Dynamiksimulation sehr großer Makromoleküle am Beispiel des photosynthetischen Reaktionszentrums von Rhodopseudomonas viridis

Helmut Heller

Dynamiksimulation

sehr großer Makromoleküle

am Beispiel des photosynthetischen

Reaktionszentrums

von *Rhodopseudomonas viridis*

Diplomarbeit

von

Helmut Heller

Physik–Department
der
Technischen Universität München
Institut T30
Prof. Dr. K. Schulten

November 1988

Zweite Auflage, Januar 1989

31.–60. Exemplar

für Benedikt

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Ein Hauptziel der Molekularbiologie ist es, den Zusammenhang zwischen Struktur und biologischer Wirkung von Proteinen zu verstehen. Lange Zeit nahm man an, daß sich die Funktion eines Biopolymers allein aus seiner Struktur ableiten ließe, also aus der Kenntnis der Gleichgewichtslagen der Atome zusammen mit ihren Partialladungen und ihren chemischen Eigenschaften. Die Gleichgewichtslagen lassen sich aus Röntgenstrukturanalysen ableiten, siehe z.B. [25]. In jüngster Zeit hat man erkannt, daß weitere Eigenschaften, die sich aus der statischen Struktur nicht ableiten lassen, für die Funktion von großer Bedeutung sind. Zu diesen dynamischen Eigenschaften zählen die thermischen Mobilitäten der Atome und die gemeinsamen Bewegungen ganzer Gruppen von Atomen. Diese Eigenschaften sind experimentell schon für kleine Moleküle nur schwer oder gar nicht zu ermitteln; bei großen Proteinen ist ein experimenteller Zugang wegen der Fülle der Details nicht möglich. Derartige Informationen gewinnen aber im Rahmen des „molecular modelling“, z.B. für die gezielte Herstellung neuer Makromoleküle mit definierten Eigenschaften, in der medizinischen und chemischen Forschung bzw. Industrie zunehmend an Bedeutung.

Als Ausweg, die benötigten Informationen dennoch verfügbar zu machen, bietet sich die Computersimulation der Dynamik von Biopolymeren an. Durch sie ist es möglich, ohne experimentelle Beschränkungen, gewissermaßen wie durch ein Mikroskop und in Zeitlupe, alle Vorgänge bis ins kleinste Detail zu beobachten (Ortsvergrößerung etwa 10^{10}-fach, Zeitdehnung etwa 10^{14}-fach). Da für große Moleküle mit mehreren tausend Atomen eine quantenchemische Beschreibung wegen des unüberwindlichen Rechenaufwandes nicht in Frage kommt, bleibt nur die Verwendung eines klassischen Molekülmodells (Newtonsche Gleichungen). Dabei werden die quantenchemischen Effekte (Bindungskräfte) in Form empirischer Kraftfelder berücksichtigt. Unverzichtbar ist dabei aber immer wieder die „Eichung“ der Rechnungen durch experimentelle Beobachtungen, denn Ziel ist ja eine möglichst realistische Beschreibung eines Teilbereichs der Natur. Problematisch bei derartigen Beschreibungen sind die Anforderungen an Rechenzeit: selbst auf den größten heute verfügbaren Rechenanlagen sind Stunden bis Tage an Rechenzeit erforderlich, um Zeitspannen im Bereich von Picosekunden (1 ps = 10^{-12} s) simulieren zu können. Für die vollständige Simulation der Dynamik eines Moleküls mit 3 600 Atomen über 1 Nanosekunde (1 ns = 10^{-9} s) unter Berücksichtigung aller Wechselwirkungen würde eine CRAY-XMP schon 93 Tage rechnen. Das Ziel aber ist, Moleküle mit etwa 10 000 Atomen einschließlich umliegender Bereiche wie Wasser, Membranen und Ionen (zusammen etwa 30 000 Atome) über Zeiträume im Bereich von Nanosekunden und länger zu simulieren.

In dieser scheinbar ausweglosen Situation liegt die Lösung nicht in einer Evolution, sondern einer Revolution: der Revolution der Parallelrechner! Die Evolution bestehender Rechenanlagen beruht im wesentlichen auf einer stetigen Verbesserung der zugrundeliegenden Hardware. So sinken die Verarbeitungszeiten immer weiter, während durch zunehmende Miniaturisierung die Packungsdichte steigt. Doch für diese Art der Entwicklung gibt es eine natürliche physikalische Grenze: sobald die Strukturen in den integrierten Schaltkreisen so klein werden, daß Quanteneffekte eine Bedeutung gewinnen, muß die herkömmliche Technologie versagen. Und diese Grenze ist fast erreicht. Daher ist die Zeit reif, sich nach Alternativen umzusehen. Die Idee ist, zur Leistungssteigerung mehrere Rechner parallel zu schalten. Diese Idee ist nicht neu und der auf diese Weise erreichbaren Leistungssteigerung sind keine prinzipiellen Grenzen gesetzt. Hinzu kommt, daß sich im Vergleich zu herkömmlichen, sog. sequentiellen Rechnern dieselbe Rechenleistung durch Verwendung einiger, nicht ganz so leistungsstarker Rechenelemente wesentlich billiger realisieren läßt. So gibt es auch schon zahlreiche Anstrengungen in dieser Richtung [18, 34, 42, 71], doch war der Aufbau von Parallelrechnern bisher sehr aufwendig. Mit der Entwicklung des Transputers[1] hat sich diese Situation gewandelt [3].

Nun ist es aber mit dem Bau eines Parallelrechners allein nicht getan, denn es ist darüberhinaus notwendig, Algorithmen zu entwickeln, die *mehrere Rechner* zur Lösung *einer Aufgabe* einsetzen. An der Umsetzung von Programmen für sequentielle Rechner in Programme für Parallelrechner (Parallelisierung) wird momentan intensiv gearbeitet [18, 20, 22, 30, 48, 53, 55, 58]. Auch diese Arbeit möchte hierzu einen Beitrag leisten, indem sie ein Problem aus der Biophysik, die Dynamiksimulation von Proteinen, aufgreift und parallelisiert. Dazu war es erforderlich, eine geeignete Parallelisierungsstrategie zu finden und diese in Hardware und Software umzusetzen. Die Arbeiten hierzu begannen im November 1987. Damals war geeignete Hardware käuflich nicht zu erhalten, so daß eine Eigenentwicklung erfolgen mußte. Das Ergebnis hiervon ist der

[1] Die Bezeichnung *Transputer* ist aus den Worten **Trans**istor, dem Grundbauelement heutiger Rechner, und Com**puter** zusammengesetzt und soll andeuten, daß die Herstellerfirma INMOS den Transputer als Grundbauelement zukünftiger Parallelrechner betrachtet.

