mc

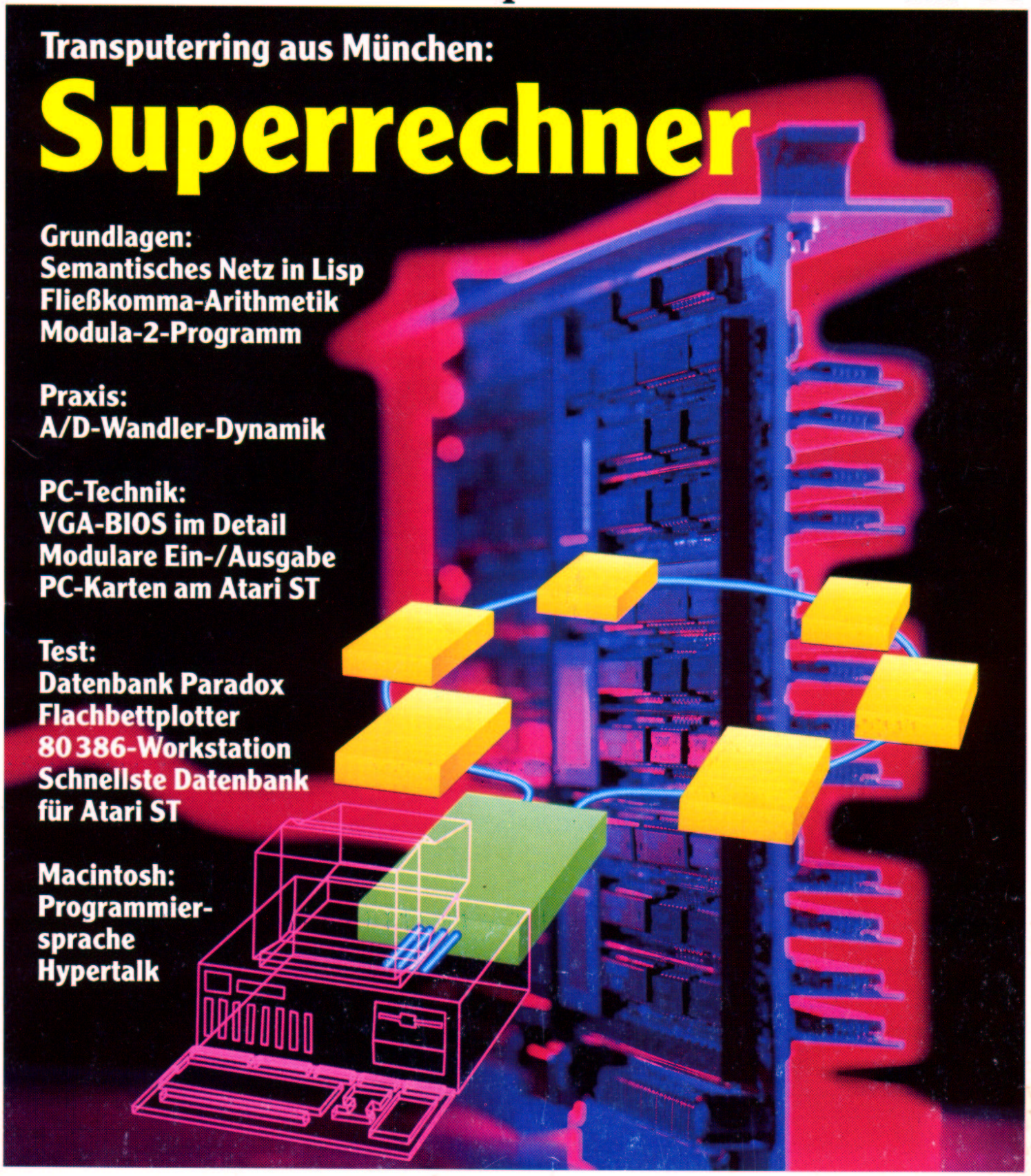
November 1988
8 DM · 70 öS · 8 sFr

Die Mikrocomputer-Zeitschrift

11/88

Transputerring aus München:

Superrechner



Grundlagen:
Semantisches Netz in Lisp
Fließkomma-Arithmetik
Modula-2-Programm

Praxis:
A/D-Wandler-Dynamik

PC-Technik:
VGA-BIOS im Detail
Modulare Ein-/Ausgabe
PC-Karten am Atari ST

Test:
Datenbank Paradox
Flachbettplotter
80386-Workstation
Schnellste Datenbank für Atari ST

Macintosh:
Programmiersprache Hypertalk

EDITORIAL

Liebe Leser!

Diesmal zeigen wir auf dieser Seite wieder junge Mitarbeiter von Professor Schulten – und ihn selbst. Wir meinen, daß sie hier den Vortritt haben sollten, weil sie in diesem Heft wirklich eine Sensation präsentieren. Es geht um die Konstruktion eines preiswerten Supercomputers. Er ist auf der Basis von Transputern konstruiert. Professor Schulten schreibt uns dazu:

„Den in unserem Artikel beschriebenen Parallelrechner haben meine beiden Mitautoren, Helmut Grubmüller (München) und Helmut Heller (Altötting), im Rahmen ihrer Diplomarbeiten in nur neun Monaten entworfen, gebaut, programmiert und zum Einsatz in der biophysikalischen Forschung gebracht: eine geistige und physische Meisterleistung. Angefangen hat alles damit, daß ich die beiden Helmuts ‚entdeckte'. Helmut Grubmüller, er war im dritten Semester, kam eines Tages nach der Vorlesung zu mir und zeigte mir einen aus MC-68000-Prozessoren selbstgebauten Parallelrechner, den er auch selbst programmiert hatte und zum Erstellen von wunderbaren ‚Ray-Tracing'-Graphiken eingesetzt hatte. Helmut Heller hielt einen Vortrag über Fraktale (die berühmten Apfelmännchen!) in meinem Projektseminar ‚Rechnergestützte Physik' – einen Vortrag, den ich nicht vergessen werde: Er hatte auf einem selbstgebauten Graphikrechner, Grundgerüst war ein alter Commodore, einen Computerfilm zum Thema hergestellt, den man nicht besser auf einer Hochleistungsgraphikstation hätte herstellen können. Seinen Rechner hatte er mitgebracht und über den langen Tisch des Vorlesungsraumes waren Platinen, Chips, Drähte und offene Tischrechner verteilt und, nachdem er an einigen Drähten gefummelt hatte, funktionierte alles. Was sollte ich nur mit diesen außerordentlichen Begabungen anfangen? Mit dem Transputerrechner kam mir eine offensichtliche Idee. Was die beiden Helmuts dann mit geringsten Mitteln auf einem 2 m × 3 m kleinen Arbeitsplatz, zum Teil mit selbstgebasteltem Werkzeug und mit einem Mac II als CAD-Station gemacht haben, schildert unser Artikel."

Von links: Schulten, Grubmüller, Heller.

Soweit Professor Schulten. Hier sei noch verraten, daß der Computer in ein 19-Zoll-Rack paßt, äußerst kompakt aufgebaut ist und als Tischgerät durchgehen könnte. In der Leistung steht er einer Cray nicht nach, wenn er voll ausgebaut ist. Und das zu einem hundertstel der Kosten.

Natürlich gibt es in dieser Ausgabe auch ganz normale Beiträge. Zum Beispiel wird ein universelles Interface geschildert, über das man PC-Karten an den Atari ST anschließen kann. Damit steht dem Atari-Fan eine weite Welt preiswerter Karten für den weiteren Ausbau zur Verfügung. Im Beitrag selbst wird konkret geschildert, wie man eine preiswerte 20-MByte-Festplatte anschließt.

Der Beitrag über die dynamische Analyse von AD-Wandlern soll dem Praktiker in der Industrie oder im Labor Kriterien und konkrete Meßhilfen in die Hand geben, um das Verhalten der Wandler besser in den Griff zu bekommen.

Ein Test einer hochwertigen Workstation auf 80386-Basis soll allen PC-orientierten Lesern aufzeigen, was Spitzenmodelle heutzutage bringen müssen. Solche Tests sind Orientierungshilfen hoffentlich auch für den, der schon ein Gerät besitzt und eine Weile damit glücklich sein möchte. Wir glauben nicht, daß man immer den größten und schnellsten und schönsten Computer haben muß. Der richtige sollte es aber sein.

Übrigens, Sie finden uns auf der Elektronika in München (vom 8. 11. 1988 bis 12. 11.) in Halle 18, Stand 18E06. Besuchen Sie uns?

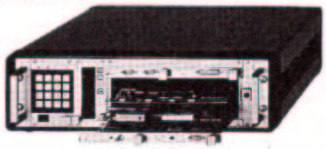
Ihr

Ulrich Rohde

DIE
UNIC-ICB-IDEE

XT/AT-KOMPATIBLE COMPUTER IN INDUSTRIELLEM DESIGN

Die starke Verbreitung von IBM-kompatiblen Rechnersystemen hat es mit sich gebracht, daß eine sehr große Anzahl von entsprechenden Betriebssystemen, Programmiersprachen, Utilities und Programmgeneratoren rasch und preisgünstig erhältlich ist. Die Verwendung dieser Programme für den industriellen Einsatz (Maschinensteuerung, Betriebsdatenerfassung) scheiterte meist daran, daß normale Rechnersysteme für rauhe Umgebungsbedingungen nicht ausgelegt waren (erschütterungsempfindlich, Probleme mit Staub, Temperatur und Feuchtigkeit).



