Eine CRAY für 100.000 DM

Helmut Grubmüller Helmut Heller Klaus Schulten

30. September 1988

1 Revolution der billigen Parallelrechner

Die Entwicklung und der Einsatz des Computers werden in Vehemenz und Breitenwirkung wohl nur von wenigen Erfindungen des Menschen übertroffen. Die Geschwindigkeit der Erneuerung der Rechnertechnologie überrascht immer wieder, eine Computerrevolution scheint die andere zu jagen. So drückt diese Autoren auch ein gewisses Unbehagen, wenn sie in diesem Artikel eine neue Revolution des Computers ankündigen, aber genau das ist die Botschaft: der von Neumann-Computer wird auf breiter Basis durch den Parallelrechner abgelöst.

Sofort wird jeder Experte einwerfen, daß hier ein ‚alter Hut' angekündigt wird. Das Ende des von Neumann-Computers, d.h. des sequentiell arbeitenden Rechners mit einem oder wenigen Rechenwerken, ist allzuoft gesungen worden. Auch gibt es längst viele Beispiele von Parallelrechnern, nur merkt man davon im Rechneralltag in Büro oder Labor nicht viel. Was wir bisher mit Parallelrechnern erlebt haben, so meinen aber die Autoren, war nur das Rumoren vor dem großen Beben. Was zum jetzigen Zeitpunkt den revolutionären Umschlag bewirkt, ist der ökonomische Faktor: Parallelrechner werden endlich billig, unser hier vorgestellter Eigenbau-Rechner ist ein gutes Beispiel.

Wir wissen seit Marx von der Bedeutung ökonomischer Faktoren für Revolutionen. Wie das Marx'sche Manifest kommt die Botschaft der angekündigten Computerrevolution ebenfalls aus Großbritannien, wird verkündet bzw. verkauft von der Firma INMOS und heißt Transputer: leicht wie Lego-Steine soll man diese Rechnerelemente zusammensetzen können, und kinderleicht soll auch die Programmierung der entstehenden Parallelrechner sein.

Die Botschaft von INMOS ist schon einige Jahre alt, die verbreitete Anfangseuphorie hatte aber zunächst das Gegenteil einer Revolution bewirkt. Die Programmiersprache (occam), bzw. der gelieferte Compiler, hatte viele Tücken, die Transputer-Rechenelemente hatten Defizite, z.B. konnten sie keine Floating Point-Arithmetik. Viele haben sich deshalb früh abgewandt, einige haben den Transputer totgesagt. Aber das europäische Kind lebt und hat sich trefflich entwickelt: der jüngste Transputer ist ein Schmuckstück von VLSI-Design, vereinigt er auf einem Chip doch neben einem 32 bit RISC Computer eine 64 bit Floating Point-Einheit, aus denen eine Dauerleistung von 1.5 Mflops resultiert, 4 Kbytes on chip RAM (50 ns) sowie vier sog. Links, über die der Chip mit anderen Transputern mit einer Bandbreite von etwa 2 MB/sec (full duplex) kommunizieren kann. Auch der neue (occam II) Compiler kombiniert mit einem Folding-Editor (s.u.) ist zufriedenstellend und lebt in idealer Ehe mit der Hardware.

Die Programmiersprache occam beruht auf einem von C.A.R. Hoare vorgeschlagenen Modell der Parallelverarbeitung, den sog. kommunizierenden sequentiellen Prozessen, und gehorcht dabei den folgenden Axiomen:

(i) Kein Prozeß kann Daten mit einem parallelen Prozeß teilen.

(ii) Alle Daten werden zwischen Prozessen über Kommunikationsprimitive ausgetauscht.

(iii) Die Kommunikationsprimitive arbeiten synchron, d.h. solange, bis sowohl der sendende als auch der empfangende Prozeß die Kommunikation für beendet hält.

Die Grundelemente von occam sind dabei Prozesse und Kanäle, über letztere kommunizieren die Prozesse. Ideal ist, daß Prozesse und Kanäle auf einem einzigen Transputer virtuell parallel arbeiten können, d.h. sie werden vom Benutzer parallel programmiert, vom Transputer aber sequentiell abgearbeitet, daß aber

andererseits auch eine Abbildung von Prozessen auf mehrere Transputer und von Kanälen auf deren Links möglich ist und dabei echte Parallelität erreicht wird. Je nach Geldbeutel und Problemstellung kann ein unterschiedlicher Grad von echter Parallelität ohne wesentliche Neuprogrammierung gewählt werden: der Student mit einem einzigen Transputer in seinem Atari rechnet formal genauso wie der Professor auf seiner Transputerfarm.

Doch auch die Revolution der billigen Parallelrechner hat ihre Kosten, sogar enorme, die sich aber auf der Softwareseite niederschlagen. Der Gewinn an Rechenleistung durch Parallelität kann nämlich nur eingestrichen werden, wenn der Programmierer eine Strategie findet, die Leistung eines Programmes auf viele gleichzeitig ablaufende Prozesse ohne große Kommunikationskosten zu verteilen. Sein Ziel sollte dabei sein, die Geschwindigkeit seines Programmes im selben Maße wie die Zahl der ihm zur Verfügung stehenden Transputer ansteigen zu lassen. Dieses Ziel ist nicht leicht zu erreichen. Von den Problemen, die in dem Ausspruch eines der Autoren bei der Entwicklung unseres Programmes „Das Programm rechnet zwar falsch aber parallel!" deutlich werden, einmal abgesehen, ist es gar nicht klar, daß Verdopplung der Zahl der Prozessoren auch eine Verdopplung der Rechengeschwindigkeit bringt; es halten sich im Gegenteil Gerüchte, daß bei manchen Programmierern zusätzliche Prozessoren wegen des Koordinierungsaufwandes (Wer hat nicht schon einmal ausgerufen: „Das mache ich lieber alleine, dann geht es schneller!") die Rechengeschwindigkeit verlangsamen. Wir sind auf jeden Fall sehr stolz, daß bei unserer Programmstrategie 3, 4, 5, ... Prozessoren etwa 2, 3, 4, ... mal so schnell rechnen wie ein einzelner Transputer. Auf jeden Fall bietet die Revolution der billigen Parallelrechner enorme Chancen für intelligente Programmierer. Man stelle sich vor: Kluge Köpfe können Firmen die Rechenleistung, die heute nur auf teuren Supercomputern erzielt werden kann, zu einem hundertmal billigeren Preis verkaufen; die Software-Branche, kaum aus den Kinderschuhen, strebt ihren zweiten Frühling zu, indem sie alles, was bisher produziert wurde, parallelisiert. Daß es dabei Chancen aber auch große Hindernisse gibt, hat die Informatikforschung der letzten Jahre gezeigt.

2 Parallelisierung - ein Wunschtraum

Welche Problemstellungen führen zum Großverbrauch an Supercomputerzeit und sind deshalb ein Target für die Programmierung billiger Parallelrechner? Aus ihrem eigenen Erfahrungsbereich führen die Autoren hier eine Reihe von Antworten an, ohne aber, das sei betont, auch bereits gute Strategien zur Parallelisierung zu kennen.

In der Welt, in der wir leben, der physikalischen, biologischen bzw. sozialen Umwelt, laufen Prozesse parallel in Raum und Zeit ab. Das ist so trivial, daß man selten ein Wort darüber verliert, legt aber nahe, daß Computersimulationen dieser Welt, etwa Simulationen ökonomischer Szenarien oder im Flugsimulator auch parallel gerechnet werden können. Für Simulationsrechnungen werden große Summen ausgegeben. Die jüngste Diskussion um die Ersetzung von Tiefflügen der Bundeswehr durch Übungen am Flugsimulator veranschaulicht den Wert von aufwendigen Computersimulationen. Unseres Erachtens liegt hier eine große Chance für Einsparungen durch Parallelisierung, es läßt sich aber auch die Schwierigkeit veranschaulichen. Grund für das Studium der erwähnten Systeme ist die vielfältige, nicht leicht überschaubare Abhängigkeit der beteiligten Prozesse. Für den Parallelprogrammierer bedeutet nun aber diese Abhängigkeit von Prozessen gerade Kommunikationsaufwand, der ungünstig zu Buche schlagen kann. Eine gute Lösung wird der Programmierer nur erreichen, wenn er die logische Anordnung der Kommunikationskanäle zwischen den simulierten Prozessen durchschaut und seine Transputer entsprechend verschalten kann. Dabei hat er die Schwierigkeit zu meistern, daß der Transputer nur wenige (zur Zeit vier) permanente Verbindungen (Links) zur Verfügung hat, eine Zahl, die oft die simulierte Wirklichkeit nur schlecht widerspiegelt. Der Programmierer muß versuchen, durch intelligente indirekte Verbindungen dieses Defizit zu umgehen. Hier liegt meist seine eigentliche Denkaufgabe. In dieser Hinsicht ist es erfreulich, daß der Hersteller der Transputer einen Chip auf den Markt gebracht hat, der es gestattet, software-mäßig während des Programmablaufes die Verknüpfungen zwischen Transputern zu ändern.

Bei einer technisch bedeutende Klasse von Algorithmen, den Rechnungen mit sog. finiten Elementen, liegt die kommunikative Kopplung durch die zwei-dimensionale oder drei-dimensionale Ausdehnung der dabei beschriebenen mechanischen Werkstücke von vornherein fest. Die genannten Rechnungen beschreiben zum Beispiel Festigkeitseigenschaften von flachen Werkstücken und teilen das Werkstück dazu in Zellen ein. Sind die Zellen viereckig, so besitzt jede Zelle vier Nachbarn, mit denen Information, die lokalen mechanischen

Eigenschaften betreffend, ausgetauscht werden. Man könnte also einen Transputer für jede Zelle einsetzen und entsprechend den Nachbarschaftsverhältnissen durch Links verknüpfen. Da technische Simulationen aber zigtausende von Zellen erfordern, wäre dies keine wirtschaftliche Lösung und der Programmierer wird eine gröbere Einteilung verwenden.

Verwandte numerische Problemstellungen ergeben sich bei hydrodynamischen Berechnungen, etwa im Zusammenhang mit der Entwicklung moderner Flugzeugtypen. So wurde z.B. der Airbus 320 weitgehend am Reißbrett mit Hilfe des Computers und nicht nur mit Hilfe des Windkanals optimiert. Hydrodynamische Rechenprogramme bauen auch auf einer räumlichen Zellenstruktur auf, müssen aber komplexere physikalische Information als herkömmliche mechanische Rechnungen mit finiten Elementen verwalten. Auch hier liegt eine Parallelisierung nahe und in der Tat werden entsprechende Anstrengungen von vielen Arbeitsgruppen unternommen, unter anderem von der Gesellschaft für Mathematik und Datenverarbeitung in Zusammenhang mit dem Suprenum-Projekt, das einen großen Parallelrechner zum Ziel hat.

Ein weiteres natürliches Target für Parallelisierung ist die Computergraphik. Diesen Weg hat die Technik bereits lange durch die Bereitstellung entsprechender Hardware beschritten. Die Chance liegt jetzt darin, durch parallele Software komplexere graphische Information in 4D, d.h. in Raum und Zeit, variabel zu verarbeiten, etwa realistisch aussehende bewegte Szenen mit Hilfe von sog. Ray Tracing Algorithmen zu produzieren, etwa für Film und Fernsehen bzw. für die entsprechende Reklamebranche.

Viel weniger offensichtlich, aber nicht weniger wünschenswert wären parallele Algorithmen für Optimierungsaufgaben, Datenbankprogramme und Anwendungen der sog. künstlichen Intelligenz. Die Informatik befaßt sich heute intensiv mit der Entwicklung von Parallelisierungsstrategien für diese Fälle. Eine mögliche Vorgehensweise lehnt sich dabei an die natürliche Intelligenz biologischer Gehirne an, die mit ihren vielen Nervenzellen Information offensichtlich parallel verarbeiten. Der entsprechende Forschungszweig, die Neuroinformatik, der auch in der Arbeitsgruppe der Autoren verfolgt wird, ist allerdings mit der Schwierigkeit konfrontiert, daß die Nervenzellen, die CPU's des Gehirns, einen hohen Verknüpfungsgrad aufweisen: Neuronen sind durch bis zu 10 000 ‚Links', d.h. Synapsen, mit anderen Zellen verbunden. Dennoch sind einige Leistungen natürlich intelligenter Systeme bereits in paralleler Programmierung erstaunlich gut nachvollzogen worden. So ist es in der Arbeitsgruppe der Autoren gelungen, einen Roboter zu programmieren, der sich durch eine enorme Lernfähigkeit auszeichnet und allein durch Eigenbeobachtung durch zwei Stereokameras lernt, seine Gliedmaßen im Raum zu orientieren und gezielt zu bewegen. Der Hintergrund zu diesen Arbeiten wurde in einem mc-Artikel dargestellt [1].