Transputer-Parallelrechner ALTER EGO–12, der in Kapitel 4 vorgestellt wird. Weiteres Ziel der Eigenentwicklung war es, zu zeigen, zu welch geringen Kosten sich ein solcher Rechner aufbauen läßt (siehe Abschnitt 4.6). Weil nur das „Gespann“ Rechner und Programm zusammen bei der Lösung biophysikalischer Fragestellungen helfen kann, wurde für diese Arbeit außerdem noch das umfangreiche parallele Programm EGOentwickelt. Diese beiden Komponenten zusammen erlauben nun, in Bereiche der Molekulardynamiksimulation vorzustoßen, die bisher wegen der auf herkömmlichen Großrechnern um etwa einen Faktor 100 höheren Rechenkosten verschlossen waren.

Zu diesen Bereichen zählt die erstmalige Simulation der Dynamik des vollständigen photosynthetischen Reaktionszentrums des schwefelfreien Purpurbakteriums *Rhodopseudomonas viridis* unter Berücksichtigung aller Wechselwirkungen. Dieser Proteinkomplex ist in die Zellmembran des Bakteriums eingebaut und hat die Aufgabe, Licht zu absorbieren und mit dieser Energie einen Elektronengradienten über die Membran aufzubauen. Durch Kopplung an andere chemische Prozesse kann der Elektronengradient in einen Protonengradienten, eine erste biologisch verwertbare Energieform, umgewandelt werden. Im Reaktionszentrumlaufen somit die entscheidenden Prozesse der Absorption von Sonnenlicht und der Umwandlung dieser Energie in chemische Energie ab. Da das Reaktionszentrumdes Photosystems II der grünen Pflanzen[2] dem hier betrachteten bakteriellen Reaktionszentrumsehr ähnelt [50], ist es hier möglich, Prozesse zu studieren, welche die energetische Grundlage des Lebens auf der Erde bilden. Voraussetzung dazu war die Aufklärung der Struktur dieses Proteins mit Hilfe der Röntgenstrukturanalyse [25, 26] durch Michel, Deisenhofer und Huber am Max-Planck-Institut für Biochemie in Martinsried bei München. 1988 wurde den drei Wissenschaftlern dafür der Nobelpreis für Chemie verliehen.

Bevor die Entwicklung neuer Rechenstrategien in Angriff genommen werden kann, ist es nötig zu wissen, wie man herkömmlicherweise biologische Makromoleküle im Rechner beschreibt. Auf diesem Sektor ist schon viel Arbeit geleistet worden [21, 45, 46, 64, 69]. Die dabei entwickelten Konzepte lassen sich auf Parallelrechner übertragen und wurden von uns übernommen; sie sind kurz im Kapitel 2 beschrieben. Kapitel 3 stellt die durchgeführten Berechnungen und ihre Analysen und Ergebnisse vor. In Kapitel 4 findet sich eine Beschreibung der erstellten Hardware und Software: zuerst wird noch einmal das Kernproblem dargestellt und eine geeignete Lösungsstrategie erarbeitet. Bevor diese besprochen wird, wird der Grundbaustein „Transputer“ und seine Programmiersprache occam IIvorgestellt. Es schließt sich eine Beschreibung des selbst entwickelten und gebauten Rechners an. Im Abschnitt 4.4.3.2 wird die Implementierung der Lösungsstrategie auf diesem Rechner beschrieben. Den Abschluß des Kapitels bilden Überlegungen

[2]Pflanzen besitzen zwei verschiedene Reaktionszentren; dies ist für die Spaltung von Wasser notwendig.

zu Geschwindigkeit und Kosten des Rechners sowie Betrachtungen der Kompatibilität des erstellten Programms mit vorhandenen Programmen. In Kapitel 5 werden die wichtigsten Ergebnisse zusammengefaßt.

Die Durchführung eines derart umfangreichen Projekts [Entwicklung der Schaltpläne, Aufbau eines Prototypenrechners, Design des Leiterplattenlayouts, Beschaffung aller Bauteile, Bestückung der Platinen, Austesten des Rechners und Beseitigung der Fehler („hardware debugging“), Entwurf einer Parallelisierungsstrategie für Molekulardynamiksimulationen, deren Umsetzung in ein effizientes, paralleles Programm, dessen Implementation in der neuen Programmiersprache occam IIauf dem selbstgebauten Rechner, Austesten der Software und Fehlerbeseitigung („software debugging“), Laufzeitoptimierungen am Programm und schließlich Durchführung von molekulardynamischen Simulationsrechnungen und deren Auswertung] innerhalb des vorgegebenen zeitlichen Rahmens von einem Jahr wäre für mich alleine unmöglich gewesen. Daher fand von Anfang an eine Zusammenarbeit mit Helmut Grubmüller [36] statt, die sich als äußerst fruchtbar erwies. Mit Fortschreiten der Arbeit und unter dem größer werdenden Zeitdruck verstärkte sich die Trennung der Bereiche Hardware und Software. Helmut Grubmüller führte das aufwendige Design der dreilagigen Leiterplatte aus, während sich der Autor der Erstellung des Programms widmete. Durch diese Trennung, die jeweils ein sehr eigenständiges Arbeiten ermöglichte, konnte die Effizienz der Arbeit gesteigert werden, was für die Einhaltung des zeitlichen Rahmens erforderlich war. Trotzdem wurden alle weitreichenderen Entscheidungen gemeinsam oder in Abstimmung getroffen.

Vor den Beginn der eigentlichen Arbeit möchte ich noch eine Entschuldigung stellen: viele Begriffe in der elektronischen Datenverarbeitung entstammen dem englischen Sprachraum. Obzwar es für manche Fachwörter deutsche Übersetzungen gibt (beispielsweise heißt „stack“ im Deutschen „Kellerspeicher“), behalte ich in dieser Arbeit die englischen Bezeichnungen bei, sofern diese die gebräuchlicheren sind. Viele solche Wörter haben sogar schon Eingang in die deutsche Sprache gefunden (z.B. „Hardware“). Zur Erklärung von nicht allgemein geläufigen Begriffen findet sich am Ende der Arbeit ein Glossar, auf das im laufenden Text durch (⇒G) verwiesen wird.

Kapitel 2

Proteine und ihre Dynamiksimulation auf Computern

Um die Dynamik eines Moleküls berechnen zu können, müßte man im Prinzip die Schrödingergleichung

$$\mathcal{H}|\Psi\rangle = i\hbar\frac{\partial}{\partial t}|\Psi\rangle \tag{2.1}$$

für ein Vielteilchensystem aus Elektronen und Kernen lösen. Solche quantenchemischen Rechnungen sind schon für kleine Moleküle sehr schwierig und bei großen Molekülen gänzlich undurchführbar. Daher ist man gezwungen, ein klassisch mechanisches Modell für die Kernbewegung einzuführen[1] und die intramolekularen Kräfte aus einem semiempirischen Potential abzuleiten. Dieses Potential, das von den Elektronen erzeugt wird, beinhaltet in seiner Form und in den Kraftkonstanten die quantenmechanischen Aspekte der elektronischen Wechselwirkung. Man muß nun also „nur noch" die klassischen Newtonschen Bewegungsgleichungen der Atomkerne

$$\vec{F}_i = m_i\frac{d^2}{dt^2}\vec{r}_i = -\nabla_i E(\vec{r}_1, \ldots \vec{r}_N) \tag{2.2}$$

mit einer komplizierten Energiefunktion E lösen. Dabei ist $\vec{r}_i$ der Ort des i-ten Atoms mit der Masse m_i, $\vec{F}_i$ ist die Kraft auf das Atom i, $i = 1, \ldots N$, $\nabla_i = (\frac{d}{dx_i}, \frac{d}{dy_i}, \frac{d}{dz_i})^T$, und N ist die Zahl der Atome im Molekül.