VORTEILE DER UNIC-ICB-RECHNER

100 % softwarekompatibel zu IBM XT/AT-Rechnern. Vollständiger Aufbau auf Doppel- und Einfach-Europakarten – genormte Gehäuse, beliebige Schutzarten. Erweiterter Temperaturbereich durch entsprechende Steckverbindungen und Bauteileauswahl. Erschütterungsunempfindlich durch Einsatz nichtrotierender Massenspeicher (RAM/EPROM-Floppy).
Vollständig modular aufgebautes System, dessen Komponenten für Softwareentwicklung und eventuell Debugging auch am Personalcomputer angeschlossen werden können. Original IBM-Erweiterungskarten (Netzwerkkarten, verschiedene Schnittstellenkarten usw.) sind über eine Adapterkarte direkt auf ICB-Systemen einsetzbar.
Durch die Verwendungsmöglichkeit eines Personalcomputers zur Programmentwicklung für eine industrielle Applikation entfällt die ansonsten notwendige Anschaffung eines eigenen Entwicklungssystems (VME, ECB usw.).
Durch konsequente 19"-Technik (zum Einsatz kommen Platinen im Einfach- und/oder Doppel-Euro-Format) sind ohne großen Aufwand beliebige Schutzarten nach DIN 40 050 realisierbar.

INDUTRONIC®
Dipl.-Ing. Kreiger GmbH & Co. KG
D-8011 Poing/München
Gruberstraße 46
Telefon (0 81 21) 7 01-0
Telefax (0 81 21) 7 13 64

Ich bitte um Zusendung weiterer Informationen:

Name:

Firma:

Straße:

Plz/Ort:

Telefon:

Helmut Grubmüller, Helmut Heller, Klaus Schulten

Eine Cray für „jedermann"

Supercomputer aus München: Eine Cray für 100 000 DM

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.

Es drückt die Autoren 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 Fließkomma-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-Fließkomma-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 MByte/s (full duplex) kommunizieren kann. Auch der neue (Occam II) Compiler kombiniert mit einem sogenannten Folding-Editor (s. u.) ist zufriedenstellend und lebt in idealer Ehe mit der Hardware.

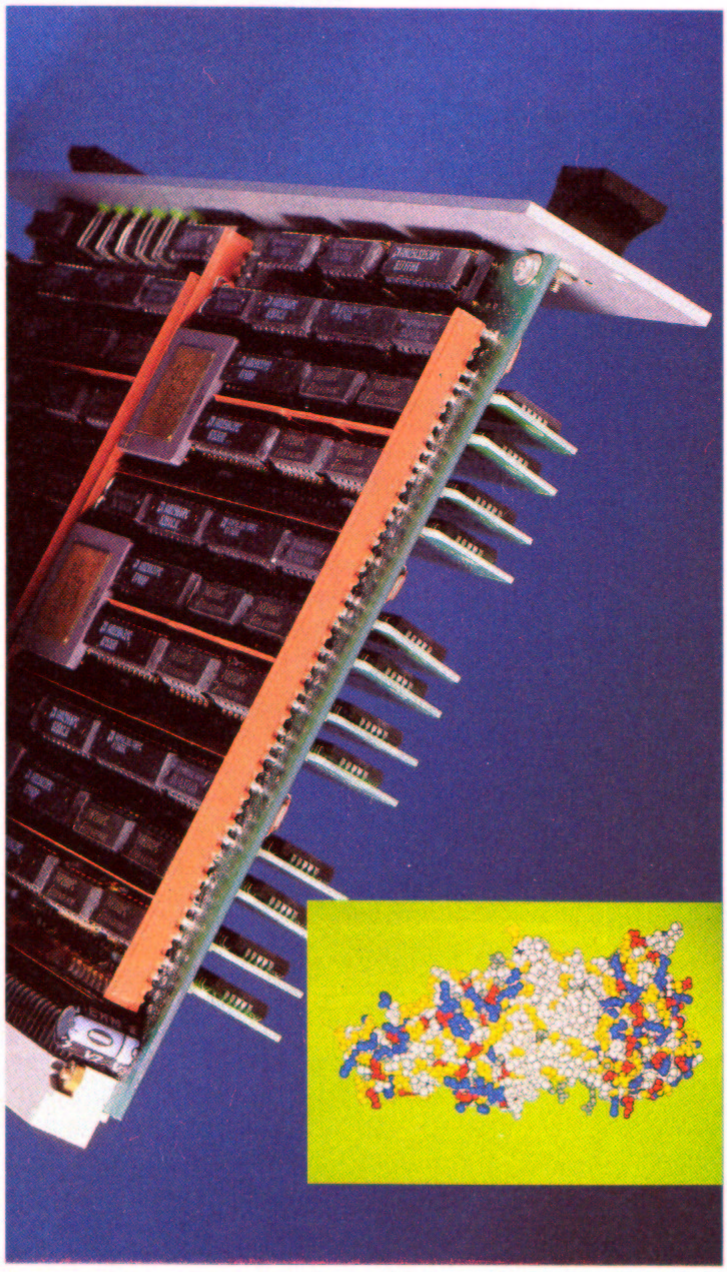


Bild 1. Das ist die Transputerplatine aus der TU München: beidseitig bestückt und drei Lagen. Der Speicher befindet sich auf der Rückseite. Darunter das Reaktionszentrum der Photosynthese, für deren Simulation die Platine arbeiten soll

Die Programmiersprache Occam beruht auf einem von C.A.R. Hoare vorgeschlagenen Modell der Parallelverarbeitung, den sogenannten 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 halten.

Die Grundelemente von Occam sind dabei Prozesse und Kanäle, über letztere kommunizieren

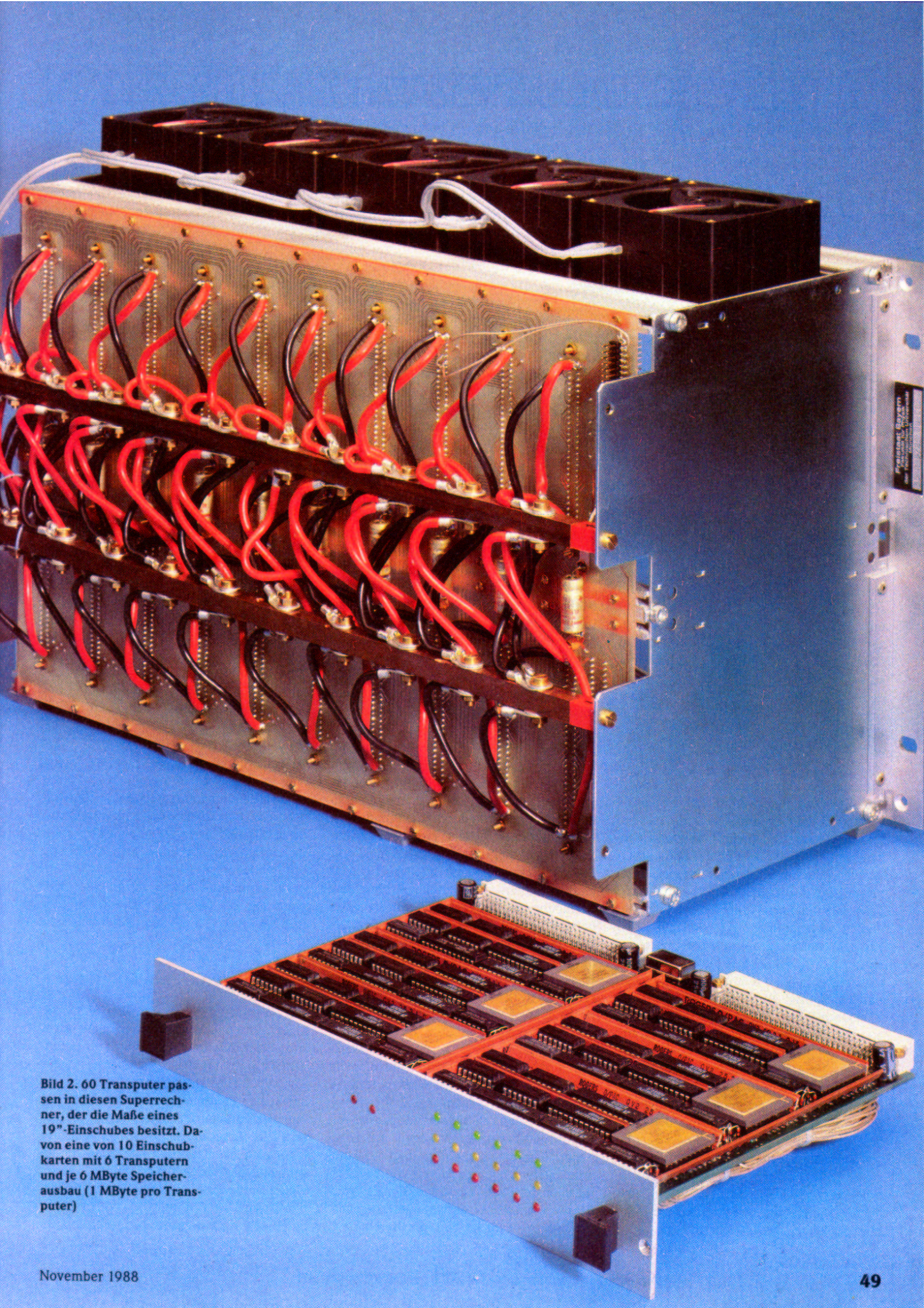


Bild 2. 60 Transputer passen in diesen Superrechner, der die Maße eines 19"-Einschubes besitzt. Davon eine von 10 Einschubkarten mit 6 Transputern und je 6 MByte Speicherausbau (1 MByte pro Transputer)

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 Software-Seite 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 ihrem 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.