3 Biomoleküle im Parallelrechner

Ausgangspunkt für die in diesem Artikel dargestellte Entwicklung und Programmierung eines Parallelrechners war eine Próblemstellung, welche die Arbeitsgruppe der Autoren seit langem beschäftigt, die Computersimulation von biologischen Makromolekülen. Der Hintergrund zu diesem Problem ist ebenfalls in einem mc-Artikel dargestellt worden [2], auf den wir uns im folgenden wiederholt beziehen werden. Es geht dabei darum, den Computer als intelligentes Mikroskop einzusetzen, welches die strukturellen, elektrischen und dynamischen Eigenschaften von großen Biomolekülen vergrößert (10^6 mal, Struktur), durch Färbung kodiert (elektrische und mechanische Eigenschaften) und verlangsamt (10^{12} mal, Dynamik) darstellt. Dieser Einsatz des Computers stellt ein zunehmend wichtiger werdendes Hilfsmittel zur rationalen Syntheseplanung, etwa im Verbund mit dem sog. genetic engeneering, in der chemischen und pharmazeutischen Industrie dar, ist aber auch ein wichtiges Forschungsinstrument auf dem Weg zu einem Verständnis der Lebensprozesse auf molekularer Stufe.

Das biologische Makromolekül, das unsere Arbeitsgruppe vor allem interessiert, ist ein Komplex aus Proteinen mit zusammen ca. 13000 Atomen, dessen Aufgabe es ist, in photosynthetischen Bakterien Sonnenlicht in biochemisch verwertbare Energie, nämlich ein elektrisches Membranpotential, zu verwandeln.

Das Protein, in Bild 1 farbig dargestellt, stellt den molekularen Grundbaustein für eine biologische Sonnenbatterie dar, wie sie auch in allen Pflanzen zur Ernte des Sonnenlichtes eingesetzt wird. Das Molekül, in eine biologische Membran eingebettet und von Wasser umgeben, ist aber so groß, daß die besten heute verfügbaren Supercomputer nicht ausreichen, um es als ganzes simulieren zu können. Unsere Arbeitsgruppe sah sich in der unangenehmen Lage, trotz eines enormen Aufwandes an Rechenzeit auf zwei Cray XMP 48-Supercomputern und einem eigenen Convex C1 Minisupercomputer diesem Molekül nicht Herr werden

zu können, und wir sahen als einzigen Ausweg den Einsatz eines massiv parallelen Rechners.

Dieser Ausweg schien allerdings zunächst gar nicht gut gangbar zu sein. Die Atome in Makromoleküle üben nämlich alle aufeinander Kräfte aus, die im Sinne der obigen Analyse paraller Prozesse dem absolute Extrem von Kommunikationskanälen entsprechen: jedes Atom, das sich infolge der entwickelten Kräfte bewegt, stellt einen Prozess dar, der aber Information über den Bewegungszustand aller anderen Prozesse haben muß. Gerade als wir mit dem Grübeln über eventuell dennoch geeignete Parallelisierungsstrategie begannen, stießen wir auf einen Artikel von W.D.Hillis [5], dem jungen Vater der Connection Maschine des wohl bekanntesten massiv parallelen Rechners. Hillis machte in seinem Artikel einen einleuchtende Vorschlag. Diesem Vorschlag, der auf einer besonders einfachen Verschaltung der Prozessoren, nämlic einer Ringschaltung, beruht, sind wir nachgegangen, haben sie konkretisiert und den hier beschriebene Rechner gebaut und programmiert.

Eine wichtige Entscheidung bei unserer Arbeit betraf die Frage, ob wir warten sollten, bis Transputerrechne auf dem Mark erhältlich sind, oder ob wir selber einen Rechner bauen sollten. Wir haben uns für die letzter Möglichkeit entschieden, und zwar im wesentlichen aus zwei Gründen: Erstens benötigten wir den Rechn für unsere Arbeit so schnell wie möglich und wir wußten nicht, inwieweit wir den Lieferversprechen d verschiedenen Hersteller trauen konnten: als wir begannen, war noch kein Rechner auf dem Markt erhältlich Ein zweiter wesentlicher Grund war der Preis. Nach einem Überschlag der Kosten sahen wir voraus, da wir die Rechenleistung einer Cray zu einem mindestens hundertmal günstigeren Preis haben konnten un wir sahen hier eine große Chance für die Wissenschaft der Biomoleküle, die wie jede Wissenschaft notorisc unter Geldmangel leidet. Wir wollten also für uns selbst, aber auch für unsere wissenschaftlichen Kollege einen möglichst preisgünstigen Rechner entwickeln, d.h. einen Rechner zum Preis der eigentlichen Hardware Kosten. Es sollte aber betont werden, daß es in der allernächsten Zukunft möglich sein wird, Rechner wi den unsrigen auch käuflich zu erwerben.

3.1 Der numerische Hintergrund

Wir wollen im folgenden Abschnitt kurz die numerischen Rechenschritte in Molekulardynamikprogramme rekapitulieren. Für eine ausführliche Diskussion sei der Leser auf den früheren mc-Artikel verwiesen [2 Bei der Computersimulation geht man von einem sog. „klassischen Modell" des Biomoleküls aus. Dab gelten für die Positionskoordinaten (x, y, z) der Atommassenschwerpunkte die von *Newton* formulierte Bewegungsgleichungen für N Atome $(i = 1, 2, \ldots N)$

$$\begin{aligned} m_i \ddot{x}_i &= -\frac{d}{dx_i} E(x_1, y_1, z_1, x_2, y_2, z_2, \ldots x_N, y_N, z_N) \\ m_i \ddot{y}_i &= -\frac{d}{dy_i} E(x_1, y_1, z_1, x_2, y_2, z_2, \ldots x_N, y_N, z_N) \\ m_i \ddot{z}_i &= -\frac{d}{dz_i} E(x_1, y_1, z_1, x_2, y_2, z_2, \ldots x_N, y_N, z_N) \quad , \end{aligned}$$

wobei x_i, y_i, z_i die Koordinaten des i-ten Atomes sind. Die Funktion E, die von den Koordinaten abhäng gibt die Gesamtenergie des Biomoleküls an.

Die Gesamtenergie setzt sich aus mehreren Beiträgen zusammen, welche den unterschiedlichen Kräfte zwischen den Atomen des Moleküls entsprechen

$$E = E_B + E_{El} + E_\theta + E_\phi + E_{vdW} + E_H + E_I \quad .$$

Der erste Energiebeitrag E_B beschreibt die Bindungen zwischen den Atomen und resultiert in hochfrequente Schwingungen der Atome. Der zweite Beitrag beschreibt die elektrostatische Wechselwirkung im Molekü Es gilt

$$E_{El} = \sum_{\text{Atompaare } i,j} \frac{q_i q_j}{4\pi\epsilon r_{ij}} \quad ,$$

wobei r_{ij} den Abstand zwischen Atomen i, j eines Paares darstellt. Entsprechende Ausdrücke existiere für die anderen Energiebeiträge, wobei zu beachten ist, daß die jeweiligen Energien von der Atomsort abhängen. Dazu enthält das Dynamikprogramm einen komplexen Datensatz von Konstanten, welche di Wechselwirkung zwischen den einzelnen Atom- und Bindungssorten von Biomolekülen beschreiben.

Eine für biologische Makromoleküle spezifische Wechselwirkung, die mit sogenannten Wasserstoffbrücken zusammenhängt, sei hier besonders erwähnt. Eine typische Wasserstoffbrücke ist in Bild 2 dargestellt. Die resultierende Wechselwirkung umfaßt mehrere Atome, nämlich alle Atome, die an der Wasserstoffbrücke direkt beteiligt sind.
Die Integration der *Newton'schen* Bewegungsgleichungen erfolgt nach einem nach Verlet benannten Verfahren. Bei diesem Verfahren wird für jeweils eine Atomkoordinate, z.B. $x_i(t)$ der neue Wert $x_i(t+\Delta t)$ zum Zeitpunkt $t + \Delta t$ nach der Formel

$$x_i(t+\Delta t) = 2x_i(t) - x_i(t-\Delta t) + f(t)(\Delta t)^2/m_i$$

$$f(t) = -\frac{d}{dx_i}E(x_1(t), y_1(t), z_1(t), x_2(t), y_2(t), z_2(t), \ldots x_N(t), y_N(t), z_N(t))$$

berechnet. Zur Anwendung dieser Formel werden Koordinaten $x_i(t-\Delta t)$ und $x_i(t)$ benötigt, d.h. die Koordinaten zu zwei vorherigen Zeitpunkten. Der maximale Integrationsschritt Δt wird durch die schnellsten im System vorhandenen Freiheitsgrade bestimmt und beträgt normalerweise eine Femtosekunde, was 10^{-15} Sekunden entspricht. Die schnellsten biologisch relevanten Prozesse dauern etwa 10 000 derartige Integrationsschritte und es ist deshalb klar, daß die Beschreibung größerer Biomoleküle mit einigen Tausend Atomen numerisch gesehen sehr aufwendig ist. Der weitaus größte Teil der Rechenzeit wird dabei jedoch für die Berechnung der Kraft $f(t)$ im Verlet-Algorithmus verwendet. Die weiter unten angegebenen Benchmarks beziehen sich deshalb auf die Bestimmung dieser Kräfte für alle Atome der jeweiligen Biomoleküle.

4 Problem und Topologie

Die Aufgabenstellung lautet also, die Dynamik großer Moleküle zu simulieren. Um die Bewegung der einzelnen Atome berechnen zu können, ist es notwendig, alle Kräfte, die auf jedes einzelne Atom wirken, zu kennen. Wie bereits in [2] erläutert wurde, setzen sich die Kräfte aus folgenden Anteilen zusammen: Elektrostatische Kräfte, Van der Waals Kräfte, Bindungskräfte, Winkelbiegungen, Dihedralkräfte, Extraplanarkräfte und Wasserstoffbrücken.
Am bekanntesten davon ist wohl der Beitrag der elektrostatischen Kräfte. Diese Kräfte wirken zwischen zwei Atomen und sind abhängig von deren Ladungen q_1, q_2 und ihrem Abstand r:

$$F_1 = \frac{q_1 q_2}{4\pi\epsilon r^2} \qquad (1)$$

Dabei ist F_1 der Betrag der Kraft, die auf Atom 1 wirkt und die von Atom 1 nach Atom 2 gerichtet ist. Die Kraft, die auf Atom 2 wirkt, ist der ersten Kraft entgegengesetzt gleich, d.h. es gilt das Newton'sche Prinzip *ACTIO = REACTIO*.
Da an jedem Atom die Coulombkräfte aller anderen Atome angreifen, ist es notwendig, F_1 bzw. F_2 für alle Atompaare auszuwerten. Für N Atome sind dies $N(N-1)/2$ Paare. Bei einer Molekülgröße von 13000 Atomen, einer Zahl, die dem photosynthetischen Reaktionszentrum (s.o.) entspricht, sind die Kräfte von 84493500 Paaren zu berechnen. Dieser quadratisch mit N wachsende Beitrag beansprucht besonders bei größeren Molekülen den Hauptanteil der Rechenzeit. Daher werden wir uns im folgenden auf diese Kräfte konzentrieren.
In der Natur wirken alle Coulombkräfte gleichzeitig. Herkömmliche Molekulardynamikprogramme werten die Kräfte dagegen nacheinander, Atompaar für Atompaar (sequentiell) aus. Es liegt nahe, die von der Natur demonstrierte Parallelität auch bei der Computersimulation auszunutzen. Es bieten sich verschiedene Möglichkeiten der Realisation an. Naheliegend wäre etwa folgende Aufteilung der Arbeit auf die einzelnen Prozessoren, die in Bild 3 illustriert ist:
Jedem Prozessor wird ein Raumbereich (z.B. Würfel) mit den darin eingeschlossenen Atomen zur Bearbeitung zugeteilt. Diese Vorgehensweise hat jedoch gravierende Nachteile:

- Jeder der von uns verwendeten Transputer (siehe [3]) besitzt nur 4 Links zur Kommunikation mit anderen Transputern. Da an jeden Würfel 6 weitere angrenzen, mit denen Daten ausgetauscht werden müssen, wären zum Aufbau dieser Topologie 6 Links nötig.