[1] *Born-Oppenheimer*-Näherung: wegen des großen Massenunterschieds ($\gg 10^3$) zwischen Elektronen und Kernen können sich erstere fast augenblicklich auf jede Kernkonfiguration einstellen. Daher können Elektronen- und Kernfreiheitsgrade getrennt betrachtet werden [32]. Die Kerne bewegen sich in der Energielandschaft, die von den Elektronen erzeugt wird.

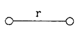
2.1 Die Energiefunktion

Die empirische Energiefunktion E stellt zusammen mit den Koordinaten der Atome das Molekül im Rechner dar. Sowohl an der Bestimmung der Atomkoordinaten [24], als auch der Kraftkonstanten der Energiefunktion wurde in den letzten Jahren kontinuierlich gearbeitet. Die Kenntnis dieser Größen bildet die Grundlage der Simulation. Unser paralleles Programm EGObenutzt diejenige Energiefunktion, die für die Simulationsprogramme XPLOR [23] und CHARMM [21] entwickelt wurde. Sie setzt sich aus folgenden Beiträgen zusammen

$$E = E_b + E_\theta + E_\phi + E_\omega + E_{el} + E_{vdW} + (E_{Hb} + E_Z) \quad , \tag{2.3}$$

wobei die letzten beiden, in Klammern stehenden Energiebeiträge in EGOnoch nicht implementiert sind. Im einzelnen berechnen sich die Energiebeiträge wie folgt:

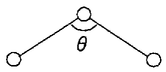
E_b: Änderungen des Abstandes r_b zweier chemisch gebundener Atome i und j geben näherungsweise zu einem harmonischen Potential Anlaß



$$\left.\begin{aligned} E_b(\vec{r}_i, \vec{r}_j) &= K_b(r_b - r_0)^2 \, , \; r_b = |\vec{r}_i - \vec{r}_j| \\ E_b &= \sum_{\substack{\text{Bindung zwischen} \\ \text{Atom } i \text{ und } j}} E_b(\vec{r}_i, \vec{r}_j) \quad . \end{aligned}\right\} \tag{2.4}$$

r_0 ist der Gleichgewichtsabstand der beiden an der Bindung beteiligten Atome und K_b ist die Kraftkonstante. Die Atome führen also hochfrequente Streckschwingungen entlang der Bindungen aus.

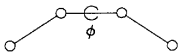
E_θ: Aufgrund der Gestalt der Elektronenorbitale haben die Winkel zwischen Bindungen bevorzugte Werte, d.h. es gibt zwischen je drei verbundenen Atomen ein Winkelbiegungspotential, das ebenso wie bei der Bindungslänge harmonisch genähert wird



$$E_\theta = \sum_{\text{Bindungswinkel}} k_\theta(\theta - \theta_0)^2 \quad . \tag{2.5}$$

k_θ und θ_0 sind die Kraftkonstante und der Gleichgewichtswinkel analog zu Gleichung (2.4).

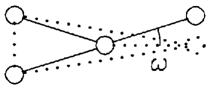
E_ϕ: Dieser Energieterm beschreibt Dihedralschwingungen, also Torsionen entlang einer Bindung. Das Potential ist periodisch und wird durch eine Cosinusfunktion approximiert



$$\left.\begin{aligned} E_\phi &= \sum_{\text{Dihedralwinkel}} \left[|k_\phi| - k_\phi \cos(n\phi - \phi_0) \right], \\ & n \in \{1, 2, 3, 4, 6\} \quad . \end{aligned}\right\} \tag{2.6}$$

ϕ_0 ist wieder der Ruhelagenwinkel, k_ϕ die Kraftkonstante, die negativ ist, falls die *cis*-Konformation (diese ist in der Abbildung symbolisiert) die energetisch günstigere ist, und n ist die Symmetriezahl.

E_ω: In biologischen Makromolekülen gibt es häufig planare Strukturen mit ausgedehnten konjugierten Elektronensystemen, etwa aromatische Ringe oder Porphyrine wie z.B. in Chlorophyll und Pheophytin. Diese werden durch die bisher charakterisierten Bindungspotentiale nur unzulänglich beschrieben. Daher führt man sogenannte Extraplanarwinkel ω ein, die eine Abweichung von der Planarität beschreiben



$$E_\omega = \sum_{\text{Extraplanarwinkel}} k_\omega (\omega - \omega_0)^2 \quad . \tag{2.7}$$

ω_0 ist der Ruhelagenwinkel und k_ω die Kraftkonstante.

E_{el}: Die elektrostatische Energie spielt sowohl eine entscheidende Rolle für die Bildung der sekundären und tertiären Struktur (⇒G) als auch für die biologische Funktion eines Proteins [47, 68]. Sie wird durch das Coulombsche Gesetz

$$E_{el} = \sum_{\langle i,j \rangle,\, i \neq j} \frac{q_i q_j}{4\pi\varepsilon_0 \varepsilon_r r_{ij}} \tag{2.8}$$

beschrieben. q_i und q_j sind die Partialladungen der Atome i und j, r_{ij} ist der Betrag ihres Abstandes, ε_0 ist die elektrische Feldkonstante und ε_r ist die relative Dielektrizitätskonstante. Letztere wird meist zu 1 (Vakuumwert) oder 2–4 gewählt, wobei letztere Werte die elektronische Polarisierbarkeit der Atome und chemischen Bindungen berücksichtigen.

Da das elektrostatische Potential nur mit $1/r$ abfällt, ist seine Reichweite sehr groß. Zu seiner korrekten Berechnung muß sich die Summation in Gleichung (2.8) über alle Atompaare erstrecken. Im Abschnitt 4.2 wird die Berechnung dieses Energiebeitrags noch ausführlich erläutert, da sie die meiste Zeit beansprucht.

Bei der Summe über alle Paare $\langle ij \rangle$ sind solche Paare ausgeschlossen, die kovalent oder über Bindungswinkel verbunden sind, da die Wechselwirkung dieser Paare schon durch die oben genannten Potentiale [Gleichungen (2.4, 2.5)] berücksichtigt wird. Darüber hinaus lassen sich auch die Nicht-Bindungswechselwirkungen zwischen dritten Nachbarn („1–4 Wechselwirkung“) unterdrücken, da sie zum Teil schon in den Potentialen nach Gleichungen (2.6, 2.7) berücksichtigt sind.

Im Programm EGOläßt sich dieser Ausschlußmodus durch einen Parameter im Steuerungs-fold (siehe Abschnitt 4.4.3.2) kontrollieren.

