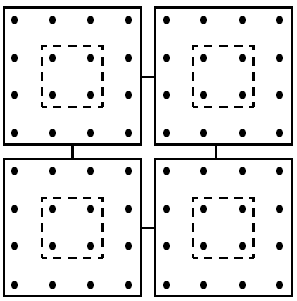

Abbildung 5: Geometrischer Parallelismus: Die Poisson Gleichung

# 7 1 Prozessor 10 MIPS — 100 Prozessoren 1000 MIPS ?

Der für sich allein schon leistungsfähige Transputer lässt sich aufgrund seiner Kommunikationsfähigkeiten sehr einfach als modularer Baustein für Mehrprozessorsysteme einsetzen. Damit hat man also die Möglichkeit beim Erreichen der Leistungsgrenze eines Systems einfach weitere Prozessoren hinzuzufügen. Dabei stellt sich das Problem wie die Gesamtleistung eines solchen Systems auch wirklich ausgenutzt werden kann [26]. Beim Bau eines Hauses genügt es beispielsweise nicht, einfach mehr Arbeitskapazität zur Verfügung zu stellen. Um diese auch effizient einsetzen zu können, ist eine sorgfältige Planung und Koordination erforderlich. Andernfalls läuft man sogar Gefahr durch gegenseitige Störungen die Arbeiten zu verzögern. Gewisse Arbeiten können parallel ausgeführt werden: z.B. der gleichzeitige Aufbau verschiedener Mauern eines Hauses. Das Dach jedoch kann erst dann in Angriff genommen werden, wenn alle Mauern hochgezogen sind. Am Dach selbst können wieder mehrere Arbeiter gleichzeitig eingesetzt werden.

Auf ein Mehrprozessorsystem übertragen, zeigen diese Feststellungen, dass für eine optimale Leistung die Arbeit von allen vorhandenen Prozesoren zu möglichst gleichen Teilen erbracht werden soll, und dass die Prozessoren untereinander koordiniert arbeiten. Für die Programmierung eines Mehrprozessorsystems werden die verschiedenen Arbeiten, die parallel ausgeführt werden können, zu Prozessen zusammengefasst. Um die Koordination zu gewährleisten, kommunizieren die Prozesse miteinander und synchronisieren sich gegenseitig. Damit eine Aufgabe nun in der für ein gegebenes Mehrprozessorsystem kürzesten Zeit gelöst werden kann, muss der Programmierer die Aufgabe zuerst bestmöglichst in verschiedene parallele Prozesse unterteilen. Es stellt sich dann das Problem, die Prozesse *optimal* auf das Mehrprozessorsystem abzubilden. Ein Werkzeug oder eine Systemsoftware, die für diese Verteilung eingesetzt werden können, müssen den folgenden Bedingungen genügen:

**Lastoptimierung (*load balancing*):** Die Prozesse und die Arbeit, die die Prozesse zu erledigen haben, sollen möglichst gleichmässig auf die vorhandenen Prozessoren verteilt werden.

**Kommunikationsoptimierung :** Zwischen je zwei Prozessoren gibt es immer eine obere physikalische Grenze für die Kapazität der Datenübertragung (Kommunikation). Bei

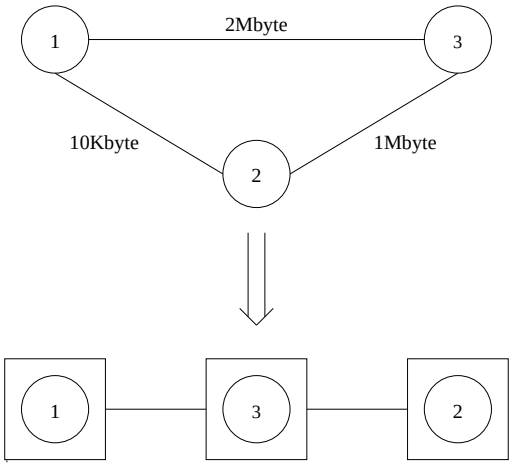


Abbildung 6: Kommunikationsoptimierung: Wegen der linearen Struktur des Prozessornetzwerkes, muss der Datentransfer zwischen den am Rand plazierten Prozessen via den mittleren Prozessor stattfinden. Aus diesem Grund sollten die Prozesse 1 und 2 am Rand plaziert werden, um die Auslastung der beiden Kommunikationskanäle optimal auszugleichen (2,01MByte und 1,01Mbyte).