- Die ungleichmäßige Verteilung der Atome im Raum führt zu einer ungleichen Auslastung der einzelnen Prozessoren. Dies ließe sich durch eine Verzerrung des Raumgitters verbessern, was jedoch erheblichen Verwaltungsaufwand mit sich bringt.
- Ein nahe einer Würfelgrenze liegendes Atom kann sich im Laufe der Zeit von einem Würfelbereich zum anderen bewegen. Dies kompliziert das Programm und bedingt zusätzliche Kommunikation zwischen den beiden beteiligten Prozessoren. als Problem

Diese Probleme vermeidet eine andere Topologie, die in Bild 4 vorgestellt wird. Dabei sind die Transputer zu einem Ring verbunden. Die Atome werden ungeachtet ihrer räumlichen Anordnung gleichmäßig auf die Transputer verteilt. Die einem bestimmten Transputer zugeordneten Atome werden später als „eigene" Atome des Transputers, alle anderen Atome als die ihm „fremden" Atome bezeichnet. Bei der Berechnung wird wie folgt vorgegangen:

1. Jeder Transputer berechnet die Wechselwirkung der ihm zugeteilten Atome untereinander.
2. Nun kopiert jeder Transputer die Koordinaten „seiner" Atome im Uhrzeigersinn zu seinem Nachbarn weiter. Ebenso empfängt er von seinem anderen Nachbarn dessen Koordinaten („fremde" Koordinaten).
3. Jetzt werden alle Wechselwirkungen dieser Atomgruppen („eigene" Atome mit „fremden" Atomen) untereinander berechnet.
4. Die „fremden" Koordinaten werden zum nächsten Transputer weitergereicht.

Punkt 3 und 4 werden sooft wiederholt, bis alle Wechselwirkungen berechnet sind. Wegen der gleichmäßigen Arbeitsverteilung rechnen stets alle Transputer. Somit zählt diese Topologie zu den effizientesten und wurde deshalb von uns gewählt.

5 Aufbau eines Rechenknotens

Ein Rechenknoten besteht aus folgenden Komponenten, die im Blockschaltbild (Bild 5) zu erkennen sind:

- Transputer IMS T800G20S (32bit Daten- und Adreßbus, 20MHz Taktfrequenz)
- 1 MB dynamisches RAM als Lokalspeicher für Programm und Daten, aufgebaut aus 4 SIP-Modulen mit je 9 256Kb x 1 Chips (100 ns)
- Einem byte-weisen Parity check-Mechanismus
- Treiber
- LED Anzeigen für den Betriebszustand

Es fällt auf, daß in der Liste kein ROM auftaucht. Dies mit gutem Grund, da wir das Transputerfeature ‚boot from link' verwenden und das Programm daher zu Beginn der Berechnungen einmal in das RAM jedes Knotens geladen wird. Die Größe des Lokalspeichers von 1 MB wurde durch verschiedene Faktoren bestimmt: zum einen waren zur Zeit der Entwicklung auf dem RAM-Markt nur 256Kb Bausteine zu erhalten, sodaß 4 MB eine zu große Fläche auf der Platine beansprucht hätten. Zum anderen konnten dadurch mit denselben Mitteln mehr Rechenknoten realisiert werden.
Da unser System sehr viele Knoten enthalten wird (bis zu 60 Prozessoren in einem 19" Gehäuse), muß der Erkennung von Speicherfehlern Aufmerksamkeit geschenkt werden, wie folgende Überlegung zeigt. Man muß zwischen *hard errors* und *soft errors* unterscheiden (siehe [6]). Erstere werden durch die endliche Lebensdauer des Speicherchips hervorgerufen (Bauteil irreparabel zerstört) und sind zu vernachlässigen. Die *soft errors* dagegen sind mit 100 - 1000 FIT (Failure in Time, 1 FIT ist ein Ausfall in einer Milliarde Bauelementestunden) dominant. Sie entsprechen einem Ladungsverlust eines Kondensators (ca. 50 fF) im DRAM durch äußere Einflüsse (Alphastrahlung aus dem Chipgehäuse, Störspannungen) und zerstören

das Bauteil nicht. Bei einem Computer mit 120 Prozessoren ist alle paar Wochen mit einem soft error zu rechnen. Solche Rechenzeiten sind für Molekulardynamik durchaus realistisch und ein einziges falsches Bit im Speicher kann die ganze Rechnung unbrauchbar machen. Daher wird für jedes Byte (der Transputer erlaubt byteweisen Speicherzugriff) ein zusätzliches Prüfbit (Parity) mit abgespeichert und beim Lesen überprüft. So lassen sich alle Ein-Bit-Fehler erkennen (welche nahezu ausschließlich auftreten). In diesem Fall wird dies dem Programm mitgeteilt, welches dann in der Lage ist, den letzten (fehlerhaften) Rechenschritt mit korrekten Daten zu wiederholen. Daher erübrigt sich eine aufwendige Korrektur (wozu 7 Prüfbits für 32 Datenbits nötig wären) auf Hardwareebene.
Die Adreßtreiber separieren die Adressen von den Daten und multiplexen erstere auf den DRAM-Adreßbus. Je drei LEDs auf der Frontplatte geben einen Überblick über den momentanen Betriebszustand eines jeden Knotens:

- Die rote Error-LED ist direkt an das Error-Pin des Transputers angeschlossen. Sie leuchtet auf bei Fehlern wie z.B. Division durch Null, arithmetischem Überlauf sowie falscher Array-Indizierung. Außerdem kann sie durch den Befehl `STOP` explizit gesetzt werden.
- Die gelbe Parity-LED signalisiert einen Parity-Fehler. Ihr Zustand wird in einem Flip-Flop gespeichert, das nur vom Programm aus zurückgesetzt werden kann.
- Die grüne Busy-LED blitzt bei jedem Zugriff auf den externen Speicher kurz auf. Ihre Helligkeit erlaubt daher Rückschlüsse auf die Häufigkeit der Speicherzugriffe. Daraus lassen sich Informationen über den Betriebszustand des Rechners bei der Programmausführung ableiten.

Im folgenden soll auf das Memory Timing etwas näher eingegangen werden (siehe [7]). Gegenüber vielen herkömmlichen Prozessoren zeichnet sich der Transputer durch sein sehr flexibles Memory-Interface aus. Vier nahezu freiprogrammierbare Strobe-Leitungen stellen RAM-Steuersignale wie zum Beispiel RAS, CAS oder Latch-Enables zur Verfügung. Darüberhinaus ist der Transputer in der Lage, den für DRAMs benötigten Refresh durchzuführen. Die je 32 Adreß- und Datenleitungen sind auf einen gemeinsamen Bus gemultiplext. Da die DRAMs jedoch getrennte Daten und Adressen verlangen, müssen diese wieder voneinander getrennt werden. Dies geschieht, wie in Bild 6 und Bild 7 erklärt, folgendermaßen: Mit der steigenden Taktflanke von T1 gibt der Transputer die Adressen aus. Aufgrund der Gatterlaufzeit des Buffers *AM29827* liegen die unteren neun Adreßbits (A2...A10) ca. 30 ns später am RAM an und werden während T2 mit der fallenden RAS-Flanke übernommen. Gleichzeitig speichert das Latch *AM29843* die Adressen A11...A19 zwischen (diese werden zu einem Zeitpunkt benötigt, zu dem bereits Daten am Bus anliegen). Während mit der fallenden Flanke von T3 der Buffer hochohmig wird, gibt das Latch die CAS-Adressen für das RAM frei. Diese werden am Ende von T3 mit der fallenden CAS-Flanke in's RAM übernommen (early-write). In einem Schreibzyklus liegen die Daten seit dem Anfang von T2 am RAM an und können damit jetzt gespeichert werden. Bei einem Lese-Zyklus legt das DRAM die Daten ab T5 auf den Bus, welche der Transputer am Ende von T5 übernimmt. Zu diesem Zeitpunkt werden alle Steuersignale inaktiviert, sodaß mit der steigenden Flanke von T6 der Buszyklus abgeschlossen ist.

6 Die Platine

6.1 Sechs Transputer auf einer Platine

Es war nicht nur unser Ziel, einen Computer mit großer Rechenleistung zu entwerfen, sondern auch, diese Leistung auf möglichst kleinem Raum unterzubringen (siehe Foto 8 und Bild 9). Deshalb wurde beim Platinenlayout eine hohe Packungsdichte angestrebt, um sechs Transputer auf einer Doppeleuropakarte unterzubringen. Dies wurde durch folgende Maßnahmen erreicht:

- Verwendung von 3-Lagen-Multilayer in Verbindung mit einer Leiterbahnbreite und einem Leiterbahnabstand von 0.2 mm.
- Verwendung von passiven SMD-Bauteilen.

- Beidseitige Bauteilbestückung (Bild 9):

  Auf der Platinenoberseite wurden Transputer, Parity-Logik sowie Treiber-ICs montiert. Die Verwendung von speziellen IC-Fassungen ermöglichte es dann, die RAM-SIP-Module auf der Platinenunterseite einzulöten. Diese Maßnahme erlaubte eine extrem kurze Leiterbahnführung, woraus sehr saubere Signalpegel und eine reduzierte Störanfälligkeit resultierten.

- Pufferung der Versorgungsspannung durch Verwendung hochkapazitiver Spannungsschienen („Q Pack"), die stehend zwischen den DIL-Reihen plaziert sind.

Wie bereits erwähnt, ist die Rechner-Topologie durch unser Problem, die Simulation von Makromolekülen bereits festgelegt. Um aber auch für etwaige zukünftige Anwendungen die notwendige Flexibilität zu erhalten, haben wir uns entschlossen, alle Links eines jeden Transputers aus der Platine herauszuführen.

6.2 Zehn Platinen in einem 19"-Einschub

Die gewünschte Topologie kann dann durch eine geeignet beschaltete Backplane realisiert werden. Ebenso ist es möglich, auf dieser Backplane die Link-Verbindungen elektronisch umschaltbar zu machen (durch sogenannte *Link-switches*). Dadurch könnte die Topologie sogar innerhalb eines laufenden Anwenderprogramms binnen weniger μs geändert werden. Zunächst haben wir uns jedoch für eine (preisgünstigere) festverdrahtete Backplane mit der bereits geschilderten Ringtopologie entschieden.
Neben den Linkverbindungen ist auf der Backplane auch noch eine Interruptlogik untergebracht, die es ermöglicht, jeden der 60 Transputer getrennt anzusprechen. Ein 1-aus-16 Dekoder (74LS154) selektiert einen der 10 Platinen und ein 1-aus-8 Dekoder (AM26LS2538) auf jeder Platine wählt dann einen der 6 Transputer auf der Platine aus. Ein zentraler *Enable*-Anschluß aktiviert dann das *event request pin* am Transputer.
Eine Platine beansprucht vier Tiefeneinheiten (= 40.64 mm) in einem 19"-Einschub; das bedeutet, daß 10 Platinen oder 60 (!) Transputer in diesem Einschub Platz finden. Die verbleibenden zwei Tiefeneinheiten werden für Resettaster und Betriebsanzeige genutzt.