E_{vdW}: Im Programm wird gemeinsam mit der elektrostatischen Energie auch die van der Waals Energie

$$E_{vdW} = \sum_{\langle i,j\rangle,\, i\neq j} \left(\frac{A}{r_{ij}^{12}} - \frac{B}{r_{ij}^{6}} \right) \tag{2.9}$$

berechnet, da sich beide Summationen über dieselben Paare erstrecken. So muß der Abstand für beide Wechselwirkungen nur einmal berechnet werden.

Unter dem Begriff ‚van der Waals Energie' sind in der Molekulardynamik zwei Energiebeiträge zusammengefaßt: die aus dem Pauliprinzip resultierende abstoßende Austauschwechselwirkung ($\sim 1/r^{12}$) und die durch induzierte Dipol-Dipol-Wechselwirkung hervorgerufene anziehende van der Waals Wechselwirkung ($\sim 1/r^{6}$). Zur Modellierung wird das in Gleichung (2.9) aufgeführte Lennard-Jones-Potential verwendet.

E_{Hb}: Die Wasserstoffbrückenenergie hängt nicht nur vom Abstand der beteiligten Atome, sondern auch vom Winkel zwischen der Wasserstoffbrücke und der Bindung mit einem, dem Akzeptor benachbarten Atom[2] ab. Aufgrund von Speicherplatzproblemen sowie wegen der nur ungenau bekannten Position der Wasserstoffatome wird bei großen Molekülen, wie zum Beispiel beim Reaktionszentrum, dieses explizite Wasserstoffbrückenpotential nicht angewendet (die genaue Form ist in [69] angegeben). Stattdessen werden Wasserstoffbrücken im Rahmen der schon besprochenen Wechselwirkungen durch stark polarisierte elektrische Dipole beschrieben [64].

E_Z: Manche strukturellen Änderungen des Moleküls sind Artefakte, die etwa daher rühren, daß die Umgebung (Wasser, Lipidschicht, etc.) unvollständig simuliert wird. Um solche Artefakte zu verhindern, kann man die Atome an den Orten der Röntgenstruktur mit geeigneten Potentialen festbinden

$$E_Z = \sum k_z (r_z - r_{z_0})^n\,, \quad n \in \mathcal{Z} \quad . \tag{2.10}$$

Daneben können in E_Z auch noch Zwangskräfte enthalten sein, die Bindungslängen oder -winkel starr halten.

Die Kräfte auf die einzelnen Atome werden dann aus der Energiefunktion durch (symbolische) Differentiation (2.2) gewonnen. Eine ausführlichere Beschreibung der einzelnen Terme findet sich in [69]; von dort wurden auch mit freundlicher Genehmigung die Zeichnungen übernommen.

[2] dem sogenannten Akzeptorantezedenten

2.2 Gewinnung der Parameterwerte

Um Simulationen ausführen zu können, müssen die Werte aller oben eingeführten Parameter bekannt sein. Dazu führt man entweder genaue quantenchemische Rechnungen an kleineren Molekülen, die den Proteinsegmenten ähnlich sind, durch oder bestimmt das Kraftfeld empirisch, indem man die Kraftkonstanten solange variiert, bis die berechneten Observablen (Schwingungsfrequenzen) mit den experimentellen Beobachtungen (Infrarotspektroskopie) übereinstimmen. Man geht dabei davon aus, daß sich chemische Bindungen gleicher Art in großen Molekülen nicht anders verhalten als in kleinen.

Ein Problem ist die Bestimmung der Partialladungen q_i und q_j. Bei den meisten heute verwendeten Simulationsprogrammen und auch bei EGObleiben diese Ladungen während des ganzen Simulationsverlaufs unverändert. Außerdem bekommen sie zu Beginn der Simulation Werte zugewiesen, die unabhängig von der tatsächlichen räumlichen Proteinstruktur sind. Es wird also die Polarisierbarkeit von Proteinbausteinen vernachlässigt.

2.3 Anfangsbedingungen

Für die Integration der Newtonschen Gleichungen (2.2) benötigt man je einen Satz von Koordinaten und von Geschwindigkeiten.

Die Koordinaten erhält man durch Röntgenstrukturanalyse. Aus den Bragg-Reflexionen von Röntgenstrahlen an einem Proteinkristall bestimmt man nach einer komplizierten Analyse die ortsabhängige Dichte der Elektronen. Anschließend muß eine gegebene Aminosäuresequenz möglichst widerspruchsfrei in die Elektronendichteverteilung eingepaßt werden. Die so gewonnenen Koordinaten stellen aber keine „Momentaufnahme" des Proteins dar, sondern sind als Mittelwerte aufzufassen[3]. Auch sind die *in vitro* untersuchten Kristalle in ihrer Struktur nicht unbedingt identisch mit den *in vivo* vorkommenden Proteinen, denn oft werden zum Kristallisieren Lösungsmittel verwendet, die in der lebenden Zelle nicht vorkommen.

Neuere Untersuchungen [26] bieten eine Ortsauflösung von 0.1–0.2 Å; berücksichtigt man ferner die Beweglichkeit der Atome, welche im Bereich zwischen 0.45 Å (für Kohlenstoffatome des „Rückgrates") bis 0.55 Å (als Mittelwert bei Berücksichtigung aller Atome) liegt, so sind die Positionen einzelner Atome bis auf etwa 0.7 Å genau bestimmt [64]. Um die tatsächlichen Orte der Atome möglichst gut zu beschreiben,

[3]Eine mittlere Struktur ist nicht unbedingt eine physikalisch, bzw. chemisch sinnvolle. Für ein Atom in einem symmetrischen Doppelminimumpotential ist die mittlere Lage genau in der Mitte zwischen den beiden Minima, d.h. an einer Stelle mit sehr kleiner Aufenthaltswahrscheinlichkeit.

berücksichtigt man beim Auswerten der Daten auch die Bindungsverhältnisse im Protein. Oft werden Bindungslängen nahe den Gleichgewichtsabständen gewählt und festgehalten. Daraus resultiert eine unnatürlich niedrige Energie in den Bindungsfreiheitsgraden, welche im Lauf der Dynamiksimulation ansteigt (siehe auch Abbildung 3.6). Die Röntgenstruktur beschreibt also ein (bezüglich der Bindungen) lokal entspanntes Protein. Dagegen sind die einzelnen Gruppen (z.B. Aminosäuren) zueinander noch nicht in ihrem energetischen Minimum. Dies drückt sich in hohen Werten der van der Waals Energie zu Beginn der Simulation aus (siehe auch Abbildung 3.9), da diese am empfindlichsten auf zu kleine Abstände reagiert ($\sim 1/r^{12}$). Aus diesen Gründen muß man das Protein vor der eigentlichen Dynamiksimulation einer Minimierung und Equilibrierung unterziehen, um mit einer realistischen Struktur arbeiten zu können. Diese Schritte sind in den Abschnitten 2.4.3 und 2.4.4 beschrieben und wurden in Kapitel 3 auch, soweit bekannt, erstmals für das gesamte Reaktionszentrumdurchgeführt.