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, softwaremäßig während des Programmablaufes die Verknüpfungen zwischen Transputern zu ändern.

Bei einer technisch bedeutende Klasse von Algorithmen, den Rechnungen mit sogenannten 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 im Windkanal 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 Computergrafik. Diesen Weg hat die Technik bereits lange durch die Bereitstellung entsprechender Hardware beschritten. Die Chance liegt jetzt darin, durch parallele Software komplexere grafische 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 Werbebranche.

Biomoleküle im Parallelrechner

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 CPUs 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 bereits in mc dargestellt [1].

Ausgangspunkt für die in diesem Artikel dargestellte Entwicklung und Programmierung eines Parallelrechners war eine Problemstellung, 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 (10^8 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 Engineering, 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. 13 000 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 Makromolekülen üben nämlich alle aufeinander Kräfte aus, die im Sinne der obigen Analyse paraller Prozesse dem absoluten 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 Parallelisierungsstrategien begannen, stießen

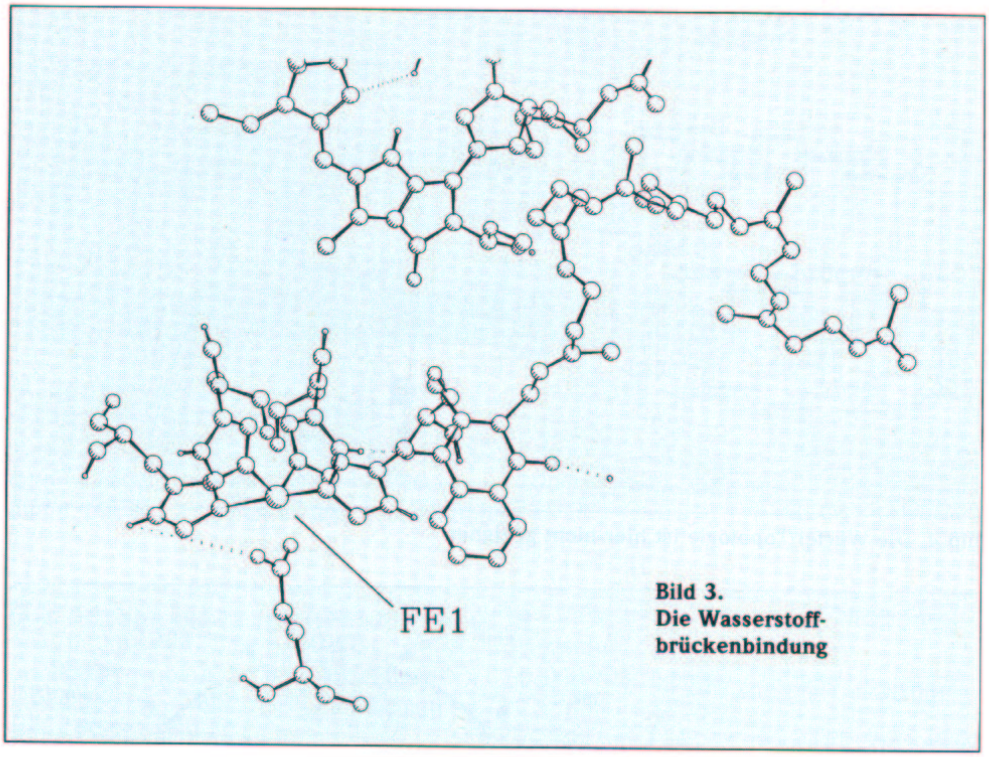


Bild 3. Die Wasserstoffbrückenbindung

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 einleuchtenden Vorschlag. Diesem Vorschlag, der auf einer besonders einfachen Verschaltung der Prozessoren, nämlich einer Ringschaltung, beruht, sind wir nachgegangen, haben sie konkretisiert und den hier beschriebenen Rechner gebaut und programmiert.

Eine wichtige Entscheidung bei unserer Arbeit betraf die Frage, ob wir warten sollten, bis Transputerrechner auf dem Markt erhältlich sind, oder ob wir selber einen Rechner bauen sollten. Wir haben uns für die letztere Möglichkeit entschieden, und zwar im wesentlichen aus zwei Gründen: Erstens benötigten wir den Rechner für unsere Arbeit so schnell wie möglich und wir wußten nicht, inwieweit wir den Lieferversprechen der 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, daß wir die Rechenleistung einer Cray zu einem mindestens hundertmal günstigeren Preis haben konnten und wir sahen hier eine große Chance für die Wissenschaft der Biomoleküle, die wie jede Wissenschaft notorisch unter Geldmangel leidet. Wir wollten also für uns selbst, aber auch für unsere wissenschaftlichen Kollegen 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 wie den unsrigen auch käuflich zu erwerben.

Der numerische Hintergrund

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

$$m_i \ddot{x}_i = -\frac{d}{dx_i} E(x_1, y_1, z_1, x_2, y_2, z_2, \ldots, x_N, y_N, z_N)$$

$$m_i \ddot{y}_i = -\frac{d}{dy_i} E(x_1, y_1, z_1, x_2, y_2, z_2, \ldots, x_N, y_N, z_N)$$

$$m_i \ddot{z}_i = -\frac{d}{dz_i} E(x_1, y_1, z_1, x_2, y_2, z_2, \ldots, x_N, y_N, z_N)$$

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

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

$$E = E_B + E_{El} + E_{rest}$$

Der Energiebeitrag E_B beschreibt die Bindung zwischen den Atomen und resultiert in hochfrequenten Schwingungen der Atome. Der zweite Beitrag beschreibt die elektrostatische Wechselwirkung im Molekül. Es gilt:

$$E_{El} = \sum_{\text{Atompaare } ij} \frac{q_i q_j}{4\pi\varepsilon r_{ij}}$$

wobei r_{ij} den Abstand zwischen Atomen i, j eines Paares darstellt. Entsprechende Ausdrücke existieren für die restlichen Energiebeiträge, wobei zu beachten ist, daß die jeweiligen Energien von der Atomsorte abhängen. Dazu enthält das Dynamikprogramm einen komplexen Datensatz von Konstanten, welche die 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 3* 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.

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 und q_2 sowie ihrem Abstand r:

$$F_1 = \frac{q_1 q_2}{4\pi\varepsilon r^2}$$

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 13 000 Atomen, einer Zahl, die dem photosythetischen Reaktionszentrum (s. o.) entspricht, sind die Kräfte von 84 493 500 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 4* 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 vier Links zur Kommunikation mit anderen Transputern. Da an jeden Würfel sechs weitere angrenzen, mit denen Daten ausgetauscht werden müssen, wären zum Aufbau dieser Topologie sechs 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.