7 Das Programm

Wenden wir uns nun der Softwareseite zu. Das Programm selbst ist aus drei Modulen aufgebaut. Das erste Modul bildet ein *Interface* zum PDB-Datenformat (Protein Data Bank), in dem die Koordinaten und Parameter der Atome, welche die interatomaren Kräfte festlegen, abgelegt sind. Dieses Format wird auch von herkömmlichen Molekulardynamikprogrammen benutzt. Das zweite Modul ist ein *Controllerprogramm*, das neben der Datenversorgung des *Rechenprogrammes* auch Überwachungs- und Analysefunktionen beinhaltet. Das dritte Modul stellt das eigentliche Dynamik*rechenprogramm* dar, das auf jedem der Netzwerktransputer abläuft. Die Trennung Controllerprogramm/Rechenprogramm ergibt sich fast zwangsläufig aus den verschiedenen Anforderungen und der Einbindung in die INMOS-Entwicklungsumgebung TDS (Transputer Development System). Das *Controllerprogramm* läuft auf dem Hosttransputer ab (bei uns ein T414 auf einem etwas umgebauten B004 von INMOS in einem IBM-kompatiblen AT) und ist daher in Form eines *EXEs* geschrieben, der üblichen Form eines Userprogrammes. Es hat damit Zugriff auf die hierarchische Foldingstruktur. Das Rechenprogramm läuft auf mehreren Transputern ab und muß daher die Form eines *PROGRAMs* haben.

7.1 TDS-Entwicklungssystem und Folding Editor

Hier sind einige Anmerkungen zur Entwicklungsumgebung, in der das Programm entstand, notwendig. Zentraler Bestandteil des TDS ist der Folding Editor, aus dem heraus auch der Compiler und eigene EXEs gestartet werden. Dem folding („fold" heißt „Falte") am nächsten verwandt wäre ein hierarchisches Fenstersystem. Jede Falte, deren Inhalt zusätzlich durch eine Titelzeile charakterisiert werden kann, ist durch drei Punkte (... Titelzeile) gekennzeichnet und nimmt in geschlossener Form gerade eine Zeile auf dem Bildschirm ein. So kann man die grobe Struktur eines Programms auf einen Blick erkennen (Bild 10). Wird dagegen das fold entfaltet, so gibt es seinen gesamten Inhalt preis, der ebenfalls wieder folds enthalten kann (siehe Programmlistings). Geöffnete folds erkennt man an drei geschweiften Klammern ({{{ Titelzeile). Ihr

Ende wird ebenso markiert (}}}}). So ist mit Hilfe der folds eine übersichtliche und gut strukturierte Gestaltung des Programmtextes möglich. Alle Editorfunktionen, der Compiler und auch Userprogramme (EXEs) werden über Funktionstasten aufgerufen (keine Kommandozeile!) und beziehen sich (soweit sinnvoll) auf den Inhalt des folds, auf dem gerade der Cursor steht. So ist es nur konsequent, die Moleküldatenfiles mit Hilfe eines EXEs in ein fold mit hierarchischer Subfoldstruktur zu schreiben, zumal die Daten dabei gleich geordnet werden können, was schließlich die Ladezeit des Netzwerkes verkürzt. Letztendlich ist es dadurch auch möglich, kleinere Korrekturen an den Daten direkt mit dem Foldingeditor vorzunehmen, ohne das TDS verlassen zu müssen.
Eine Simulation wird nun einfach durch Aufruf des Controller-EXEs auf der Falte mit den Daten des gewünschten Moleküls gestartet.

7.2 Datenfluß

Da ein bidirektionales Hardwarelink zwischen zwei Transputern *zwei* unidirektionalen occam-links (*channels*) entspricht, können im Ring Daten in beiden Richtungen ausgetauscht werden. In der einen Richtung kreisen die Koordinatenpakete, während in der anderen Richtung Kräfte versandt werden.

7.2.1 T800-Rechenknoten (Modul *Rechenprogramm*)

Wie in Bild 11 zu sehen, laufen in diesem Modul vier Prozesse parallel auf jedem T800:

- Ein *Routing-Prozeß*, der entscheidet, ob die ankommenden Kraftdaten für diesen Prozessor bestimmt sind oder ob sie weitergereicht werden sollen. Gehören die Kraftdaten hierher, so werden sie über einen internen occam-Kanal an den Prozeß *Buffer* geleitet. Im anderen Fall passieren sie einen *Multiplexer* und erreichen schließlich den nächsten Transputer.
- Ein *Buffer*, der die Kräfte so lange zwischenspeichert, bis *Berechnung* bereit ist, sie weiterzuverarbeiten. Auf diesen *Buffer* wird bei der Besprechung des Beispielprogrammes noch näher eingegangen.
- Der *Multiplexer* faßt die Kraftdaten aus dem *Routing-Prozeß* und die neu berechneten Kräfte zusammen, um sie an den folgenden Transputer zu übermitteln.
- Der Prozeß *Berechnung* schließlich ist sehr umfangreich und verarbeitet nicht nur die Kraftdaten, sondern auch die Koordinatensätze von den anderen Transputern. Auf ihn wird später ausführlich eingegangen.

Zusätzlich können auch noch einer oder mehrere Pufferprozesse für die Koordinaten eingefügt werden. Sie gleichen unterschiedliche Rechenzeiten aus und ergeben so eine bessere Auslastung des Systems. *Routing-Prozeß* und *Multiplexer* können auch in einem gemeinsamen Prozeß zusammengefaßt werden, was Speicherplatz für Variablen spart.

7.2.2 T414-Rechenknoten (Modul *Controller*)

Die in Bild 12 dargestellte Prozeßstruktur ist völlig anders, da der Hosttransputer als Master ganz andere Aufgaben zu erfüllen hat. Wir erkennen folgende parallele Prozesse:

- Ein *Parity-Prozeß* überwacht den Interrupteingang des T414, der mit dem parity error-Ausgang des Netzwerkes verbunden ist. Tritt irgendwo ein parity Fehler auf, so teilt dies der Prozeß auch allen anderen Prozessen mit. Dies ist wichtig, damit keine fehlerhaften Daten in die Analyse gelangen und abgespeichert werden. Ferner müssen *alle* parallelen Prozesse terminieren, damit auch der Gesamtprozeß endet, was für einen Restart von Bedeutung ist, denn nach einem parity Fehler ist noch nicht alles verloren: ein Aufsetzen auf den letzten gültig berechneten Daten ist möglich. Dies war auch ausschlaggebend bei der Abwägung parity check oder error-detection-and-correction: was in Software geht, muß man nicht in Hardware machen.
- Ein *Pass-through-Prozeß* schließt einfach den Kraftkanal vom Eingang zum Ausgang kurz, denn Kraftdaten werden hier nicht benötigt.

- Die zwei *Buffer-Prozesse* erfüllen wichtige Aufgaben:

  Im worst case-Fall ereignet sich ein Speicherfehler gerade in dem T800, der dem T414 direkt vorgeschal tet ist. Und zwar gerade dann, wenn dessen DMA-Maschine soeben über das Link Werte an den T41 überträgt. Da die Nachricht des parity Fehlers den Master oder gar die Analyse erst nach eine gewissen Verzögerung erreicht, könnte es ohne Buffer schon zu spät sein: fehlerhafte Daten stehen in Restartfile! Dies nun gerade verhindert der Eingangsbuffer (im Bild 12 links).

  Ein weiterer schlimmer Unfall wäre es, wenn ein Speicherfehler das Programm eines der beiden be nachbarten Prozessoren so korrumpieren würde, daß dieser mitten im Datentransfer über ein Lin seinen Dienst einstellt. Um dann den berüchtigten Dead-lock zu verhindern, „opfern" sich die Puffer prozesse: sie können unter Umständen festhängen, aber es ist sichergestellt, daß Master und Analys ordnungsgemäß terminieren und ein gültiges Restartfile erzeugen.

- Der *Masterprozeß* hat alle Fäden in der Hand:

  Er zählt die Umläufe der Koordinaten, reicht sie gegebenenfalls zur Analyse weiter, bestimmt di Rechenzeit, fragt globale Energiewerte ab und hält den User über den Bildschirm auf dem Laufenden

- Das *Analyseprogramm* in seiner einfachsten Version schließlich speichert die berechnete Trajektorie al und hält die Restartfiles auf dem neuesten Stand.

Eine zusätzliche Pufferung der Datenübergabe an das Analyseprogramm entkoppelt Analyse und Berechnung weiter voneinander, sodaß beide ohne gegenseitige Behinderung parallel ablaufen können.

7.3 Flußdiagramm

Schauen wir uns das Berechnungsprogramm, das auf jedem T800 abläuft, etwas genauer an (Bild 13) Während der *Ladephase* erhält jeder Transputer zuerst „seine" Daten, das sind die Koordinatensätze de zu ihm lokalen Atome, sowie Informationen über ihre Bindungen und die zu betrachtenden Winkel. An schließend werden die globalen Daten, das sind Kraftkonstanten und Parameter der verschiedenen Wechsel wirkungspotentiale, an alle Transputer verteilt.
Ein spezielles Umschaltsignal steuert den Übergang in die eigentliche Berechnungsphase. Dieser Pro grammteil terminiert nicht, doch kann der Hosttransputer (T414) ein Resetsignal auslösen, welches da gesamte Netzwerk neu startet.
Die Berechnung selbst ist ein sequentielles Programm, das aber über die eingezeichneten Kanäle (Bild 11 mit anderen Prozessen, die zum Teil auf anderen Transputern ablaufen, kommuniziert. Nach einer Ini tialisierung der lokalen Variablen (Zeiger in diverse Listen, Kraftakkumulatoren, etc.) werden zuerst all Wechselwirkungen der Atome ausgerechnet, die zum jeweiligen Transputer lokal sind. Das EXE *Controlle* hat während der Ladephase dafür gesorgt, daß alle (internen) Listen gerade die richtigen Einträge, die z den lokalen Atomen passen, besitzen. Dieser Teil unterscheidet sich kaum von den herkömmlichen, sequen tiellen Dynamikprogrammen. Der folgende *Startschritt* ist neu: der Transputer reicht „seine" Koordinate an seinen rechten Nachbarn weiter. Wie im Kapitel *Topologie* erläutert, werden zur Berechnung der externe Wechselwirkungen nun die Koordinaten vom linken Nachbarn verwendet. Die Elektrostatikberechnung etw ist nun einfach eine Doppelschleife über alle internen und alle externen Atome. Die hier berechneten Kräft werden zum einen auf die lokalen Kraftakkumulatoren addiert, zum anderen werden sie mit umgekehrten Vorzeichen an den linken Nachbarn zurückgegeben, so daß dieser sie ebenfalls in seine Kraftakkumulatore addieren kann. Dann werden die *fremden* Koordinaten an den rechten Nachbarn weitergegeben, und da Spiel wiederholt sich. Bei 12 Transputern erreichen so nach 12 Schritten jeden Transputer wieder die eigene Koordinaten. Diese werden aber nicht gebraucht und können daher gelöscht werden (Stoppschritt); ohn diesen Stoppschritt wäre der Ring im nächsten Zyklus nicht bereit, den Startschritt auszuführen: jeglich Aktivität käme zum Erliegen (sogenannter Deadlock). Da nun die Gesamtkraft auf jedes interne Aton bekannt ist, können in einem Integrationsschritt (nach *Verlet*, siehe [9]) die neuen Positionen der Atom berechnet werden. Schließlich werden noch die skalaren Größen, wie etwa die verschiedenen Energiebeiträge an den Master und die Analyse übermittelt. So kennzeichnen etwa stark veränderliche Werte der Gesamten ergie Ungenauigkeiten der Berechnung, da in dem berechneten System die Gesamtenergie eigentlich konstan bleiben müßte.

7.4 SEQ versus PAR

Hier drängt sich nun die Frage auf, warum diese grobe Art der Parallelisierung (auf jedem Transputer läuft ein im wesentlichen sequentielles Programm) gewählt wurde. Bei strikter Anlehnung an die Natur würde man für jedes Atom einen (parallelen) Prozeß kreieren und derartige Prozesse dann auf die verfügbaren Transputer verteilen. Dies würde das Programm vereinheitlichen (keine „eigenen" und „fremden" Atome mehr), bringt aber einen Geschwindigkeitsverlust. Da wir jedoch an einem möglichst schnellen Programm interessiert sind, steht dem folgendes im Wege:

1. Trotz des hardware schedulers ist bei jedem parallelen Prozeß ein gewisser scheduling overhead und damit eine Programmverlangsamung nicht zu vermeiden.

2. Die 4 KB on-chip RAM stellen eine wertvolle Resource dar. Bei nur einem Prozeß je Transputer kann dieser *alle* seine skalaren Variablen dort ablegen und auf sie mit maximaler Geschwindigkeit zugreifen. Bei mehreren (10—100) parallelen Prozessen je Transputer, die alle ihren eigenen workspace benötigen, finden nur die Variablen sehr weniger Prozesse im on-chip RAM Platz; kostbare Zeit verstreicht ungenutzt beim Zugriff auf das externe RAM.