Für die Geschwindigkeiten der einzelnen Atome kann man experimentell keine Werte erhalten. Da sie aber alle Voraussetzungen[4] des Gleichverteilungssatzes

$$\langle v_i^2 \rangle = \frac{3k_B T}{m_i} \tag{2.11}$$

erfüllen, kann man die einzelnen Geschwindigkeitskomponenten mit Gauß-verteilten Zufallszahlen besetzen. Der Betrag der Geschwindigkeiten besitzt dann die bekannte Maxwellsche Verteilung (siehe Abschnitt 3.2.4). Aus (2.11) errechnet sich nach Summation über alle Atome (im thermischen Gleichgewicht) die Temperatur zu

$$T = \frac{1}{3k_B N} \sum_{i=1}^{N} \langle v_i^2 \rangle m_i \quad . \tag{2.12}$$

2.4 Dynamik

Schon in der Einleitung wurde betont, daß die Zeitspannen, über die große Moleküle simuliert werden können, sehr kurz sind. Daher simuliert man vornehmlich Moleküle, bei denen biologisch wichtige Prozesse auf sehr kurzen Zeitskalen ablaufen. Dazu zählen die primären Prozesse der Photosynthese, wie sie im Purpurbakterium *Rhodopseudomonas viridis* ablaufen. Der erste Elektronentransferschritt ist innerhalb von 2.8 ps [19] beendet. Damit ist eine Verschiebung von Molekülsegmenten verbunden [65, 66], die durch Simulationen untersucht werden kann. Ferner können die Beweglichkeiten von Atomen oder Atomgruppen und Korrelationsfunktionen untersucht werden.

---

[4] Erstens muß gelten $E_{\text{tot}} = \sum_{i=1}^{N} \epsilon_i(v_i) + E'(\vec{r}_1 \ldots \vec{r}_N)$ wobei v_i die Geschwindigkeit des Atoms i ist. Zweitens muß $\epsilon_i(v_i)$ eine quadratische Funktion von v_i sein. [57, Seite 292ff]

2.4.1 Integration

Der Hauptteil an Rechenzeit wird für die Berechnung der Energiefunktion und ihrer Ableitungen (Kräfte) benötigt. Daher sollte diese Funktion pro Integrationsschritt nur einmal ausgerechnet werden. Methoden wie das bekannte Runge-Kutta Verfahren oder Extrapolationsmethoden scheiden damit aus. Einfache Einschrittverfahren (z.B. das Eulersche Polygonzugverfahren [63, Seite 106]) haben keine ausreichende numerische Stabilität. Es wird daher in der Molekulardynamik fast ausschließlich das *Verlet*-Verfahren [67] verwendet. Es ist das einfachste Beispiel eines Mehrwert-Prediktor-Korrektor Verfahrens und ist in der zweiten Ordnung des Zeitschritts Δt genau:

$$\begin{aligned} \vec{r}_i(t+\Delta t) &= 2\vec{r}_i(t) - \vec{r}_i(t-\Delta t) + \frac{\vec{F}_i(t)}{m_i} \cdot (\Delta t)^2 && (2.13) \\ \vec{F}_i(t) &= -\nabla_i E(\vec{r}_1(t), \ldots \vec{r}_N(t)) \quad . && (2.14) \end{aligned}$$

Hier werden zwei aufeinanderfolgende Koordinatensätze $\vec{r}(t-\Delta t)$ und $\vec{r}(t)$ zur Integration verwendet. Nur zu Integrationsbeginn benutzt man statt dessen (wie oben beschrieben) einen Koordinaten- $(\vec{r}_i(0))$ und einen Geschwindigkeitensatz $(\vec{v}_i(0))$

$$\vec{r}_i(\Delta t) = \vec{r}_i(0) + \vec{v}_i(0) \cdot \Delta t + \frac{1}{2}\frac{\vec{F}_i(0)}{m_i} \cdot (\Delta t)^2 \quad . \tag{2.15}$$

Besondere Beachtung verdient die Größe des Zeitschritts Δt. Wird er zu klein gewählt, so verschenkt man Computerzeit. Seine maximale Größe wird durch die schnellsten im System vorkommenden Freiheitsgrade bestimmt. Dies sind im allgemeinen die Bindungslängenschwingungen, vornehmlich die von Wasserstoffatomen. In der Literatur werden Integrationszeitschritte von 0.5 fs [21, 69] bis 1.0 fs [64] empfohlen.

2.4.2 Compound-Atome

Ein „Atom“ in der Computerdarstellung kann unter Umständen mehreren natürlichen Atomen entsprechen und wird dann als *compound-Atom* bezeichnet. Häufig werden die Wasserstoffatome ihren Bindungspartnern zugerechnet, so daß sie nicht mehr einzeln simuliert werden müssen. Jedes compound-Atom wird im Rechner als ein Objekt mit der Summe der Massen seiner Konstituenten behandelt. Dadurch reduziert sich bei gewöhnlichen biologischen Molekülen die Anzahl der Atome auf etwa die Hälfte, was die Simulation größerer Moleküle ermöglicht. Ferner kann man durch diese Vereinfachung in einigen Fällen die schnellsten Freiheitsgrade im Molekül (Streckschwingungen der Bindungslängen von Wasserstoffatomen) eliminieren und den Zeitschritt der Integration entsprechend vergrößern, da diese Freiheitsgrade nur schwach an andere koppeln. Außerdem ist die genaue Position der Wasserstoffatome in der Regel unbekannt, da sie in der Röntgenstrukturanalyse nicht beobachtet werden.

Durch Verwendung von compound-Atomen vernachlässigt man Dipolmomente, die etwa bei C–H Bindungen auftreten können. Im allgemeinen aber ist die Näherung ausreichend.

2.4.3 Minimierung

In Abschnitt 2.3 wurde schon erwähnt, daß die Röntgenstruktur für eine Dynamiksimulation erst equilibriert werden muß. Dazu kann man entweder mit einer Gradientenabstiegsmethode oder mit *simulated annealing* arbeiten. Erstere Methode wird überwiegend angewendet, liefert sie doch mit wenig Rechenaufwand gute Ergebnisse. Wir haben uns jedoch für die Methode des simulated annealing entschieden, da sie einfacher zu implementieren ist. Darüber hinaus darf man vermuten, daß dieses Verfahren realistischere Ausgangsdaten für Molekulardynamikrechnungen liefert [69, Seite 32].

Simulated annealing ist eine normale Molekulardynamiksimulation, doch wird den kinetischen Freiheitsgraden kontinuierlich durch Reskalierung der Geschwindigkeiten Energie entzogen. Man führt also auf mikroskopischer Ebene eine Reibung ein. Da alle Freiheitsgrade untereinander gekoppelt sind, teilt sich dieser Energieverlust auch den Freiheitsgraden der potentiellen Energie mit, so daß die potentielle Energie dann ebenfalls abnimmt. Je geringer man die Reibungskonstante wählt, um so länger dauert zwar die Minimierung (exponentieller Abfall der Geschwindigkeiten), doch um so tiefer liegend kann auch das eingenommene Minimum werden.

Das Molekül durchläuft während der Minimierung keine thermischen Gleichgewichtszustände, sondern es stellt sich eine Verteilung der Energie auf die verschiedenen Freiheitsgrade ein, die vom thermischen Gleichgewicht weit entfernt ist, da man nicht die (Rechen-)Zeit hat, adiabatisch abzukühlen.