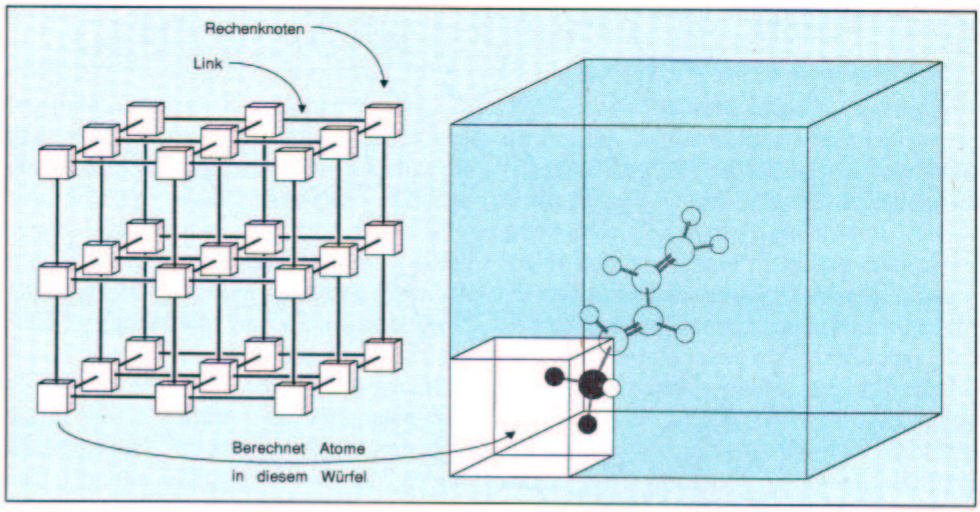


Bild 4. Die Würfel-Topologie ist hier nicht geeignet

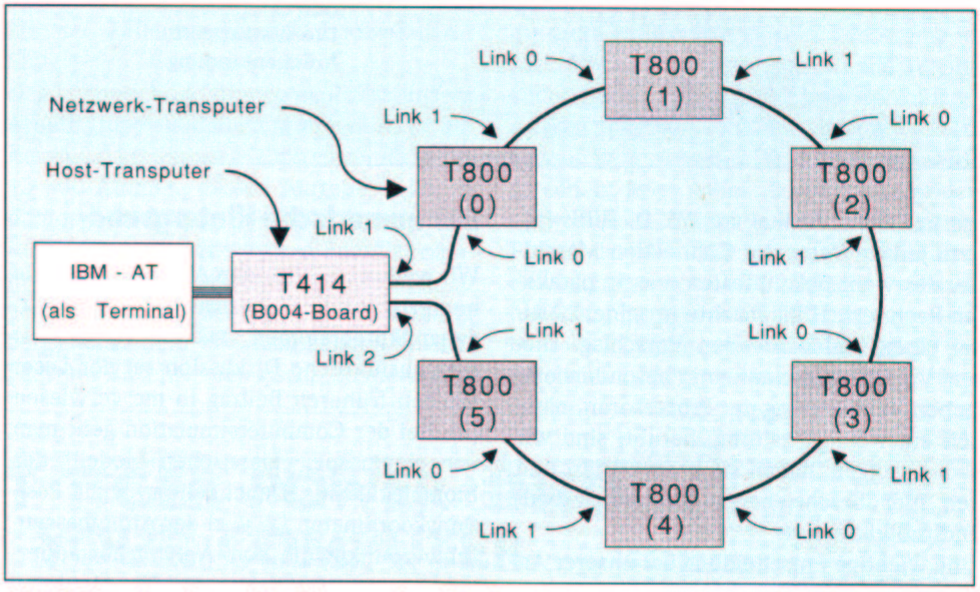


Bild 5. Eine Anordnung im Ring erweist sich als sehr brauchbare Topologie

Diese Probleme vermeidet eine andere Topologie, die in *Bild 5* 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.

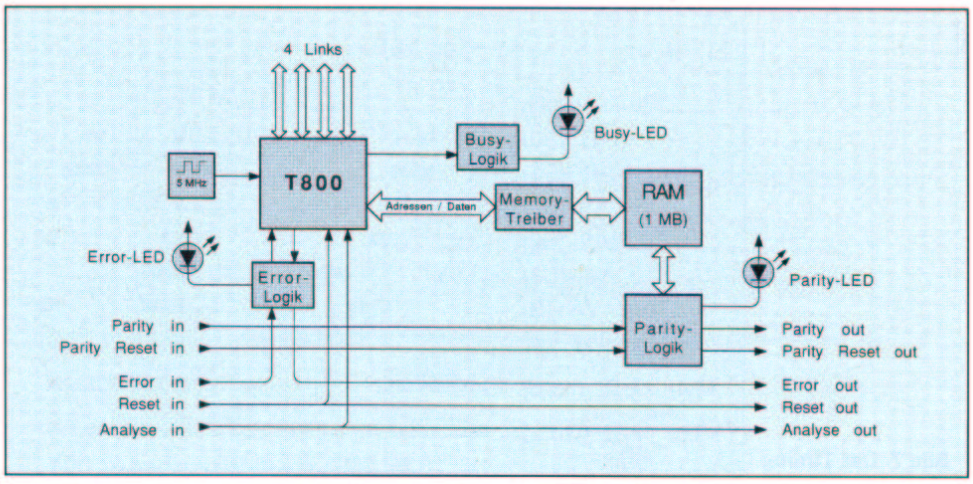


Bild 6. Das Blockschaltbild eines Rechenknotens

Der Aufbau eines Rechenknotens

Ein Rechenknoten besteht aus folgenden Komponenten, die im Blockschaltbild *(Bild 6)* zu erkennen sind:

- Transputer IMST800G20S (32 Bit Daten- und Adreßbus, 20 MHz Taktfrequenz)
- 1 MByte dynamisches RAM als Lokalspeicher für Programm und Daten, aufgebaut aus vier SIP-Modulen mit je neun 256 kBit × 1 Chips (100 ns)
- Einem byte-weisen Parity-check-Mechanismus
- 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 MByte wurde durch verschiedene Faktoren bestimmt: zum einen waren zur Zeit der Entwicklung auf dem RAM-Markt nur 256-KBit-Bausteine zu erhalten, so daß 4 MByte 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-Zoll-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 bis 1000 FIT (**F**ailure **i**n **T**ime, 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

GRAPHTEC

A3-Plotter Serie MP 3000

Komfortable und preisgünstige Plotter-Serie mit 8 Federn, hohem Bedien-Komfort und optimaler Zeichenqualität

- Plottfläche 404×285 mm
- Selbstabdeckende Stiftstation
- Einstellbarer Federandruck
- Mech. Auflösung 0,025 mm
- Programm. Auflösung 0,1/0,025 mm
- Elektrostatische Papierhalterung
- Bedienungsfeld mit LCD-Display
- Plottgeschwindigkeit 400 mm/s
- Schnittstellen: RS-232-C und Centronics (Standard)
- Option: IEC-Bus
- GP-GL / HP-GL* Emulation umschaltbar

NEU! Bleistift-Version

* HP-GL ist eingetragenes Warenzeichen der Hewlett-Packard

GRAPHTEC RICHTUNGSWEISEND

Wir stellen aus: SYSTEC 88, München, Halle 7, Stand 7 A 20
electronica 88, München, Halle 20, Stand 20 C 02

GRAPHTEC GmbH · Postfach 1132 · D-8031 Seefeld (Obb.) 1 · (08152) 7912-0
(W)-Berlin (030) 8835063 · Hamburg (040) 669139 · Düsseldorf (0211) 721145 · Karlsruhe (07243) 90085 · Wien (0222) 973397

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 sieben 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 Flipflop 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. Darüber hinaus 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 7* und *8* erklärt, folgendermaßen: Mit der steigenden Taktflanke von **T1** gibt der Transputer die Adressen aus. Aufgrund der Gatterlaufzeit des Buffers

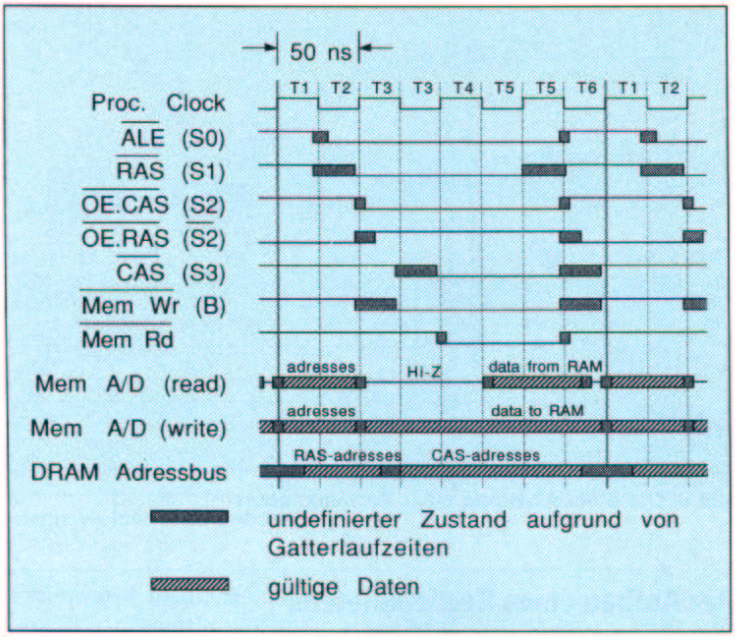


Bild 7. Das Timing auf der Platine zwischen CPU und Speicher

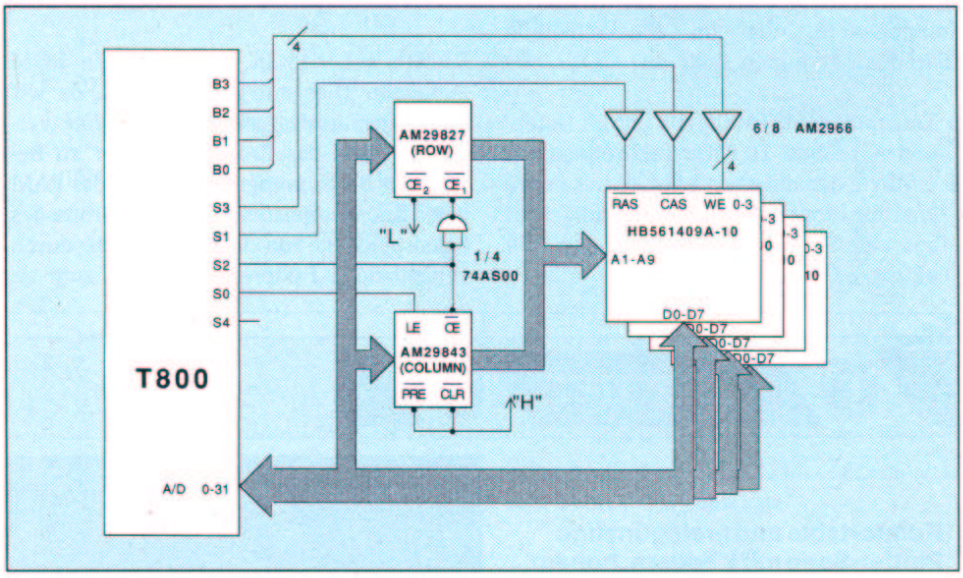


Bild 8. Die Ansteuerung der Speicherbausteine

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 ins 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, so daß mit der steigenden Flanke von **T6** der Buszyklus abgeschlossen ist.