3. Ein Transputer für sich ist immer noch ein sequentieller Rechner. Daher kann von den vielen parallelen Prozessen doch immer nur einer je Transputer aktiv sein, während alle anderen auf die Zuteilung des Prozessors warten.

4. Neben Punkt 2. aber ist der schwerwiegendste Grund, der gegen einen Prozeß pro Atom spricht, das Auftreten von Mehrkörperwechselwirkungen und der damit gegenüber der SEQ-Lösung stark ansteigende Speicherbedarf.

   Sind an einer Wechselwirkung mehr als zwei Atome gemeinsam beteiligt, so müssen zu ihrer Berechnung die Koordinaten *aller beteiligten* Atome bekannt sein. Dies ist etwa bei der Winkelwechselwirkung der Fall, bei der drei Atome einen Winkel definieren. Im ungünstigsten Fall muß die Berechnung so lange aufgeschoben und müssen die Koordinaten eines Atoms zwischengespeichert werden, bis alle externen Atome an dem Prozeß vorbeigekommen sind. Dies wiederum kann für alle Wechselwirkungen (Dihedral- und Extraplanarwechselwirkung benötigen Koordinateninformation von je zwei Atomen) gelten, sodaß unter Umständen in jedem Prozeß die Koordinaten von fünf Atomen gespeichert werden müssen.

   Bei der mehr sequentiellen Lösung dagegen mitteln sich diese Effekte weitgehend heraus, sodaß bei 300 Atomen je Prozeß nur zusätzlicher Speicherplatz für etwa 100 Atome bereitgehalten werden muß. Dies ist nuf 1/15 des Bedarfs der vollständig parallelen Lösung.

   In diesem Zusammenhang müssen die Wasserstoffbrückenbindungen (s.o.) erwähnt werden, da sie nur schwer in das Parallelisierungsschema passen. Es sind nämlich bis zu vier Atome an einer Bindung beteiligt und im Prinzip steht jeder Akzeptor mit jedem Donator in Wechselwirkung. Dies hat (bei beiden Verfahren) hohen zusätzlichen Speicherbedarf zur Folge. Diese Art der Wechselwirkung ist in der aktuellen Programmversion noch nicht enthalten.

Aus den angegebenen Gründen parallelisieren wir nur da, wo auch wirklich parallel gerechnet werden kann: auf verschiedenen Transputern!

8 Kommunikation im Ring

Wie geht nun die Kommunikation im Ring vor sich? Der Einfachheit halber beschränken wir uns in der folgenden Diskussion auf die elektrostatische Wechselwirkung, die eine reine Zweikörperkraft ist. Ferner nehmen wir an, es seien alle Paare zu berücksichtigen, d.h. jedes Atom wirkt auf jedes andere ein. Bei tatsächlichen Dynamiksimulationen werden dagegen, um Rechenzeit zu sparen, oft Wechselwirkungspaare jenseits eines bestimmten Abstandes vernachlässigt [2, Seite 52]. Nehmen wir an, daß ein Rechner mit sechs Transputern mit der in Bild 4 dargestellten Ringtopologie zur Verfügung steht. Zunächst ist klar, daß nicht jeder Transputer alle externen Wechselwirkungen berechnen darf, denn sonst würden die Kräfte doppelt

berechnet (Transputer 0 rechnet die Wechselwirkung 0 mit 3 und Transputer 3 rechnet 3 mit 0, was na dem physikalischen Prinzip *ACTIO = REACTIO* das gleiche ist). Jeder Transputer darf also nur mit d Hälfte der anderen „wechselwirken", und zwar so geschickt, daß trotzdem alle Paare berechnet werden. A erste Lösung schlagen wir die Aufteilung in Tabelle 1 vor. Bei dieser Aufteilung wird keine Wechselwirku

0	rechnet	0 5 4
1	rechnet	1 0 5
2	rechnet	2 1 0
3	rechnet	3 2 1
4	rechnet	4 3 2
5	rechnet	5 4 3

Tabelle 1: Falsche Aufteilung der Rechenlast

mehr doppelt gerechnet. Doch bei genauerem Hinsehen entdecken wir, daß die Wechselwirkung zwisch den Atomen des Transputers 0 und denen von Transputer 3 fehlt. Die richtige Aufteilung ist in Tabelle

0	rechnet	0 5 4 3
1	rechnet	1 0 5 4
2	rechnet	2 1 0 5
3	rechnet	3 2 1
4	rechnet	4 3 2
5	rechnet	5 4 3

Tabelle 2: Aufteilung der Rechenlast

angegeben. Leider muß man bei dieser Aufteilung in Kauf nehmen, daß während eines Zeitschrittes nur d Hälfte der Transputer rechnen. Bei einer *ungeraden* Zahl von Transputern läßt sich eine bessere Aufteilu erreichen. Allgemein gilt für große Transputerzahlen, daß $2N$ Transputer dieselbe Rechenleistung erbringe wie $(2N - 1)$ Transputer.

9 Do it Yourself—ein vereinfachtes Beispielprogramm

Um die wesentlichen Aspekte unseres Molekulardynamikprogrammes zu demonstrieren, haben wir da Beispielprogramm aus dem früheren mc-Artikel zum Thema ‚Moleküle im Computer' [2, Seite 54f] in oc cam II umgeschrieben und parallelisiert. Das Programm ist in Bild 14 aufgelistet. Das Listing illustrie einerseits die diskutierte Parallelisierungsstrategie, andererseits gibt es dem Leser ein Gefühl für die Sprach occam II. Wer selbst über einen oder mehrere Transputer sowie einen occam II-Compiler verfügt, muß da Programm aber nicht abtippen — gegen eine Schutzgebühr ist es auch auf Diskette zu erhalten (siehe [10]) Trotz einiger Vereinfachungen kann das Programm nicht in seiner vollen Länge abgedruckt werden. Dah sind einige folds, deren Inhalt für das unmittelbare Verständnis verzichtbar erscheint, nicht geöffnet, so dern nur mit ihrer Titelzeile aufgeführt (durch „(NO LIST)" gekennzeichnet). Bei der Druckausgabe kan die sonst sehr übersichtliche Foldingstruktur nur sehr unvollkommen wiedergegeben werden, da viele fold gleichzeitig offen sind. Beim Arbeiten am Bildschirm aber ist das anders; so wurde dieses Programm ohn einen einzigen Papierausdruck erstellt.

9.1 Die Physik im Beispiel

Anstelle der echten molekularen Kräfte wird hier ein einfaches, anschaulicheres Potential verwendet, nämlic das von freien Atomen, die sich in einem zwei-dimensionalen Kasten mit elastischen Wänden befinden. D Kasten wird durch die zwei Seitenlängen box.x und box.y und die Breite der Randzone box.hardness, di

Abbildung 1: Reaktionszentrum der Photosynthese

Abbildung 2: Wasserstoffbrückenbindungen

Abbildung 3: Würfel-Topologie

Abbildung 4: Ring-Topologie

Abbildung 5: Blockschaltbild eines Rechenknotens

Abbildung 6: Memory Timing

Abbildung 7: Memory-Ansteuerung

Abbildung 8: Titelfoto

Abbildung 9: Schematische Zeichnung einer Platine mit vergrößerter Darstellung eines Rechenknotens

Abbildung 10: Übersichtliche Darstellung der Grobstruktur mit geschlossenen folds

Abbildung 11: T800 Prozeßstruktur

Abbildung 12: T414 Prozeßstruktur

Abbildung 13: Flußdiagramm

Abbildung 14: Listing des occam II Beispielprogrammes

Atome durch den Radius *atom.radius*, die Masse atom.mass und die Breite ihrer Randzone atom.hardness beschrieben. Je schmäler die Randzone ist, desto härter ist das Potential und umso kleiner muß der Integrationszeitschritt gewählt werden. Außerdem kann man noch die Höhe der Potentialschwellen für den Stoß von Atomen untereinander (Ea) sowie für die Wechselwirkung von Atomen mit der Wand (Eb) angeben. Während die Rechenzeit für die Atom-Wand-Wechselwirkung linear mit der Zahl der Atome ansteigt, steigt die Rechenzeit für die Atom-Atom-Wechselwirkung genau wie bei echter Molekulardynamik quadratisch. Die Integration der Bewegungsgleichungen wird mit dem *Verlet*-Verfahren[9] durchgeführt, das numerisch sehr stabil ist.

9.2 Das parallele Programm

Auch dieses Beispielprogramm hat dieselbe Grundstruktur wie das echte Programm — kein Wunder: ist es doch durch Vereinfachung daraus entstanden! occam-Neulingen dürfte am Programm am meisten das strikte Einrücken und das Fehlen von, aus Pascal gewohnten, *begin—end* Konstruktionen auffallen. Beides hängt zusammen, denn in occam werden die Gültigkeitsbereiche durch Einrücken um zwei Stellen markiert.
Das *EXE Interface* dient zum Aufbau eines Datenfolds. Dazu fragt es den Benutzer interaktiv einige Werte; andere Werte sind zum Teil in der library vorgegeben oder werden – wie die Koordinaten und Geschwindigkeiten – mit Hilfe des Zufallszahlengenerators erwürfelt. Alle Werte werden in einem neu kreierten fold gespeichert. Nachdem die Daten einmal im fold abgelegt sind, können sie jederzeit mit dem Editor geändert werden.
Ist das Datenfold erst einmal erstellt, so kann das *EXE Controller* darauf starten. Zuerst setzt es mit Hilfe des Programmes *clear* alle Parityflags zurück und löscht den Speicher aller Rechenknoten (natürlich geschieht dies parallel). Nachdem jeder Knoten mit dem (für alle gleichen) Programm versorgt wurde, beginnt der *Loader* mit dem Austeilen der Datenpakete. Dazu muß er zuerst berechnen, wieviele Atome jeder Transputer als ihm „eigen" betrachten kann (dies ist in der Variablen basis abgelegt). Damit die Lastaufteilung möglichst gleichmäßig ist, darf der Unterschied in der Zahl der eigenen Atome zweier Transputer maximal 1 sein (im Allgemeinen wird die Zahl der Atome kein ganzzahliges Vielfaches der Transputerzahl sein).
Die eigentliche Ladearbeit übernimmt nun die SC (Separately Compile part) *load.T800*. Eine SC ist ein Programmmodul, das der Compiler einzeln übersetzen kann und das er, falls es nicht verändert wurde, beim nächsten Übersetzungslauf direkt linken kann, ohne es neu übersetzen zu müssen. Für jeden T800 wird nun der Reihe nach der komplette Datensatz eingelesen und übermittelt. Dabei werden gleichartige Daten (z. B. x-Koordinaten) in ein Array zusammengefaßt und gemeinsam übermittelt. Dies bringt den Vorteil, daß im fold alle Daten zu einem Atom beieinander stehen, was für manuelle Änderungen mit dem Editor sinnvoll ist, während es für die Berechnung günstiger ist, wenn alle artgleichen Daten zusammenstehen. Abschließend werden noch die globalen Daten übertragen (die für alle gleich sind), wie Energiekonstanten und Integrationszeitschritt. Das Auftreten von letzterem bewirkt bei den Rechenknoten auch das Umschalten von der Ladephase in den Rechenbetrieb.
Betrachten wir nun gleich die Ladephase aus der Sicht des Programmteils (fold *Loader*): zuerst wird, wie im Teil „Kommunikation im Ring" beschrieben, die Zahl der externen Berechnungen (Variable calculate) und die Zahl der von außen kommenden, zu addierenden Kräfte (Variable add) festgelegt (im fold *calculate*). Im anschließenden fold *load* lädt jeder T800 zuerst die nur für ihn bestimmten, lokalen Daten und gibt von da an alle einlaufenden Daten weiter. So erhält jeder „seine" Daten. Das Eintreffen des Integrationszeitschrittes schaltet die Variable *loading* auf FALSE um und die Berechnung beginnt.
Besprechen wir hier zuerst den Hauptteil im fold *coordinates*: Die Funktion f bestimmt die Form des wirksamen Potentials, sowohl zwischen den Atomen als auch mit der Wand. Das Potential darf nicht zu hart (hardness $\ll$ x) sein, weil sonst nur sehr kleine Zeitschritte gerechnet werden könnten. Da das System konservativ ist, sollte die Gesamtenergie konstant bleiben. Bei zu großem Zeitschritt registriert man aber ein kontinuierliches Ansteigen, was daher rührt, daß sehr schnelle Atome in einem einzigen Zeitschritt Strecken zurücklegen, die von derselben Größenordnung sind wie die Dicke der Potentialbarrieren. So kann es im ungünstigsten Fall vorkommen, daß sich ein Atom plötzlich auf der anderen Seite der Barriere findet, ohne jemals mit der Wand in Berührung gekommen zu sein. Seine potentielle Energie hat zugenommen, ohne daß seine kinetische Energie beeinflußt wurde.
Zurück zum Programm: Jeder Knoten berechnet zuerst die internen Wechselwirkungen. Dieser Teil unterscheidet sich nicht von dem auf einem einzelnen, sequentiellen Rechner ablaufenden Programmteil. Die