2.4.4 Equilibrierung

Ist die Temperatur des Moleküls bis nahe an die Zieltemperatur (meist 300 K) abgesunken, so muß man dafür sorgen, daß sich wieder eine ausgeglichene Energieverteilung über *alle* Freiheitsgrade einstellt. In dieser Phase, der Equilibrierung, wird die Temperatur durch Reskalierung der Geschwindigkeiten auf dem Wert der Zieltemperatur festgehalten. Dies entspricht einer starken Kopplung des Moleküls an ein Wärmebad. Energieerhaltung ist auch hier nicht gegeben. Unter diesen Bedingungen wird über einen ausreichenden Zeitraum simuliert ([69, Seite 33] spricht von 25 fs), bis sich schließlich ein Gleichgewicht aller Freiheitsgrade eingestellt hat.

Dieses Gleichgewicht überprüft man am besten durch eine anschließende Molekulardynamiksimulation ohne äußere Beeinflussung. Bleibt die Temperatur hierbei (bis auf natürliche Schwankungen $\Delta T/T \approx 1/\sqrt{N}$) konstant, so kann die erhaltene Struktur als Basis für die eigentlichen Untersuchungen mittels Molekulardynamiksimulation verwendet werden.

Kapitel 3

Ausgeführte biophysikalische Rechnungen

3.1 Rechnungen an PTI

Vor Beginn der eigentlichen Simulationsrechnungen am Reaktionszentrumvon *Rhodopseudomonas viridis* wurde das Programm EGOvielfach an kleineren Molekülen (PTI[1] und Deka-Alanin (⇒G)) auf Korrektheit und Zuverlässigkeit überprüft. Stets stimmten die von EGOberechneten Trajektorien und Energiewerte (bis auf geringe Abweichungen aufgrund unterschiedlicher Maschinenarithmetik) mit den von XPLOR [23] oder dem Simulationsprogramm MD von Windemuth [69] berechneten überein. Eine der Simulationen von PTI sei hier stellvertretend vorgestellt.

3.1.1 Zeitumkehrinvarianz

Um die Zuverlässigkeit des Programms EGOzu überprüfen, haben wir einen Vorschlag von Windemuth [69, Seite 28] aufgegriffen und die Zeitumkehrinvarianz der numerischen Simulation getestet.

3.1.1.1 Grundlegendes

Die Newtonschen Gleichungen (2.2) sind zeitumkehrinvariant, da sie die Zeit nur in gerader Ordnung enthalten. In numerischen Simulationen muß dieses Verhalten nicht unbedingt widergespiegelt werden, wie folgendes Beispiel mit dem Eulerschen Polygonzugverfahren [63, Seite 106] zeigt (wegen der größeren Übersichtlichkeit wird nur ein

[1] **P**ancreatic **T**rypsin **I**nhibitor [27]

Teilchen in einem ein-dimensionalen Potential betrachtet). Für die numerische Lösung der Newtonschen Gleichungen werden die Zeit t, der Ort x, die Geschwindigkeit v und die Kraft F diskretisiert

$$\begin{aligned} t &= n \cdot \Delta t && (3.1)\\ x(t) &= x(n \cdot \Delta t) =: x_n && (3.2)\\ v(t) &= v(n \cdot \Delta t) =: v_n && (3.3)\\ F(x(t)) &= F(x(n \cdot \Delta t)) = F(x_n) =: F_n \quad . && (3.4) \end{aligned}$$

Der Zeitschritt Δt tritt als frei wählbarer Parameter auf, dessen Wert die Güte des Näherungsverfahrens bestimmt. Unter Verwendung dieser Diskretisierung liefert das Eulersche Polygonzugverfahren eine Rekursionsgleichung für den Ort x_n und die Geschwindigkeit v_n des Teilchens zu den Zeitpunkten $n \cdot \Delta t$

$$\left.\begin{aligned} x_{n+1} &= x_n + v_n \cdot \Delta t \\ v_{n+1} &= v_n + \frac{1}{m} \cdot F_n \cdot \Delta t \quad . \end{aligned}\right\} \tag{3.5}$$

Zeitumkehr wird durch den Übergang $v \to -v$ ereicht. Führt man den nächsten Iterationsschritt mit der geänderten Geschwindigkeit aus, so erhält man

$$\left.\begin{aligned} x_{n+2} &= x_{n+1} - v_{n+1} \cdot \Delta t \\ v_{n+2} &= -v_{n+1} + \frac{1}{m} \cdot F_{n+1} \cdot \Delta t \quad . \end{aligned}\right\} \tag{3.6}$$

Hierin kann man nun x_{n+1} und v_{n+1} durch die Werte in Gleichung (3.5) ersetzen

$$\left.\begin{aligned} x_{n+2} &= x_n + v_n \cdot \Delta t - (v_n + \frac{1}{m} \cdot F_n \cdot \Delta t) \cdot \Delta t \\ v_{n+2} &= -(v_n + \frac{1}{m} \cdot F_n \cdot \Delta t) + \frac{1}{m} \cdot F_{n+1} \cdot \Delta t \quad . \end{aligned}\right\} \tag{3.7}$$

Dies vereinfacht sich zu

$$\left.\begin{aligned} x_{n+2} &= x_n - \frac{1}{m} \cdot F_n \cdot (\Delta t)^2 \\ v_{n+2} &= -v_n + (\frac{1}{m} \cdot F_{n+1} - \frac{1}{m} \cdot F_n) \cdot \Delta t \quad . \end{aligned}\right\} \tag{3.8}$$

x_{n+2} und x_n unterscheiden sich hierbei um einen Betrag $(\frac{F_n}{m} \cdot (\Delta t)^2)$, der von der Größe des Zeitschritts Δt abhängt; eine exakte Zeitumkehrinvarianz, d.h. $x_{n+2} = x_n$, ist nicht gegeben. Vielmehr kann man an der Güte der Zeitumkehrinvarianz die Wahl des Integrationszeitschritts überprüfen: wird Δt zu groß gewählt, weicht die rückwärts durchlaufene Trajektorie schon nach wenigen Schritten erheblich von der vorwärts durchlaufenen ab.

Das Zweischrittverfahren nach Verlet [67] dagegen gewährleistet die Umkehrinvarianz. Es sei hier noch einmal die Rekursionsgleichung (2.13) wiederholt

$$x_{n+1} = 2x_n - x_{n-1} + \frac{F_n}{m} \cdot (\Delta t)^2 \quad . \tag{3.9}$$

Aus dem alten Ort x_{n-1} und dem aktuellen Ort x_n wird hier der neue Ort x_{n+1} berechnet. Zeitumkehr heißt hier, daß aus dem alten Ort x_{n+1} und dem aktuellen Ort x_n der neue Ort x_{n+2} berechnet wird (d.h. alter und aktueller Ort werden vertauscht)

$$x_{n+2} = 2x_n - x_{n+1} + \frac{F_n}{m} \cdot (\Delta t)^2 \quad . \tag{3.10}$$

Setzt man hierin Gleichung (3.9) für x_{n+1} ein, so erhält man:

$$x_{n+2} = x_{n-1} \quad . \tag{3.11}$$

Das Verlet-Verfahren läuft also bei Zeitumkehr *unabhängig von der Größe des Zeitschritts* bis auf numerische Fehler stets wieder auf genau derselben Trajektorie zurück, auf der die Vorwärtsintegration erfolgte.