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 *Bild 9* und *2*). 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 Trans-

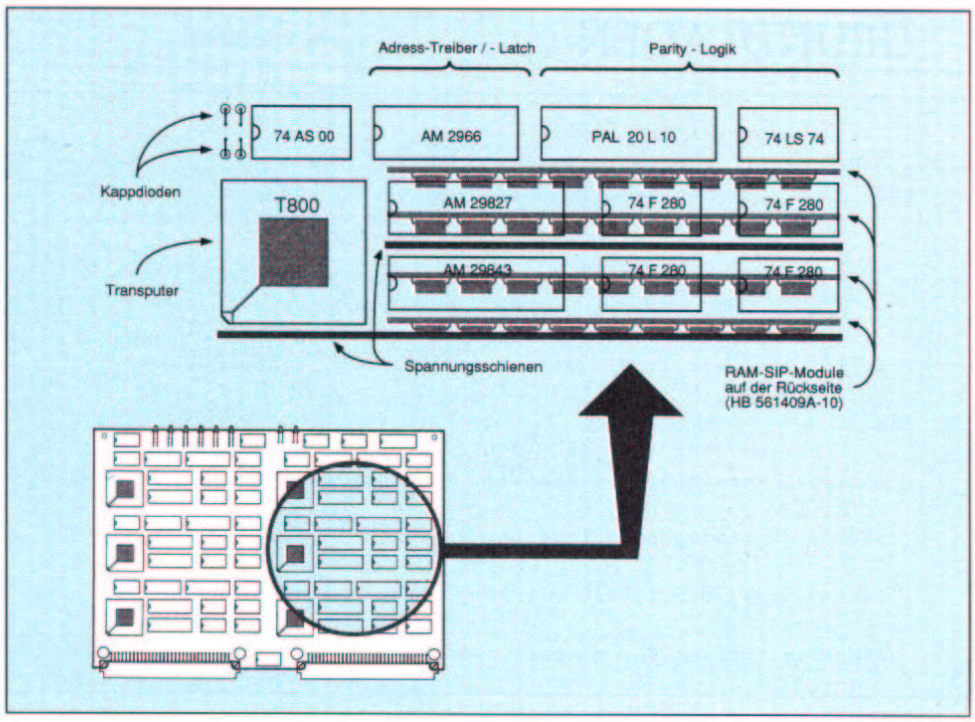


Bild 9. Schematische Zeichnung einer Platine mit vergrößerter Darstellung eines Rechenknotens

puter, 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 Platinen herauszuführen.

Zehn Platinen in einem 19-Zoll-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 Mikrosekunden 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 eine 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-Zoll-Einschub; das bedeutet, daß 10 Platinen oder 60 (!) Transputer in diesem Einschub Platz finden. Die verbleibenden zwei Tiefeneinheiten werden für Resettaster und Betriebsanzeige genutzt.

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 (**P**rotein **D**ata **B**ank), 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

(**T**ransputer **D**evelopment **S**ystem). 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 *EXE*s 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 *PROGRAM*s haben.

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 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 Folding-Editor 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.

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.

Die Programme im T800-Rechenknoten

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

- 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.

Die Programme im T414-Rechenknoten

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 (Modul Controller):

- Ein *Parity-Prozeß* überwacht den Interrupteingang des T414, der mit dem Ausgang parity error 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 die 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 Aus-

```
... Vereinbarungen (NO LIST)
PRI PAR
  PAR
    Pass.forces (scr.in[4], to.net.in, from.net.out, stopper[2])

    Buffer.in (scr.in[0], from.net.in, input, stopper[0])

    ... Master (scr.in[1], to.Analyse) (NO LIST)
    Buffer.out (scr.in[2], output, to.net.out, stopper[1])

  PAR
    ... Buffer fuer Analyse (NO LIST)
    Analyse (scr.in[3], to.Ana[11], P.error, picture, box.x, box.y)

    scrstream.multiplexor (scr.in, screen, stopper[3])

    Parity (key, scr.in[5], P.error, stopp)

    stop.mux (scr.in[6], stopper, stopp)
```

Bild 10. Die Übersicht über das Programm in der Grobstruktur mit geschlossenen Folds

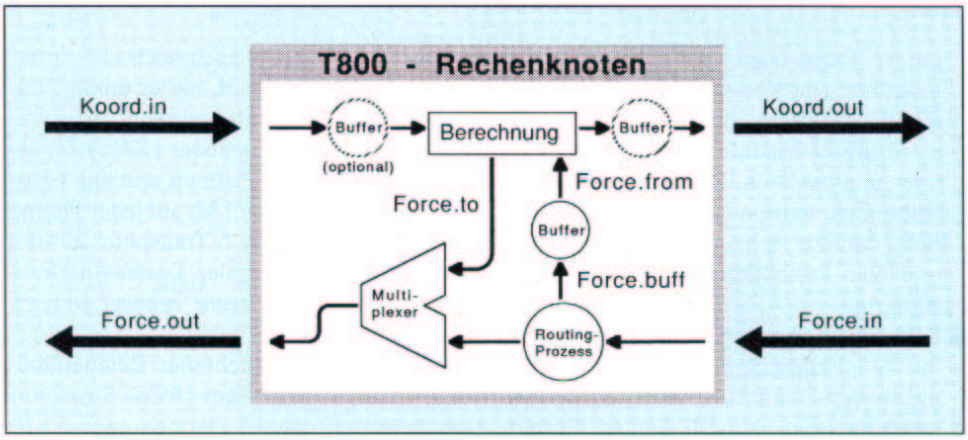


Bild 11. Die T800 Prozeßstruktur

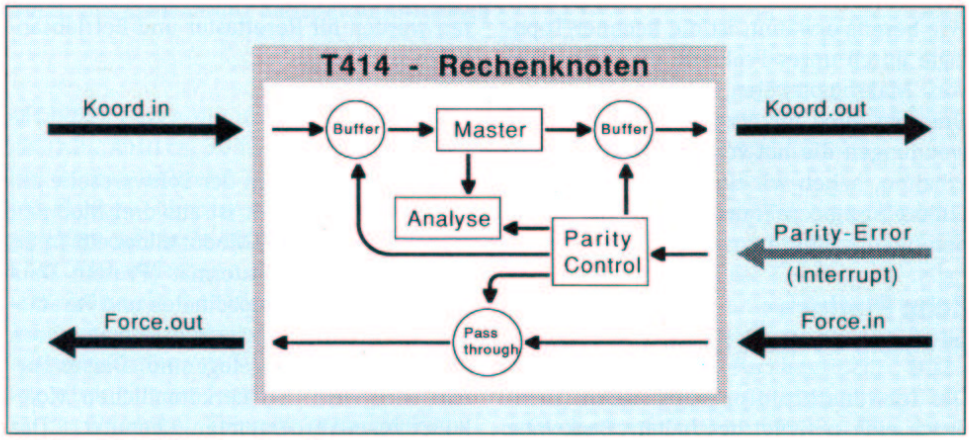


Bild 12. Die T414 Prozeßstruktur

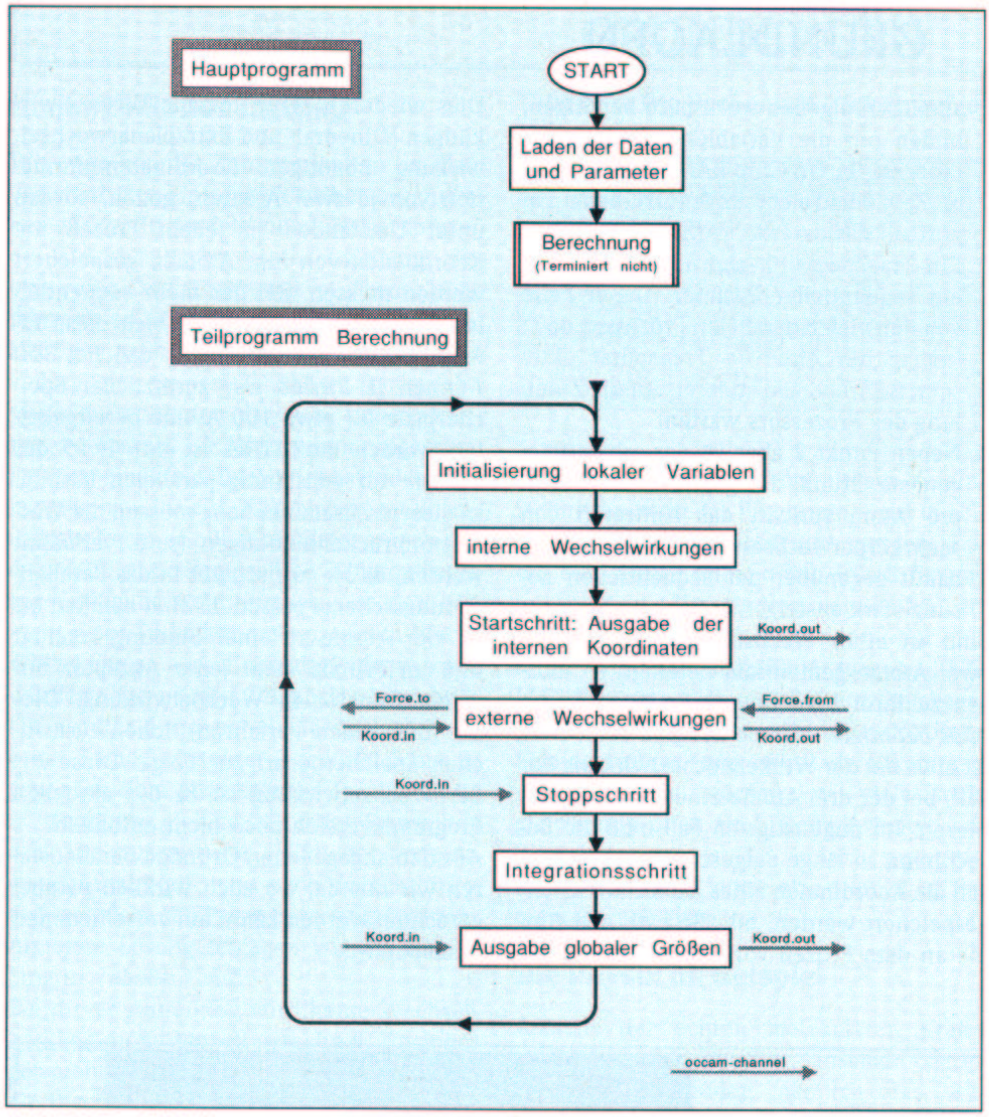


Bild 13. Das Flußdiagramm

gang kurz, denn Kraftdaten werden hier nicht benötigt.