Behandlung der externen Atome erfolgt genauso, wie bei der Erläuterung des Flußdiagrammes Bild 13 beschrieben. In Abhängigkeit von add.count und add werden auch externe Kräfte aufaddiert. Wieviele externe Kräfte zu erwarten sind, kann man sich anhand der Tabelle 2 leicht selbst überlegen und sein Ergebnis mit der Formel im fold *calculate* vergleichen.
Ehe wir das Programm wieder verlassen und uns dem *EXE Controller* zuwenden, sollten wir unser besonderes Augenmerk der unscheinbaren Prozedur *buffer process (Fuer Addition)* zuwenden. Wer die Möglichkeit hat, sollte das Programm unbedingt einmal ohne diesen Prozess ausprobieren. Es kann dann nämlich sehr leicht der berüchtigte Deadlock auftreten. Was heißt das?
Bei ungeschickter Programmierung kann folgende Situation auftreten:

```
CHAN OF MOL.KOORD Koord.in, Koord.out:
CHAN OF MOL.FORCE Force.in, Force.out:
PAR
  SEQ --Prozess 1
    Koord.in ? coordinates
    Force.out ! forces
    Koord.out ! coordinates
    Force.in ? forces
  SEQ -- Prozess 2
    Force.out ! forces
    Force.in ? forces
    Koord.out ! coordinates
    Koord.in ? coordinates
```

Keiner der beiden parallelen Prozesse kann hier weiterarbeiten, weil der eine jeweils auf den anderen wartet. Und genau so etwas kann auch im Programm auftreten, wenn der Pufferprozess zur Kraftaddition fehlt; nur ist die Lage dort deutlich verzwickter. Ohne auf Details einzugehen: es liegt daran, daß der routing process nicht mehr arbeiten kann, wenn er Kraftdaten zur Berechnung schicken will, diese aber gerade mit Anderem beschäftigt ist. Und genau da greift der buffer ein: er speichert die Kräfte so lange zwischen, bis die Berechnung zu ihrer Aufnahme bereit ist. Der routing process ist damit von ihnen befreit und kann ungehindert arbeiten.
Wer selbst mit Transputern experimentieren kann, sollte den Puffer für die Koordinaten einmal entfernen. Es kommt dann zwar nicht zum deadlock, doch steigt bei kleinen Atomzahlen die Berechnungszeit deutlich an, während sie bei sehr großen Zahlen leicht absinkt. Auch Experimente mit dem *PRI PAR* sind sehr interessant. PRI PAR unterscheidet die beiden Prioritätsebenen des Transputers (siehe [4, Seite 69f]).
Wenden wir uns abschließend noch einmal dem *EXE Controller* zu. Der Masterprozeß arbeitet wie ein T-Verteiler: zum einen reicht er die Koordinaten zum Ausgangspuffer weiter, zum anderen gibt er sie in festgelegten Intervallen an das Analyseprogramm. Dieses zeichnet die Blockgrafik auf den Schirm oder speichert die Daten ab. Eine Kette zusätzlicher Puffer für die Analyse erlaubt es, die Koordinaten aller Atome zwischenzuspeichern. Ähnlich wie beim Kraftpuffer kann so erreicht werden, daß die Transputer schon weiterrechnen können, auch wenn das Bild am Schirm noch nicht ganz aufgebaut ist. Dies ist sehr sinnvoll, weil die Bildaufbauzeit (leider) in der Größenordnung der Rechenzeit oder darüber liegt. Wenn man außerdem noch mit PRI PAR den Datentransfer gegen die Bildschirmausgabe priorisiert, bekommt man ein echtes Gefühl für Parallelität: die Berechnung vieler Schritte und die Ausgabe einer Konstellation erfolgen voll parallel.

10 Wirklich eine CRAY? — Benchmarkergebnisse

Will man die Leistung zweier Computersysteme vergleichen, so benutzt man dafür Benchmarkprogramme; künstliche Befehlsfolgen, die keinem realen Problem entstammen, aber den Anspruch erheben, „typisch" für viele Programme zu sein. Vergleicht man dann die Ausführungszeiten dieser Programme auf verschiedenen Computern, so kommt man zu den bekannten Whetstone-, Dhrystone-, Savage-, etc. Werten. Maschinen, die dabei sehr gut abschneiden, können aber dann bei der Lösung eines realen Problems, das keineswegs „typisch" sein muß, enttäuschen — und umgekehrt! Ebenso sind die Angaben der Prozessorhersteller über

MIPS und MFLOPS nur als grober Richtwert zu verstehen; die Leistung, die in einer realen Anwendu erreicht wird, kann von den Herstellerangaben stark (nach unten) abweichen. Oft bringt ein guter Compil auf einer schwachen Maschine mehr, als eine gute Maschine mit einem mäßigen Compiler bieten kann (sie hierzu auch Tabelle 4 und vergleiche C-Compiler und FORTRAN-Compiler (XPLOR)). Es ist also b Benchmarkwerten stets Vorsicht geboten.
Wirklich aussagekräftige Werte über die Gespanne Hard- und Software erhält man nur, wenn man d Ausführungszeiten des Anwendungsprogrammes vergleicht. Dies gilt umso mehr, wenn man die Leistu sequentieller Rechner oder die von Vektorrechnern mit der eines parallelen (MIMD, Multiple Instructi Multiple Data) Rechners vergleichen will: ein Problem läßt sich eben gut parallelisieren, ein anderes nich Aus diesem Grund erscheinen in der folgenden Aufstellung Tabelle 3 die Laufzeitangaben für echte Molek lardynamikrechnungen, wobei die entscheidenden Vergleichsprogramme für die jeweiligen Computer opt miert wurden. An dieser Tabelle sind einige Eigenschaften sehr schön zu erkennen:

	Alanin, 66 Atome	PTI, 568 Atome	RC, 3634 Atome
Elektrostatik und vdW-WW, 1 T800	0,13	9,2	374,0
alle Wechselwirkungen, 1 T800	0,16	10,0	392,6
alle Wechselwirkungen, optimiert, 1 T800	0,15	8,7	339,7
Elektrostatik und vdW-WW, optimiert, 12 T800	0,03	0,8	31,3
alle Wechselwirkungen, optimiert, 12 T800	0,06	0,9	31,6

Tabelle 3: Benchmarkergebnisse für Molekulardynamik. Angegeben ist jeweils die Rechenzeit für eine Dynamikschritt in Sekunden. Alanin, PTI (Pancreatic Trypsin Inhibitor), RC (Photosynthetisches Reak tionszentrum von *Rhodopseudomonas viridis*) sind verschiedene biologische Makromoleküle

- Die Rechenzeit steigt nahezu quadratisch mit der Zahl der Atome.
- Die Rechenzeit für die Elektrostatik- und vdW-Berechnungen beträgt zwischen 50% und 99% d gesamten Rechenzeit. Dieser Anteil wird umso größer, je mehr Atome das Molekül hat.
- Der Rechenzeitgewinn von 12 Transputern gegenüber einem liegt zwischen 2,5 bei kleinen Moleküle und 10,75 bei großen.

All das ist leicht zu verstehen: Je mehr Atome im Molekül sind, umso mehr dominiert die quadratis wachsende Elektrostatik- und vdW-Wechselwirkung die Rechenzeit und prägt auch der Gesamtrechenze den quadratischen Charakter auf. Der nur mäßige Gewinn an Rechenzeit bei 12 Transputern im Falle v Alanin liegt mit in der Struktur des Programmes begründet: bei nur fünf Atomen je Transputer macht si der overhead für Schleifen und das Starten von Kommunikation deutlich bemerkbar. Auch verschlechter sich so das Verhältnis von Rechenzeit zu Kommunikationszeit (die erste sollte stets sehr viel größer als d zweite sein) sehr. Dies ist in der Praxis aber ohne Bedeutung, da bei der Berechnung der Dynamik große Moleküle, für die das Programm in der Regel eingesetzt wird, der overhead nur mehr einen verschwindend Anteil hat. Der Faktor 10,75 liegt außerdem sehr nahe am theoretisch möglichen Faktor von 11,15. (D Faktor 12 kann aus den oben genannten Gründen nicht erreicht werden.) Dieses gute Skalierungsverhalt rechtfertigt auch die Extrapolation von 12 Transputern auf 50 Transputer.
Vergleichen wir nun die Transputerrechenzeiten mit denen anderer Rechner, wie in Tabelle 4. Zwischen d 31,6 s der 12 Transputer und den 7,8 s der CRAY-XMP am Institut für Plasmaphysik (IPP) in Garchin liegt nur noch ein Faktor 4! Dieser Faktor ist aber mit weniger als 50 Transputern leicht einzuholen.
Einige Werte in Tabelle 4 konnten nicht direkt gemessen, sondern mußten hochgerechnet werden, da d zur Verfügung stehende Speicher der aufgeführten Rechner für die Berechnung aller Wechselwirkungen mi XPLOR nicht ausreicht.

VAX 11-750	CONVEX C1		CRAY-XMP
XPLOR	C-Pgm.[11]	XLPOR	XPLOR
2 Stunden (geschätzt)	350 s	147 s (hochgerechnet)	7,8 s (hochgerechnet)

Tabelle 4: Benchmarkergebnisse für Molekulardynamik am Reaktionszentrum (RC). Angegeben ist jeweils die Rechenzeit in Sekunden für einen Dynamikschritt bei Berücksichtigung aller Wechselwirkungen. Hochgerechnete Werte wurden von PTI auf das Reaktionszentrum extrapoliert.

11 Kosten für einen Rechenknoten

Um die behauptete Aussage *Eine* CRAY *für 100.000 DM* zu untermauern, findet sich in Tabelle 5 eine Aufstellung über die anteiligen Kosten eines Rechenknotens. In den Preisen ist die Mehrwertsteuer bereits

Artikel	Firma	Preis [DM]
Transputer IMS T800G20S	E2000, München	1042.25
Dynamisches Ram, HB561409B-10	Termotrol, München	471.75
Stromversorgung Typ LFS-49-5	Lambda, Achern	46.25
Lüfter, Typ V72AB-220VAC	Bedeke, Dinkelsbühl	4.96
sonstige ICs	verschiedene	28.57
passive Bauteile	verschiedene	33.67
Gehäuse	Multimechanik, Filderstadt	3.91
Spannungsschienen	MPS, Feldkirchen-Westerham	67.22
Platinen	HFW, München	198.48
Fassungen	verschiedene	10.75
Gesamt		1907.81

Tabelle 5: Kostenaufstellung für einen Rechenknoten

enthalten. Leider müssen die Ausgaben für den Speicher der ungünstigen Marktlage angepaßt werden. Trotz eines Kostenanstiegs um 300% sind zur Zeit die benötigten SIP-Module überhaupt nicht zu erhalten. Deswegen sind bis jetzt auch erst 12 T800-Rechenknoten im Einsatz; 12 weitere T800 lagern schon in der Schreibtischschublade und warten auf Speicherchips.
Der Tabelle 5 entnimmt man, daß sich mit etwa DM 2.000 pro Knoten ein Netzwerk aus 50 Transputern für 100000 DM realisieren läßt. Extrapoliert man die vorher aufgeführten Benchmarkwerte, so kommt man zu der Aussage der Titelzeile.
Insgesamt hat uns der Rechner mit 12 Transputern also etwa 24000 DM gekostet. Die Geldmittel dafür und für einen weiteren Ausbau des Rechners wurden uns vom Freistaat Bayern über das Ministerium für Wissenschaft und Kunst, vom Bundesministerium für Forschung und Technologie sowie der Deutschen Forschungsgemeinschaft im Rahmen des Sonderforschungsbereiches „Primärprozesse der bakteriellen Photosynthese" zur Verfügung gestellt.