Zeitumkehrinvarianz kann hier also nicht zu einer Beurteilung der Größe des Integrationszeitschritts herangezogen werden, sondern erlaubt einzig eine Aussage über numerische Fehler (Rundungsfehler, etc.) bei der Simulation.

3.1.1.2 Simulation

Für die Simulation fand eine bei 300 K equilibrierte Struktur von PTI (ohne Wasser) Verwendung (dieselbe Struktur wurde in [69] verwendet). Der Integrationszeitschritt betrug 1 fs. Entgegen der üblichen Vorgehensweise wählten wir die anfänglichen Geschwindigkeiten nicht nach einer Maxwellverteilung, sondern setzten sie alle auf Null. Da zwischen kinetischer Energie

$$E_{kin} = \frac{1}{2} \sum_{i=1}^{N} \langle v_i^2 \rangle m_i \tag{3.12}$$

und Temperatur

$$T = \frac{1}{3k_B N} \sum_{i=1}^{N} \langle v_i^2 \rangle m_i \tag{3.13}$$

über den Gleichverteilungssatz (2.11) eine direkte Proportionalität besteht

$$T = \frac{2}{3k_B N} E_{kin} \quad , \tag{3.14}$$

betrug zu Beginn der Simulation die Temperatur 0 K, um sich dann schnell einem Wert zu nähern, der bei etwa 260 K lag (vgl. Abbildung 3.2).

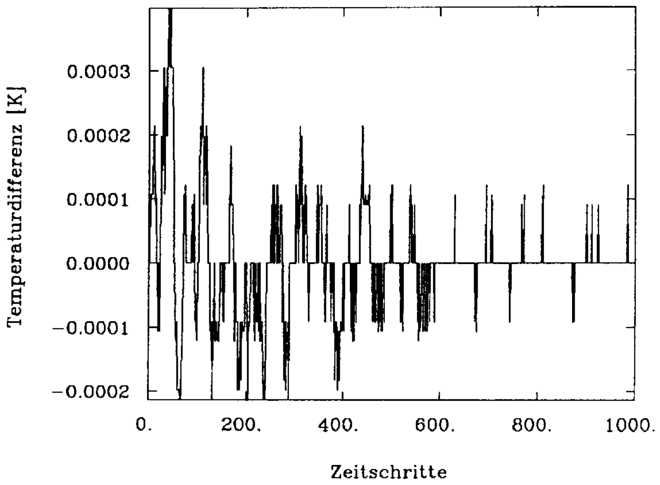


Abbildung 3.1: Gerechnet wurden 1000 Zeitschritte Dynamik (Zeitschritte aufgetragen auf der x-Achse) von PTI. Durch Zeitumkehr wurde die Trajektorie anschließend rückwärts durchlaufen. Aufgetragen sind die Temperaturdifferenzen einander entsprechender Punkte des Hin- und Rücklaufs. Die entsprechenden Absolutwerte finden sich in Abbildung 3.2 (Zeitschritt: 1 fs). Die beobachteten Differenzen haben ihren Ursprung in numerischen Ungenauigkeiten (Rundungsfehlern) bei der Berechnung.

Zum Test der Zeitumkehrinvarianz wurden 1000 Zeitschritte (1 ps) Dynamik gerechnet („Hinlauf"), dann wurden alte und aktuelle Orte vertauscht (Zeitumkehr) und erneut wurden 1000 Zeitschritte Dynamik gerechnet („Rücklauf"). Um einen besseren Vergleich der Temperaturwerte des Hin- und des Rücklaufs zu ermöglichen, ist in Abbildung 3.1 die Temperaturdifferenz einander entsprechender Zeitpunkte des Hin- und des Rücklaufs aufgetragen. Sie beträgt nur etwa ein Millionstel des Absolutwertes der Temperatur (Abb. 3.2).

Man erkennt außerdem aus Abb. 3.1, daß die Temperaturdifferenzen zu etwa gleichen Teilen positiv wie negativ sind und zum Zeitschritt Null hin zunehmen. Dies ist nicht verwunderlich, da auch die durchlaufene Trajektorie länger wird: die Temperaturwerte zum Zeitschritt 900 sind durch 200 Integrationsschritte getrennt (100 in Richtung positiver Zeit und 100 in Richtung negativer Zeit). Die Temperaturwerte zum Zeitschritt 0 dagegen sind durch 2000 Integrationsschritte getrennt; die numerischen Fehler in einem Schritt können sich hier zu wesentlich größeren Gesamtfehlern akkumulieren. Da die Temperaturbestimmung eine Mittelwertbildung beinhaltet (siehe Gleichung 3.13), bedeutet eine Temperaturdifferenz von 0 K in Abbildung 3.1 nicht notwendigerweise auch die Übereinstimmung der Orte und Geschwindigkeiten aller Atome im Molekül. Daher akkumulieren sich Rundungsfehler auch über die Nulldurchgänge der Kurve in Abbildung 3.1 hinaus. Diese Zunahme der numerischen Abweichungen hat ganz allgemein

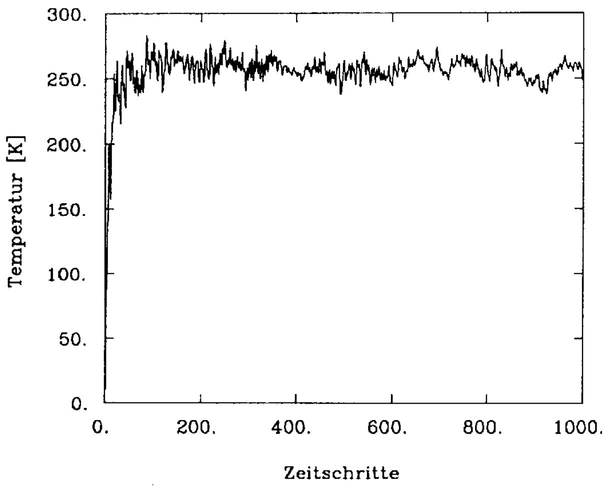


Abbildung 3.2: Absolutwerte der Temperatur (3.13) von PTI während 1000 Integrationsschritten. Der Zeitschritt betrug 1 fs.

exponentiellen Charakter.

Zusammenfassend läßt sich aus Abbildung 3.1 folgern, daß die Rundungsfehler bei Molekulardynamiksimulationen ohne Bedenken vernachlässigt werden können, zumal das verwendete Modell von sich klassisch in einer Energielandschaft bewegenden Atomkernen die Wirklichkeit nur näherungsweise beschreibt.