- Die zwei *Buffer-Prozesse* erfüllen wichtige Aufgaben: Im schlimmsten Fall ereignet sich ein Speicherfehler gerade in dem T800, der dem T414 direkt vorgeschaltet ist. Und zwar gerade dann, wenn dessen DMA-Maschine soeben über das Link Werte an den T414 überträgt. Da die Nachricht des Parity-Fehlers den Master oder gar die Analyse erst nach einer gewissen Verzögerung erreicht, könnte es ohne Buffer schon zu spät sein: fehlerhafte Daten stehen im 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 benachbarten Prozessoren so korrumpieren würde, daß dieser mitten im Datentransfer über ein Link seinen Dienst einstellt. Um dann den berüchtigten Deadlock zu verhindern, „opfern" sich die Pufferprozesse: sie können unter Umständen festhängen, aber es ist sichergestellt, daß Master und Analyse 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 die 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 ab und hält die Restartfiles auf dem neuesten Stand.

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

Das 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 der zu ihm lokalen Atome, sowie Informationen über ihre Bindungen und die zu betrachtenden Winkel. Anschließend werden die globalen Daten, das sind Kraftkonstanten und Parameter der verschiedenen Wechselwirkungspotentiale, an alle Transputer verteilt. Ein spezielles Umschaltsignal steuert den Übergang in die eigentliche Berechnungsphase. Dieser Programmteil terminiert nicht, doch kann der Hosttransputer (T414) ein Resetsignal auslösen, welches das 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 Initialisierung der lokalen Variablen (Zeiger in diverse Listen, Kraftakkumulatoren, etc.) werden zuerst alle Wechselwirkungen der Atome ausgerechnet, die zum jeweiligen Transputer lokal sind. Das EXE *Controller* hat während der Ladephase dafür gesorgt, daß alle (internen) Listen gerade die richtigen Einträge besitzen, die zu den lokalen Atomen passen. Dieser Teil unterscheidet sich kaum von den herkömmlichen, sequentiellen Dynamikprogrammen. Der folgende *Startschritt* ist neu: der Transputer reicht „seine" Koordinaten an seinen rechten Nachbarn weiter. Wie im Kapitel Topologie erläutert, werden zur Berechnung der externen Wechselwirkungen nun die Koordinaten vom linken Nachbarn verwendet. Die Elektrostatikberechnung etwa ist nun einfach eine Doppelschleife über alle internen und alle externen Atome. Die hier berechneten Kräfte werden zum einen auf die lokalen Kraftakkumulatoren addiert, zum anderen werden sie mit umgekehrtem Vorzeichen an den linken Nachbarn zurückgegeben, so daß dieser sie ebenfalls in seine Kraftakkumulatoren addieren kann. Dann werden die *fremden* Koordinaten an den rechten Nachbarn weitergegeben, und das Spiel wiederholt sich. Bei 12 Transputern erreichen so nach 12 Schritten jeden Transputer wieder die eigenen Koordinaten. Diese werden aber nicht gebraucht und können daher gelöscht werden (Stoppschritt); ohne diesen Stoppschritt wäre der Ring im nächsten Zyklus nicht bereit, den Startschritt auszuführen: jegliche Aktivität käme zum Erliegen (sogenannter Deadlock). Da nun die Gesamtkraft auf jedes interne Atom bekannt ist, können in einem Integrationsschritt (nach *Verlet*, siehe [9]) die neuen Positionen der Atome 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 Gesamtenergie Ungenauigkeiten der Berechnung, da in dem berechneten System die Gesamtenergie eigentlich konstant bleiben müßte.

Sequentiell oder parallel

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 KByte 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 sequentiellen 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, so daß 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, so daß bei 300 Atomen je Prozeß nur zusätzlicher Speicherplatz für etwa 100 Atome bereitgehalten werden muß. Dies ist nur 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!

THE QUALITY YOU CAN BANK ON

MODEM

DATATRONICS

BELL 103/212A
CCITT V21/V22/V23/V22bis
1200, 2400, 9600 BPS
AUTO-DIAL, AUTO-ANSWER
HAYES™ COMPATIBLE
MNP ERROR CORRECTION
ACOUSTIC COUPLER

DATATRONICS TECHNOLOGY, INC.
172 Nan-King E. Rd., Sec. 5, Taipei 10572, Taiwan, R.O.C.
Telex: 28440 DTXMODEM Fax: 886-2-764-0614
Tel: (02) 762-3202 (3 lines); 768-6262 (3 lines)

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 5 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 nach dem physikalische Prinzip *Actio = Reactio* das gleiche ist). Jeder Transputer darf also nur mit der Hälfte der anderen „wechselwirken", und zwar so geschickt, daß trotzdem alle Paare berechnet werden. Als erste Lösung schlagen wir die Aufteilung in *Tabelle 1* vor.

Bei dieser Aufteilung wird keine Wechselwirkung mehr doppelt gerechnet. Doch bei genauerem Hinsehen entdecken wir, daß die Wechselwirkung zwischen den Atomen des Transputers 0 und denen von Transputer 3 fehlt. Die richtige Aufteilung ist in *Tabelle 2* angegeben. Leider muß man bei dieser Aufteilung in Kauf nehmen, daß während eines Zeitschrittes nur die Hälfte der Transputer rechnen. Bei einer *ungeraden* Zahl von Transputern läßt sich eine bessere Aufteilung erreichen. Allgemein gilt für große Transputerzahlen, daß $2N$ Transputer dieselbe Rechenleistung erbringen, wie $(2N-1)$ Transputer.

Tabelle 1: Die Rechenlast – falsch aufgeteilt

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 2: Die richtige Aufteilung der Rechenlast

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

Do it Yourself – ein vereinfachtes Beispielprogramm

Um die wesentlichen Aspekte unseres Molekulardynamikprogrammes zu demonstrieren, haben wir das Beispielprogramm aus dem früheren Beitrag zum Thema „Moleküle im Computer" [2, Seite 54f] in Occam II umgeschrieben und parallelisiert. Das Programm ist in *Bild 14* in seiner Struktur aufgelistet. Das Listing illustriert einerseits die diskutierte Parallelisierungsstrategie, andererseits gibt es dem Leser ein Gefühl für die Sprache Occam II. Wer selbst über einen oder mehrere Transputer sowie einen Occam II-Compiler verfügt, muß das 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. Daher sind einige folds, deren Inhalt für das unmittelbare Verständnis verzichtbar erscheint, nicht geöffnet, sondern nur mit ihrer Titelzeile aufgeführt (durch „(NO LIST)" gekennzeichnet). Bei der Druckausgabe kann die sonst sehr übersichtliche Foldingstruktur nur sehr unvollkommen wiedergegeben werden, da viele folds gleichzeitig offen sind. Beim Arbeiten am Bildschirm aber ist das anders; so wurde dieses Programm ohne einen einzigen Papierausdruck erstellt.

Die Physik im Beispiel

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

Das parallele Programm

Auch dieses Beispielprogramm hat dieselbe Grundstruktur wie das echte Programm – kein Wunder, es ist 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 nach einigen Werten; 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. *Nicht aufgelistet, siehe [10].* 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 Programm-Modul, 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 artglei-

chen 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

```
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
```

Bild 15. Deadlock blockiert alles

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 (*Bild 15*):

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 zu 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.