> der Verteilung der Prozesse auf die Prozessoren müssen daher die Kapazitäten aller vorhandenen Kommunikationskanäle optimal ausgenützt bzw. ausgelastet werden.

Die last– und kommunikationsoptimierte Verteilung von Prozessen auf Prozessoren ist für jedes Mehrprozessorsystem die entscheidende Grundlage für einen effizienten und sinnvollen Einsatz. Nur so kann das hochgesteckte Ziel eines möglichst proportionalen Verhältnisses zwischen der Anzahl Prozessoren und der Leistung erreicht werden. Wie die meisten Optimierungsprobleme, ist auch das Problem der Prozessverteilung (*Mapping Problem*) nur mit hohem Aufwand zu lösen. Es ist deshalb besonders wichtig, Werkzeuge zur Verfügung zu haben, die das Mapping Problem nicht nur optimal, sondern auch effizient bearbeiten können.

Es ist naheliegend, dass das Mapping Problem selbst unter Ausnutzung aller vorhandenen Prozessoren eines Systems bearbeitet wird. Dabei bringt ein verteilter, die Struktur des Mehrprozessorsystems ausnutzender Algorithmus sehr gute Resultate [27],[28],[29]. Er basiert darauf, dass ein globales Optimum bezüglich Prozessorbelastung und Kommunikationslast durch ständigen Austausch von Information zwischen unmittelbar benachbarten Prozessoren stattfindet. Verschiedene Untersuchungen haben gezeigt, dass dieses lokale, verteilte Vorgehen den klassischen Optimierungsstrategien (wie Simulated Annealing) überlegen ist. Dazu ist allerdings eine genaue Kenntnis der durch die verschiedenen parallelen Prozesse verursachten Rechenlast und Kommunikationsbelastung notwendig.

Diese Anforderungen erfüllt ein neues Werkzeug: der Monitor TNT–PFY, für die Analyse der Belastung eines Prozessors durch verschiedene Prozesse [30], kann durch die Erfassung jedes Context–Switches die genaue Zeit, die jeder einzelne Prozess benötigt, bestimmen. Zudem kann das Werkzeug jeden einzelnen Kommunikationskanal überwachen und bestimmen wie gross der Datentransfer zwischen zwei Prozessen ist. Damit kann also die für eine optimale Verteilung nötige Information (Rechenlast und Kommunikation) gelie-

fert werden. Diese genauen von TNT–PFY berechneten Informationen können zudem für den Test von zeitkritischen Programmteilen und für die Aufdeckung der bei parallelen Programmen oft auftretenden Verklemmungen (*deadlocks*) verwendet werden.

## Literaturverzeichnis


[1] J.E. Boillat. Paralleles Rechnen. Vorlesungsskript, Universität Bern, 1989.

[2] P.G. Kropf. *Parallele Algorithmen für Mehrprozessorsysteme mit verteiltem Speicher.* PhD thesis, Universität Bern, Institut für Informatik und angewandte Mathematik, Länggassstrasse 51, CH-3012 Bern, December 1991.

[3] R.W. Hockney and C.R. Jesshope. *Parallel Computers 2.* Adam Hilger, Bristol, second edition, 1988.

[4] W. Myers. Massively parallel systems break through at supercomputing 90. *IEEE Computer*, pages 121–126, January 1991.

[5] Gordon Bell. The future of high performance computers in science and engineering. *Comm. ACM*, 32(9):1091–1101, 1989.

[6] J. Dongarra, A.H. Karp, K. Kennedy, and D. Kuck. 1989 Gordon Bell Prize. *IEEE Software*, pages 100–110, May 1990.