Literatur

[1] J. Buhmann, R. Divko, H. Ritter, K. Schulten: „Physik und Gehirn", mc 9/87 (1987) 108—120

[2] H. Treutlein, A. Windemuth, K. Schulten: „Molecular Design", mc 1/88 (1988) 46—57

[3] Gerd Häusler: „Die mc-Transputerkarte", mc 4/88 (1988) 38—44

[4] Gerd Häusler: „Die Assemblersprache des Transputers", Teil 2, mc 7/88 (1988) 69—73

[5] W. D. Hillis/J. Barnes: „Programming a highly parallel Computer", Nature Vol.326, 5.3.1987 27—30

[6] Karlheinz Fleder: „Schaltkreise zur Erkennung und Korrektur von Fehlern in Speichersystemen. EDAC = Error Detection And Correction", TI-Applikationsbericht EB161, Dezember 1984

[7] INMOS: „IMS T800 Transputer", Preliminary Data, April 1987, #42108200

[8] Brooks et al: „CHARMM: A Program for Macromolecular Energy, Minimization, and Dynamics Calculations", 1982

[9] L. Verlet, Phys. Rev., 159.98 (1967)

[10] Das Beispielprogramm (einschließlich aller nicht geöffneten Folds) kann auch auf Diskette bezogen werden: Gegen Einsendung einer formatierten, leeren IBM-Diskette (360KB, $5\frac{1}{4}$") und eines frankierten selbstadressierten Rückumschlages sowie einer Schutzgebühr von DM 10 (Schein) an folgende Adresse

Technische Universität München

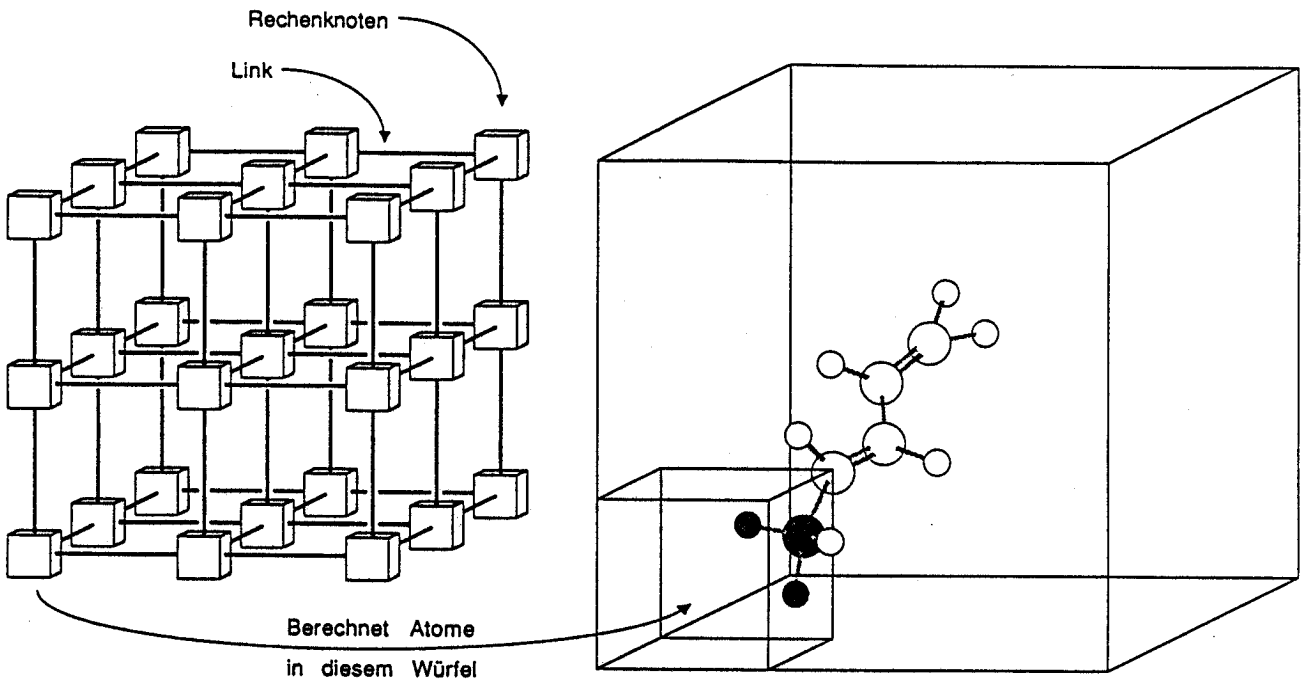
Physik Department, T30

zu Händen Herrn Helmut Heller

James Franck Straße

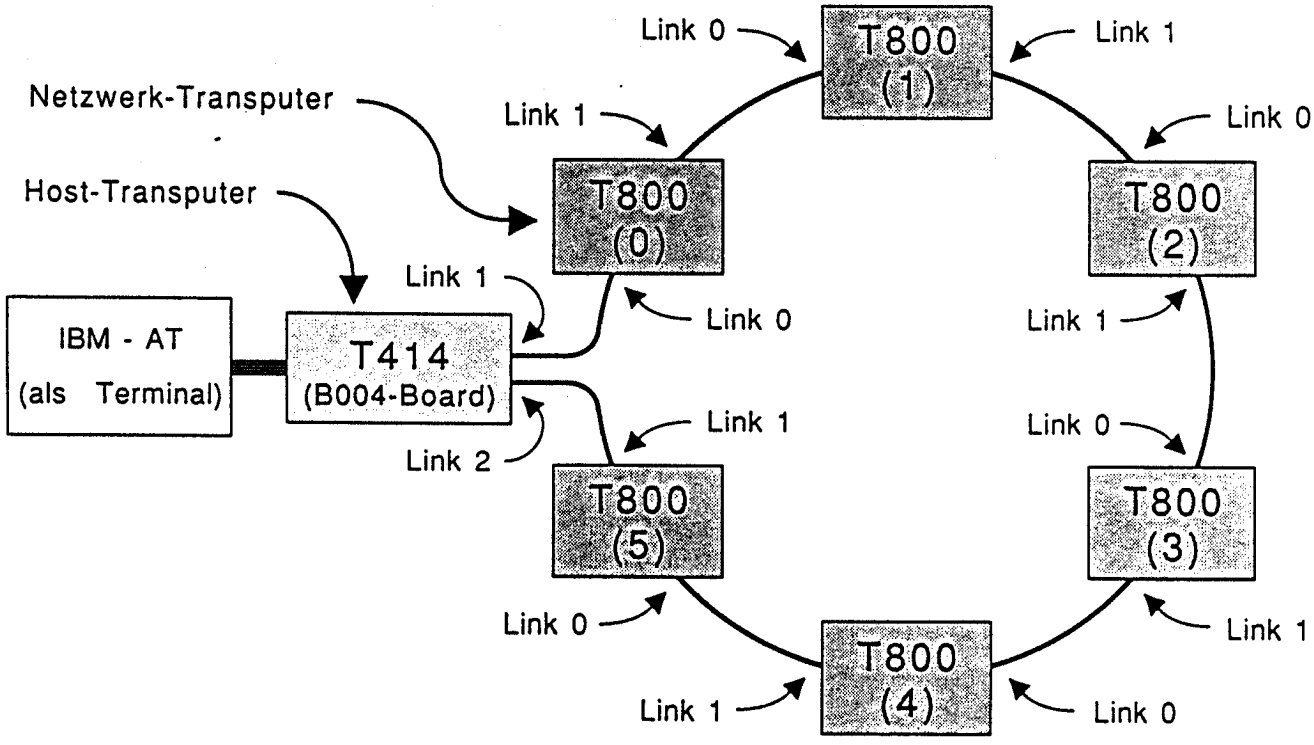
8046 Garching bei München

[11] Molekulardynamikprogramm in C, geschrieben von Andreas Windemuth, angelehnt an CHARMM.