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 Anwendung erreicht wird, kann von den Herstellerangaben stark (nach unten) abweichen. Oft bringt ein guter Compiler auf einer schwachen Maschine mehr, als eine gute Maschine mit einem mäßigen Compiler bieten kann (siehe hierzu auch Tabelle 4 und vergleiche C-Compiler und FORTRAN-Compiler (XPLOR)). Es ist also bei Benchmarkwerten stets Vorsicht geboten. Wirklich aussagekräftige Werte über die Gespanne Hard- und Software erhält man nur, wenn man die Ausführungszeiten des Anwendungsprogrammes vergleicht. Dies gilt umso mehr, wenn man die Leistung sequentieller Rechner oder die von Vektorrechnern mit der eines parallelen (MIMD, **M**ultiple **I**nstruction **M**ultiple **D**ata) Rechners vergleichen will: ein Problem läßt sich eben gut parallelisieren, ein anderes nicht. Aus diesem Grund erscheinen in der folgenden Aufstellung *Tabelle 3* die Laufzeitangaben für echte Molekulardynamikrechnungen, wobei die entscheidenden Vergleichsprogramme für die jeweiligen Computer optimiert wurden. An dieser Tabelle sind einige Eigenschaften sehr schön zu erkennen:

1. Die Rechenzeit steigt nahezu quadratisch mit der Zahl der Atome.
2. Die Rechenzeit für die Elektrostatik- und vdW-Berechnungen beträgt zwischen 50 % und 99 % der gesamten Rechenzeit.

```
--*****************************************************************
--*          COPYRIGHT by Helmut HELLER     (Tx-19880818.)        *
--*****************************************************************

{{{  SC Dynamik (Koord.in, Koord.out, Force.in, Force.out, nr)
PROC Dynamik (CHAN OF MOL.KOORD Koord.in, Koord.out,
              CHAN OF MOL.FORCE Force.in, Force.out, VAL INT my.number)
  ...  Globale Daten (NO LIST)
  SEQ
    {{{  Loader
    BOOL loading:
    SEQ
      {{{  calculate
      IF
        (Nr.of.T800.in.Net REM 2) = 0   -- gerade Transputerzahl: schlecht!
          IF
            my.number < (Nr.of.T800.in.Net/2)
              SEQ
                calculate := (Nr.of.T800.in.Net/2)+1
                add:= calculate-1
            TRUE
              SEQ
                calculate := Nr.of.T800.in.Net/2
                add:= calculate+1
        TRUE                              -- ungerade Transputerzahl: gut!
          SEQ
            calculate := (Nr.of.T800.in.Net+1)/2
            add := calculate
      }}}
      {{{  load
      SEQ
        Koord.in ? CASE loading.coordinates; Nr.of.Atoms.loc::x1
        Koord.in ? CASE loading.coordinates; Nr.of.Atoms.loc::y1
        Koord.in ? CASE loading.coordinates; Nr.of.Atoms.loc::x2
        Koord.in ? CASE loading.coordinates; Nr.of.Atoms.loc::y2
        Koord.in ? CASE loading.coordinates; Nr.of.Atoms.loc::atom.mass
        Koord.in ? CASE loading.coordinates; Nr.of.Atoms.loc::atom.hardness
        Koord.in ? CASE loading.coordinates; Nr.of.Atoms.loc::atom.radius
      loading := TRUE --zuerst wird geladen
      WHILE loading --Durchreichen der Koordinaten und laden der globalen Werte
        Koord.in ? CASE
          [Max.Atoms.per.T800]REAL32 temp:
          INT temp.nr:
          loading.coordinates; temp.nr::temp
            Koord.out ! loading.coordinates; temp.nr::[temp FROM 0 FOR INT temp.nr]
          loading.constants;Ea;Eb; box.x;box.y
            Koord.out ! loading.constants;Ea;Eb; box.x;box.y
          loading.all; half.rez.time.step
            SEQ
              loading := FALSE
              Koord.out ! loading.all; half.rez.time.step
      square.time.step := half.rez.time.step*half.rez.time.step
      half.rez.time.step:=0.5(REAL32)/half.rez.time.step
      }}}
    }}}

    {{{  Berechnung
    CHAN OF ANY Force.from, Force.to, Force.inter, Force.buff:
    CHAN OF MOL.KOORD Koord.inter:
    PRI PAR
      PAR
        {{{  buffer process (Fuer Addition)
        [Max.Atoms.per.T800]REAL32 extern.Fx, extern.Fy:
        INT Nr.of.Atoms.ext:
        WHILE TRUE
          SEQ
            Force.buff ? Nr.of.Atoms.ext::extern.Fx;
                         Nr.of.Atoms.ext::extern.Fy
            Force.from ! Nr.of.Atoms.ext::[extern.Fx FROM 0 FOR Nr.of.Atoms.ext];
                         Nr.of.Atoms.ext::[extern.Fy FROM 0 FOR Nr.of.Atoms.ext]
        }}}
        {{{  routing process
        WHILE TRUE
          [Max.Atoms.per.T800]REAL32 extern.Fx, extern.Fy:
          INT ext.number, Nr.of.Atoms.ext:
          SEQ
            Force.in ? ext.number;
                       Nr.of.Atoms.ext::extern.Fx;
                       Nr.of.Atoms.ext::extern.Fy
            IF
              ext.number = my.number  -- Kraefte fuer mich!
                Force.buff ! Nr.of.Atoms.ext::[extern.Fx FROM 0 FOR Nr.of.Atoms.ext];
                             Nr.of.Atoms.ext::[extern.Fy FROM 0 FOR Nr.of.Atoms.ext]
              TRUE
                Force.inter ! ext.number;
                              Nr.of.Atoms.ext::[extern.Fx FROM 0 FOR Nr.of.Atoms.ext];
                              Nr.of.Atoms.ext::[extern.Fy FROM 0 FOR Nr.of.Atoms.ext]
        }}}
        {{{  force multiplexer
        WHILE TRUE
          [Max.Atoms.per.T800]REAL32 extern.Fx, extern.Fy:
          INT ext.number, Nr.of.Atoms.ext:
          SEQ
            ALT
              Force.inter ? ext.number;
                            Nr.of.Atoms.ext::extern.Fx; Nr.of.Atoms.ext::extern.Fy
                SKIP
              Force.to  ?   ext.number;
                            Nr.of.Atoms.ext::extern.Fx; Nr.of.Atoms.ext::extern.Fy
                SKIP
            Force.out  !  ext.number;
                          Nr.of.Atoms.ext::[extern.Fx FROM 0 FOR Nr.of.Atoms.ext];
                          Nr.of.Atoms.ext::[extern.Fy FROM 0 FOR Nr.of.Atoms.ext]
        }}}
      PAR
        {{{  coordinates
        [Max.Atoms.per.T800]REAL32 Fx, Fy: --Kraftakkumulatoren
        REAL32 pot.energy, kin.energy:     --Energieakkumulatoren
        WHILE TRUE
          {{{  FUNCTION f
          REAL32, REAL32 FUNCTION f (VAL REAL32 x, box, hardness)
            REAL32 F, dF:
            VALOF
              VAL ABSx IS ABS(x): --zur Beschleunigung abgekuerzt.
              IF
                ABSx > (box+hardness) --ausserhalb der Box
                  F, dF := ZERO, ZERO --kraeftefrei
                ABSx < (box-hardness) --innerhalb der Box
                  F, dF := ONE, ZERO  --kraeftefrei, aber Gewinn potentieller Energie
                TRUE --im Randbereich: Zusammenstoss
                REAL32 t:
                SEQ
                  t:=(ABSx - box)/hardness
                  F:=0.25(REAL32)*(TWO-(t*(3.0(REAL32)-(t*t))))
                  dF:=0.75(REAL32)*(((t*t)-ONE)/hardness)
                  IF
                    x < ZERO
                      dF := -dF
                    TRUE
                      SKIP
            RESULT F, dF
          :
        }}}
```