[7] W. M. Bulkeley. Parallel supercomputers are catching on rapidly. *Wall Street Journal*, January 1991.

[8] V.P. Bathkar. Parallel computing: An indian perspective. In H. Burkhart, editor, *Proceedings CONPAR 90 - VAPP IV*, volume 457 of *Lecture Notes in Computer Science*, pages 10–25. Springer, 1990.

[9] The QCD Teraflop Project. A Proposal by the QCD Teraflop Collaboration, October 1990. Submitted to the U.S. Department of Energy.

[10] H. Satz. Teraflops for Europe. *Europhys. News*, 22:64, 1991.

[11] Parsytec kündigt europäischen TeraFlop-Rechner an. Parallel Link, 1991.

[12] J.E. Boillat, H. Burkhart, K.M. Decker, and P.G. Kropf. Parallel Computing in the 1990's: Attacking the Software Problem. In K.M. Decker, editor, *Parallel Architectures and Applications, Proceedings of the 3rd Graduate Summer Course on Computational Physics*, September 1991. Physics Reports, 207(3-5).

[13] INMOS. *Transputer Reference Manual.* Prentice-Hall, Englewood Cliffs, 1988.

[14] INMOS, Bristol. *The T9000 Transputer Products Overview Manual*, 1991.

[15] J.G. Harp, C.R. Jesshope, T. Muntean, and C. Whithby-Strevens. Phase 1 of the development and application of a low cost high performance multiprocessor machine. In Directorate General XIII, editor, *ESPRIT 86: Results and Achievments*, Amsterdam, 1987. Elsevier.

[16] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Englewood Cliffs, 1985.

[17] CCITT, Geneva. *CCITT Z.100 - Z.200 Recommendation*, 1984.

[18] W. Reisig. *System Entwurf mit Netzen*. Springer, Berlin, 1985.

[19] H. Genrich. Predicate/transition nets. In W. Brauer, Reisig W., and R. Rozenberg, editors, *LNCS 254, Petri Nets : Central Models and Their Properites*, pages 207–247, Berlin, 1987. Springer.

[20] D. May. Occam. *ACM SIGPLAN Notices*, 18(4), 1983.

[21] P.G. Kropf. Experiences in Transputing – Applications in Parallel Processing Using OCCAM. *IEEE COMPEURO 87 Conference Proceedings*, page 978, 1987.

[22] P.G. Kropf. A comparison between the languages CHILL and OCCAM. In *Proceedings of the 4th Chill Conference*, Munich, 1986.

[23] INMOS. *OCCAM 2 Reference Manual*. Prentice-Hall, Englewood Cliffs, 1988.

[24] A.W. Roscoe. Denotational semantic for occam. In *Proceedings of the July 1984 Seminar on Concurrency*, pages 306–329. Springer LNCS 197, 1985.

[25] A.W. Roscoe and C.A.R. Hoare. *The Laws of OCCAM Programming*. Oxford University Computing Laboratory, Oxford, 1986.

[26] G.C. Fox and P.C. Messina. Advanced computer architectures. *Scientific American*, 257(4), October 1987.

[27] J.E. Boillat, P.G. Kropf, D.Chr. Meier, and A. Wespi. An analysis and reconfiguration tool for mapping parallel programs onto transputer networks. In T. Muntean, editor, *OUG-7: Parallel Programming of Transputer based Machines*, Amsterdam, 1988. I.O.S.

[28] J.E. Boillat. Load Balancing and Poisson Equation in a Graph. *Concurrency: Practice and Experience*, 2(4), 1990.

[29] J.E. Boillat, N. Iselin, and P.G. Kropf. MARC: A Tool for Automatic Configuration of Parallel Programs. In P. Welch, D. Stiles, T.L. Kunii, and A. Bakkers, editors, *TRANSPUTING 91*, pages 311–329, Amsterdam, 1991. IOS.

[30] TNT - Parallel Computing Support, Bern. *TNT - PFY Reference Manual*, 1990.