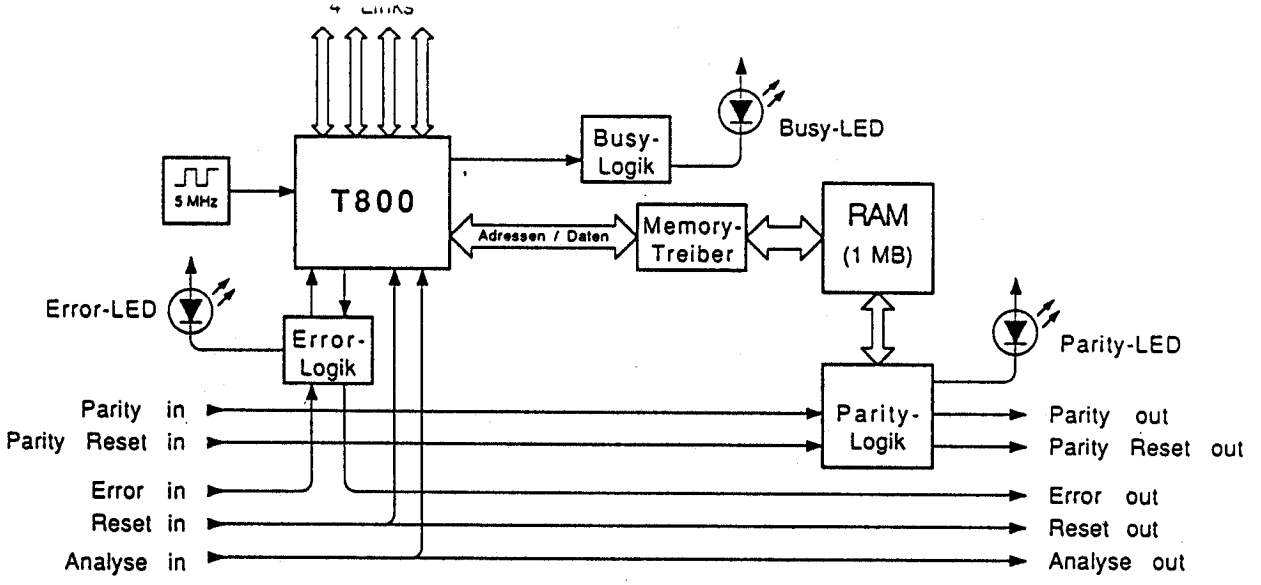
Würfel - Topologie

Abb.3



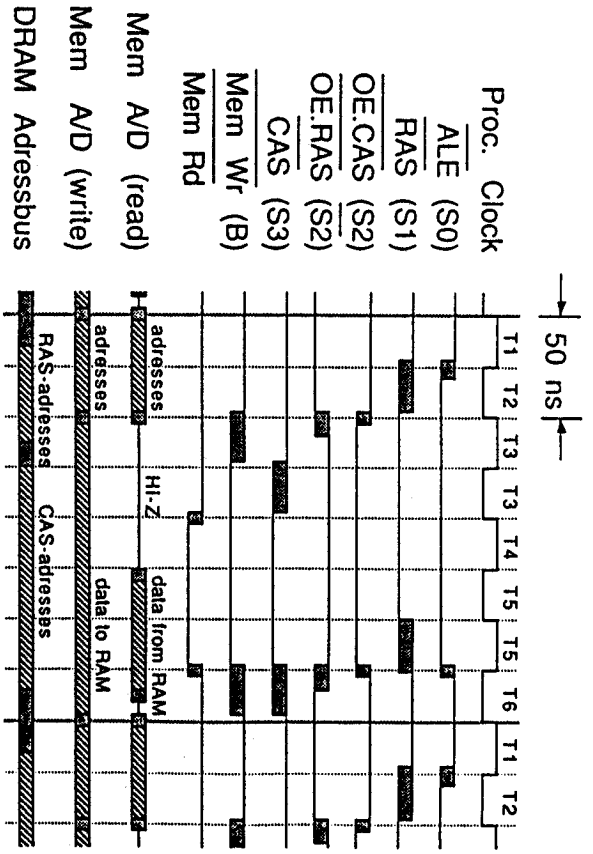
Ring - Topologie

Abb. 4



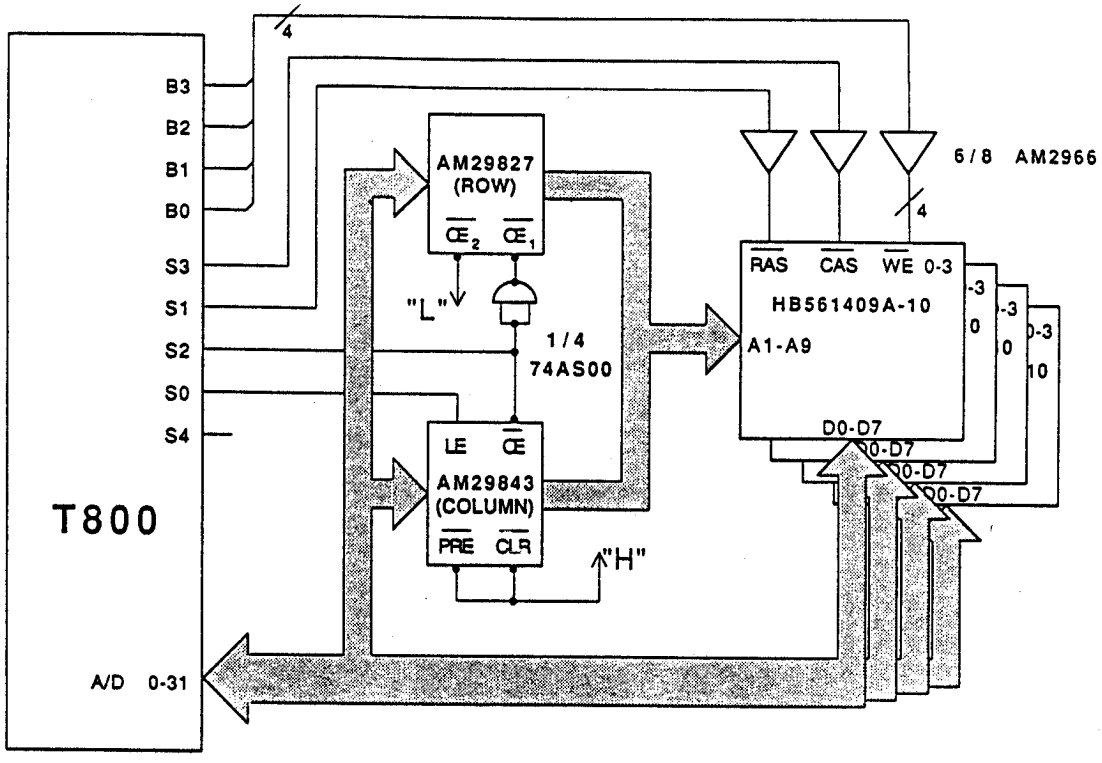
Blockschaltbild eines Rechenknotens

Abb.5



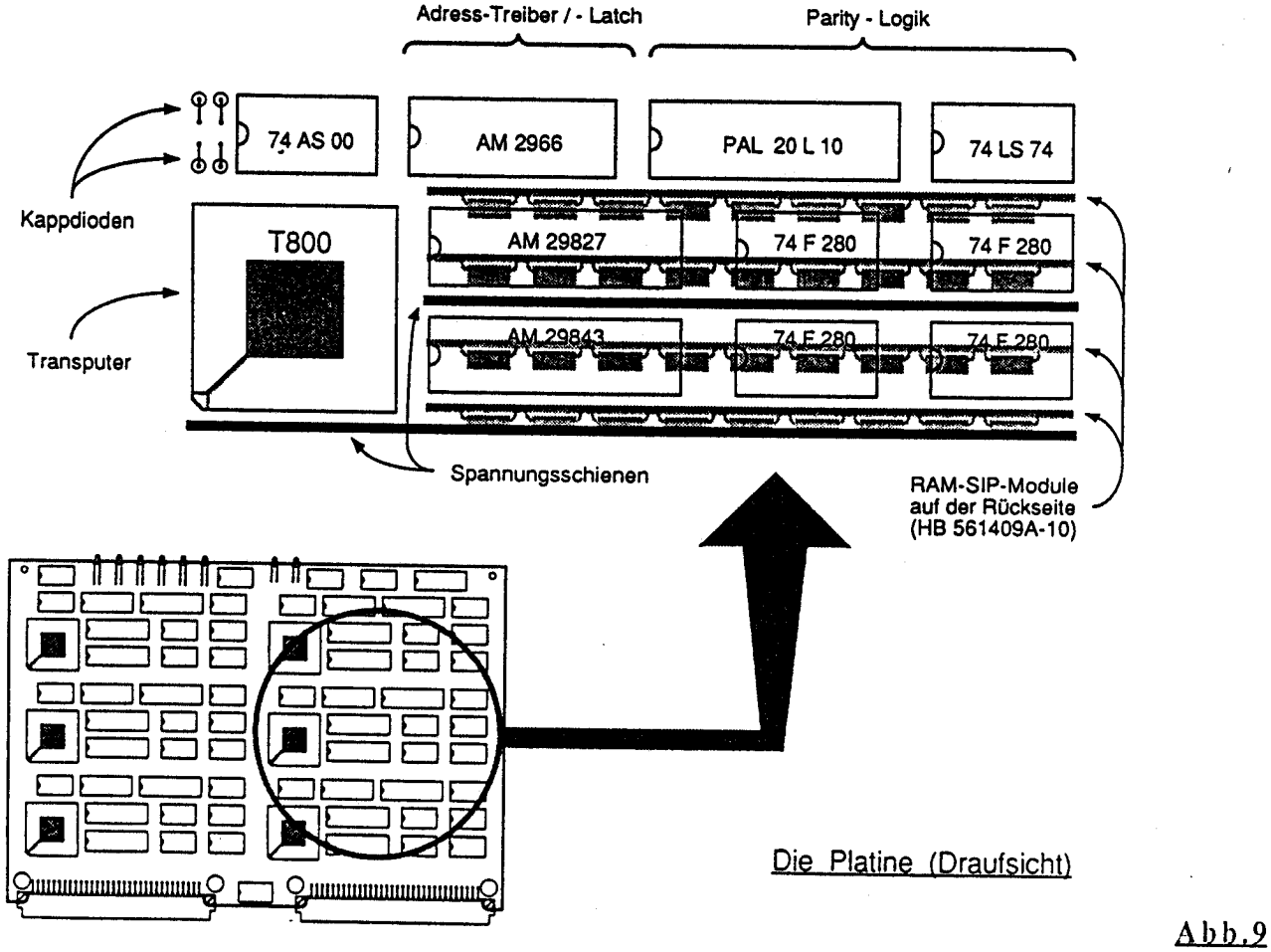
Memory Timing

Abb.6



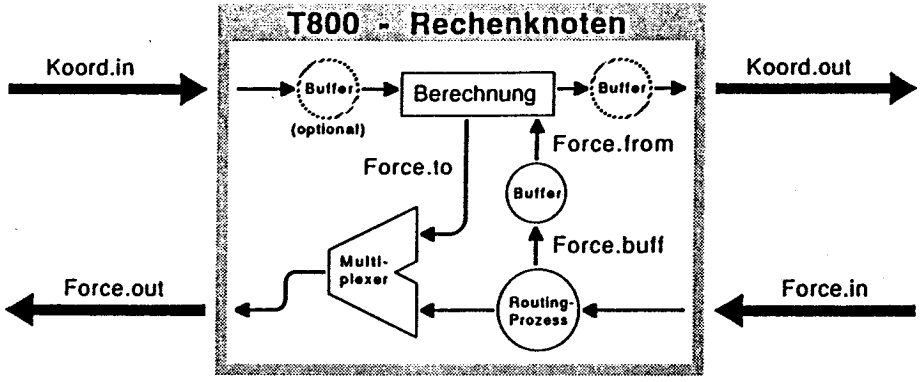
Memory - Ansteuerung

Abb.7



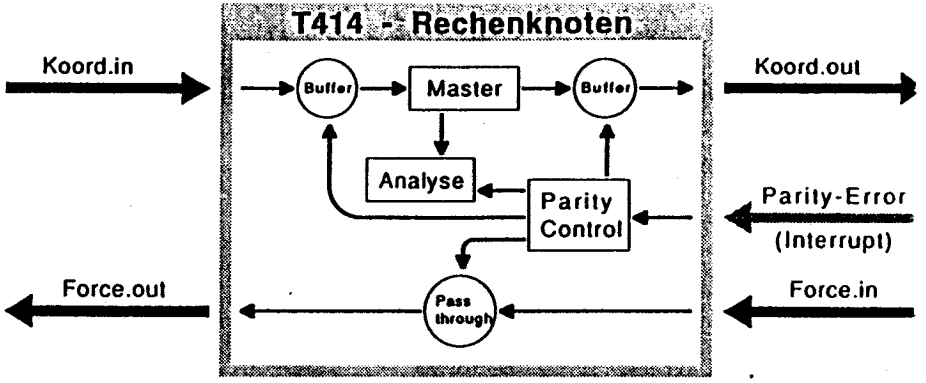
Die Platine (Draufsicht)

Abb.9



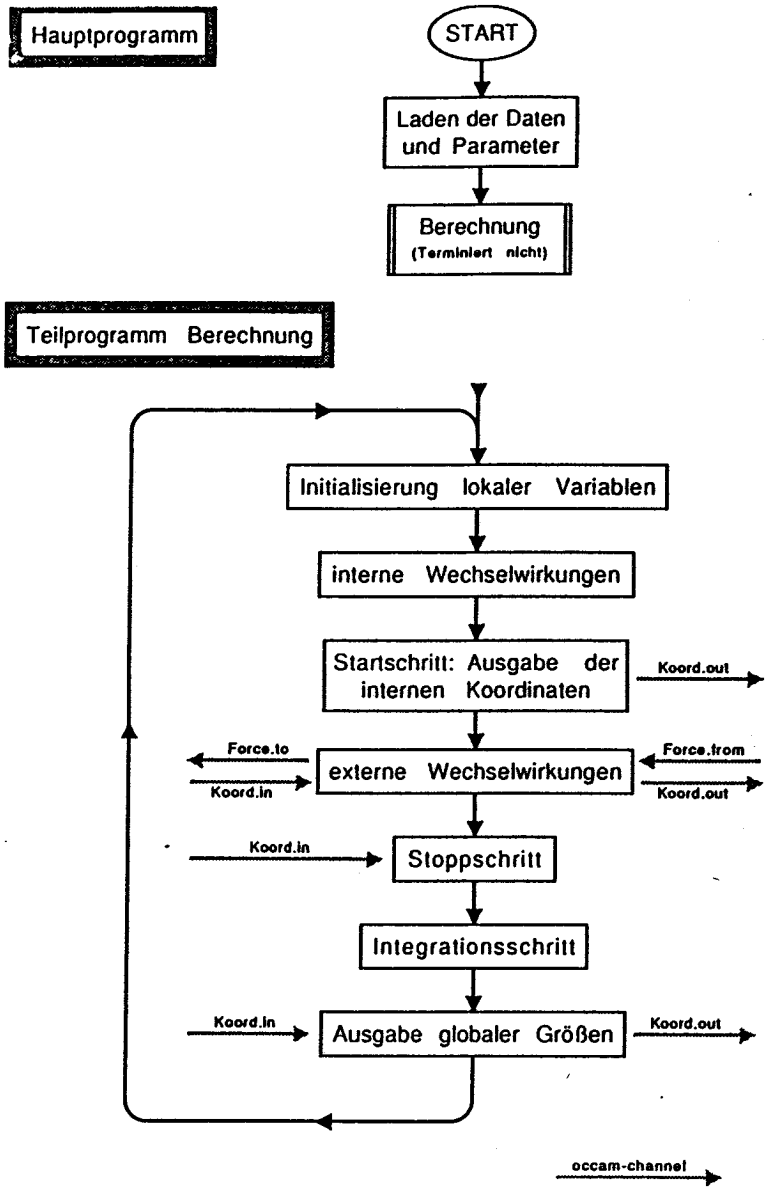
T800 Prozeßstruktur

Abb.11



T414 Prozeßstruktur

Abb.1



Flußdiagramm

Abb.13