Bild 14. Das Listing des Occam II-Beispielprogrammes – nicht alle Folds sind geöffnet

```
SEQ
  {{{  Lokale WW und Initialisierung der Akkumulatoren
  pot.energy, kin.energy:=ZERO, ZERO
  SEQ i=0 FOR Nr.of.Atoms.loc
    ...  Abkuerzungen fuer Atom i (NO LIST)
    SEQ
      REAL32 fx, fy, dfx, dfy:
      SEQ -- WW mit Waenden
        fx, dfx := f(x, box.x, box.hardness)
        fy, dfy := f(y, box.y, box.hardness)
        pot.energy := pot.energy - (Eb*(fx*fy))
        fx.i, fy.i := Eb*(dfx*fy), Eb*(fx*dfy) --Kraft auf Atom i
      SEQ j=0 FOR i -- WW mit Atomen
        ...  Variablen, Abkuerzungen fuer Atom j (NO LIST)
        SEQ
          dx, dy := x - x2[j], y - y2[j]
          Rij := SQRT((dx*dx)+(dy*dy)) -- Abstand der Atommittelpunkte
          p, df :=  f(Rij, (r+atom.radius[j]),(hard+atom.hardness[j]))
          pot.energy := pot.energy + (p*Ea)
          fi := Ea*(df/Rij)
          fx, fy := fi*dx, fi*dy
          fx.i, fy.i := fx.i - fx, fy.i - fy  --  ACTIO  und
          fx.j, fy.j := fx.j + fx, fy.j + fy  --  REACTIO!
  }}}
  {{{  Startschritt: Ausgabe der lokalen Koordinaten
  Koord.out ! coordinate;
          Nr.of.Atoms.loc::[x2 FROM 0 FOR Nr.of.Atoms.loc];
          Nr.of.Atoms.loc::[y2 FROM 0 FOR Nr.of.Atoms.loc];
          Nr.of.Atoms.loc::[atom.hardness FROM 0 FOR Nr.of.Atoms.loc];
          Nr.of.Atoms.loc::[atom.radius FROM 0 FOR Nr.of.Atoms.loc]
  }}}
  {{{  Behandlung der externen Atome und Stoppschritt
  ...  Variablen (NO LIST)
  SEQ
    add.count:=1 --zaehlt wie oft schon fremde Kraefte addiert wurden.
    {{{  externe Berechnungen
    SEQ index=1 FOR (Nr.of.T800.in.Net-1)
      SEQ
        Koord.inter ? CASE coordinate;
           Nr.of.Atoms.ext::extern.x;
           Nr.of.Atoms.ext::extern.y;
           Nr.cf.Atoms.ext::extern.hardness;
           Nr.of.Atoms.ext::extern.radius  -- Input extern
        IF
          index < calculate
            {{{  Kraftberechnung mit externen Atomen
            INT ext.number:
            SEQ
            SEQ i=0 FOR Nr.of.Atoms.ext
              ...  Abkuerzungen fuer Atom i (NO LIST)
              SEQ
                fx.i, fy.i := ZERO, ZERO -- Init Kraftakkumulatoren
                SEQ j=0 FOR Nr.of.Atoms.loc
                  REAL32 dx, dy, Rij, df, fi, fx, fy, p:
                  SEQ
                    dx, dy := x - x2[j], y - y2[j]
                    Rij := SQRT((dx*dx)+(dy*dy)) --Abstand
                    p, df :=  f(Rij, (r+atom.radius[j]),
                                      (hard+atom.hardness[j]))
                    pot.energy := pot.energy + (p*Ea)
                    fi := Ea*(df/Rij)
                    fx, fy := fi*dx, fi*dy
                    fx.i, fy.i := fx.i - fx, fy.i - fy -- ACTIO und
                    Fx[j], Fy[j] := Fx[j] + fx, Fy[j] + fy --  REACTIO!
            ext.number:=((my.number-index)+Nr.of.T800.in.Net) REM
                        Nr.of.T800.in.Net -- Zieltransputernummer
            Force.to ! ext.number;
                        Nr.of.Atoms.ext::[extern.Fx FROM 0 FOR Nr.of.Atoms.ext];
                        Nr.of.Atoms.ext::[extern.Fy FROM 0 FOR Nr.of.Atoms.ext]
                    }}}
                  TRUE
                    SKIP --alle anderen einfach nur weiterreichen
              Koord.out ! coordinate;
                Nr.of.Atoms.ext::[extern.x FROM 0 FOR Nr.of.Atoms.ext];
                Nr.of.Atoms.ext::[extern.y FROM 0 FOR Nr.of.Atoms.ext];
                Nr.of.Atoms.ext::[extern.hardness FROM 0 FOR Nr.of.Atoms.ext];
                Nr.of.Atoms.ext::[extern.radius FROM 0 FOR Nr.of.Atoms.ext]
              INT Nr.of.Atoms.ext:
              IF
                add.count < add --schon alle Kraefte da?
                  SEQ --nein!
                    Force.from ? Nr.of.Atoms.ext::extern.Fx;
                                 Nr.of.Atoms.ext::extern.Fy
                    add.count := add.count + 1
                    SEQ i=0 FOR Nr.of.Atoms.loc
                      SEQ
                        Fx[i]:=Fx[i]+extern.Fx[i]
                        Fy[i]:=Fy[i]+extern.Fy[i]
                TRUE
                  SKIP
          }}}
          {{{  Stoppschritt; zur Kompensation des Startschritts, sonst Deadlock
          Koord.inter ? CASE coordinate;
             Nr.of.Atoms.ext::extern.x; Nr.of.Atoms.ext::extern.y;
             Nr.of.Atoms.ext::extern.hardness; Nr.of.Atoms.ext::extern.radius
          }}}
        }}}
        {{{  Integrationsschritt und Rueckspeichern
        SEQ i=0 FOR Nr.of.Atoms.loc
          ...  Abkuerzungen (NO LIST)
          SEQ
            {{{  x-Koordinaten
            h1 IS x1[i]:
            h2 IS x2[i]:
            SEQ
              help := ((TWO*h2)-h1)+(Fx[i]*o)
              v := half.rez.time.step * (help-h1)
              kin.energy:=kin.energy+((v*v)*m)
              h1:=h2
              h2:=help
            }}}
            ...  y-Koordinaten, analog (NO LIST)
        kin.energy:=0.5(REAL32)*kin.energy --ausgeklammert!
        }}}
        {{{  Ausgabe globaler Groessen, wie Energie, ...
        REAL32 glob1, glob2:
        SEQ --Reihenfolge sehr wichtig!
          Koord.inter ? CASE global; glob1; glob2
          Koord.out ! global; glob1+pot.energy;
                              glob2+kin.energy  --das eigene Schaerfchen beitragen
        }}}
      }}}
      ...  buffer (Fuer Koordinaten) (NO LIST)
    }}}
  :
  }}}
  VAL Nr.of.T800.in.Net IS 12: --auch in der Library!!!
  [Nr.of.T800.in.Net + 1]CHAN OF ANY Koord.Ring, Force.Ring:
  PLACED PAR i = 0 FOR Nr.of.T800.in.Net
    PROCESSOR i T8
      PLACE Koord.Ring[i]   AT 4: --Link0.in  Koord.in
      PLACE Koord.Ring[i+1] AT 3: --Link3.out Koord.out
      PLACE Force.Ring[i]   AT 0: --Link0.out Force.out
      PLACE Force.Ring[i+1] AT 7: --Link3.in  Force.in
      Dynamik (Koord.Ring[i], Koord.Ring[i+1], Force.Ring[i+1], Force.Ring[i],i)
```

Dieser Anteil wird umso größer, je mehr Atome das Molekül hat.

3. Der Rechenzeitgewinn von 12 Transputern gegenüber einem liegt zwischen 2,5 bei kleinen Molekülen und 10,75 bei großen.

All das ist leicht zu verstehen: Je mehr Atome im Molekül sind, umso mehr dominiert die quadratisch wachsende Elektrostatik- und vdW-Wechselwirkung die Rechenzeit und prägt auch der Gesamtrechenzeit den quadratischen Charakter auf. Der nur mäßige Gewinn an Rechenzeit bei 12 Transputern im Falle von Alanin liegt mit in der Struktur des Programmes begründet: bei nur fünf Atomen je Transputer macht sich der overhead für Schleifen und das Starten von Kommunikation deutlich bemerkbar. Auch verschlechtert sich so das Verhältnis von Rechenzeit zu Kommunikationszeit (die erste sollte stets sehr viel größer als die zweite sein) sehr. Dies ist in der Praxis aber ohne Bedeutung, da bei der Berechnung der Dynamik großer Moleküle, für die das Programm in der Regel eingesetzt wird, der overhead nur mehr einen verschwindenden Anteil hat. Der Faktor 10,75 liegt außerdem sehr nahe am theoretisch möglichen Faktor von 11,15. (Der Faktor 12 kann aus den oben genannten Gründen nicht erreicht werden.) Dieses gute Skalierungsverhalten rechtfertigt auch die Extrapolation von 12 Transputern auf 50 Transputer.

Vergleichen wir nun die Transputerrechenzeiten mit denen anderer Rechner, wie in *Tabelle 4*. Zwischen den 31,6s der 12 Transputer und den 7,8s der CRAY-XMP am Institut für Plasmaphysik (IPP) in Garching 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 der zur Verfügung stehende Speicher der aufgeführten Rechner für die Berechnung aller Wechselwirkungen mit XPLOR nicht ausreicht.

Tabelle 3: Benchmark-Ergebnisse für Molekulardynamit. Angegeben ist jeweils die Rechenzeit in Sekunden

	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 4: Benchmark-Ergebnisse für Molekulardynamik am Reaktionszentrum. Angegeben ist jeweils die Rechenzeit in Sekunden für einen Dynamikschritt bei Berücksichtigung aller Wechselwirkungen. Hochgerechnete Werte wurden von PTT auf das Reaktionszentrum extrapoliert

VAX 11-750	CONVEX C1		CRAY-XMP
XPLOR	C-Pgm.[10]	XLPOR	XPLOR
2 Stunden (geschätzt)	350 s	147 s (hochgerechnet)	7,8 s (hochgerechnet)

Tabelle 5: Die Kostenaufstellung für einen Rechnerknoten

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

Die 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 enthalten. Leider müssen die Ausgaben für den Speicher der aktuellen, 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 entnimmt man, daß sich mit etwa 2000 DM pro Knoten ein Netzwerk aus 50 Transputern für rund 100 000 DM realisieren läßt. Extrapoliert man die vorher aufgeführten Benchmarkwerte, so kommt man zu der Aussage der Titelzeile.

Literatur:

[1] *J. Buhmann, R. Divko, H. Ritter, K. Schulten:* Physik und Gehirn, mc 9/87, Seiten 108...120

[2] *H. Treutlein, A. Windemuth, K. Schulten:* Molecular Design, mc 1/88, Seiten 46...57

[3] *Gerd Häusler:* Die mc-Transputerkarte, mc 4/88, Seiten 38...44

[4] *Gerd Häusler:* Die Assemblersprache des Transputers, Teil 2, mc 7/88, Seiten 69...73

[5] *W. D. Hillis/J. Barnes:* Programming a highly parallel Computer, Nature Vol. 326, 5.3.1987, Seiten 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) gibt es auch auf Diskette. Wenden Sie sich bitte an die mc-Redaktion.

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