3.1.2 Energieerhaltung

Als weiterer Test wurde die Energieerhaltung während einer Langzeitsimulation von PTI untersucht (Abbildung 3.3). Um einen direkten Vergleich mit [69, Seite 34] zu ermöglichen, wurde derselbe Energiemaßstab wie dort gewählt. Allerdings erstreckt sich die Simulation über einen etwa 40 mal so langen Zeitraum. Wegen der großen Datenmenge mußte für die Darstellung über jeweils 10 Werte gemittelt werden. Die Gesamtenergie bleibt über den gesamten Simulationszeitraum erhalten und auch die absoluten Energiewerte stimmen (bis auf die zuvor besprochene Absenkung der Temperatur auf 260 K) mit denen von [69] überein.

Diese Ergebnisse können als Bestätigung der korrekten Arbeitsweise des parallelen Programmes EGOgewertet werden. Weitergehende direkte numerische Vergleiche mit Resultaten von MD und XPLOR zeigten, daß die Rundungsfehler sehr klein sind.

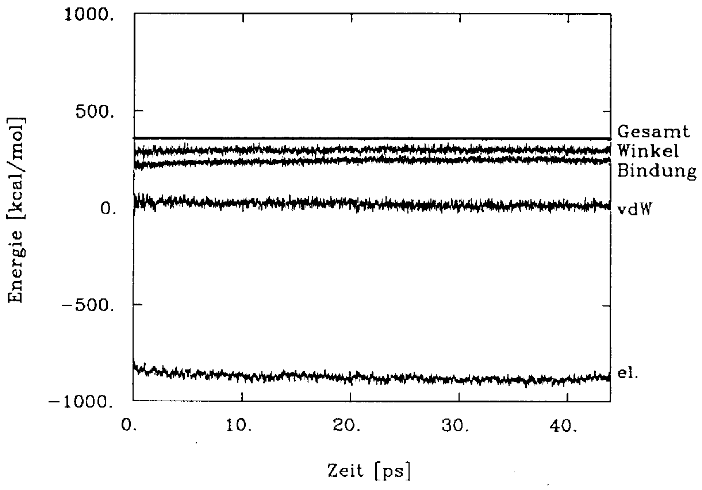


Abbildung 3.3: Langzeitdynamiksimulation von PTI, Zeitschritt: 1 fs. Dargestellt sind Gesamtenergie (Gesamt), Winkelenergie (Winkel), Bindungsenergie (Bindung), van der Waals-Energie (vdW) und elektrostatische Energie (el.).

3.2 Rechnungen am Reaktionszentrumvon *Rhodopseudomonas viridis*

Die folgenden Resultate wurden durch Simulationen des *gesamten* Reaktionszentrums von *Rhodopseudomonas viridis* mit 12 637 Atomen gewonnen. Alle Berechnungen wurden, soweit nicht anders angegeben, mit dem Programm EGOauf dem Rechner ALTER EGO–12 durchgeführt. Dabei kam kein cut-off (⇒G) zur Anwendung; stets wurden *alle* Wechselwirkungen berücksichtigt.

3.2.1 Zeitumkehrinvarianz

Die Zeitumkehrinvarianz (Abschnitt 3.1.1.1) wurde in einem kürzeren zeitlichen Rahmen als bei PTI auch beim Reaktionszentrumgetestet. Abbildungen 3.5 und 3.4 zeigen einen Ausschnitt aus einer Dynamiksimulation kurz nach einer Phase sehr starker Abkühlung, ausgehend von der Röntgenstruktur. Die entsprechend weit vom thermischen Gleichgewicht entfernte Verteilung der Energie auf die Freiheitsgrade erklärt die ungewöhnlich großen Schwankungen der betrachteten Observablen. Solche Schwankungen sind für gewöhnlich unerwünscht. In diesem speziellen Fall dagegen sind große Schwankungen von Vorteil, da man die Symmetrie bei Zeitumkehr besser beobachten kann. Der Zeitschritt beträgt 0.5 fs. Zwischen Integrationsschritt 629 und 630 (gestrichelte Linie) findet, wie oben beschrieben, die Zeitumkehr statt.

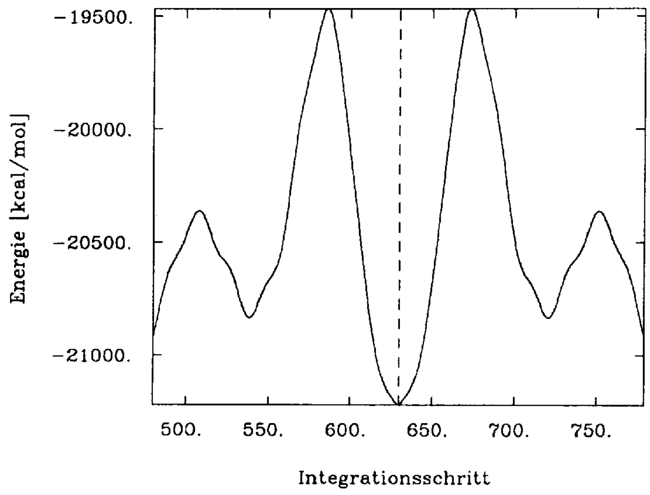


Abbildung 3.4: Verhalten der elektrostatischen Energie bei Zeitumkehr zwischen Integrationsschritt 629 und 630.

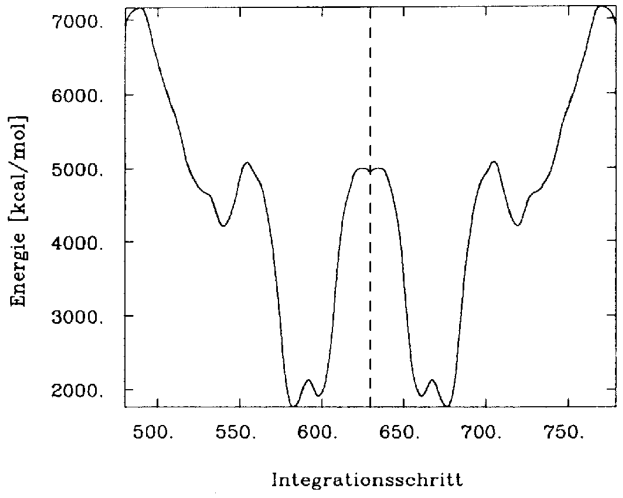


Abbildung 3.5: Verhalten der Bindungsenergie bei Zeitumkehr zwischen Integrationsschritt 629 und 630.

In Abbildung 3.4 ist die elektrostatische Energie über den Integrationsschritten aufgetragen, in Abbildung 3.5 die Bindungsenergie. Man erkennt sehr gut die vollständige Spiegelsymmetrie, was den hohen Grad an numerischer Genauigkeit belegt.

3.2.2 Minimierung des Reaktionszentrums

Mit der Aufklärung der Struktur des Reaktionszentrums durch Deisenhofer und Michel [26] wurde die Grundlage für weitere Untersuchungen, insbesondere Molekulardynamiksimulationen, geschaffen. Schon im Abschnitt 2.4.3 wurde darauf hingewiesen, daß in dieser Struktur noch unnatürliche Verspannungen enthalten sind, die vor Beginn einer Molekulardynamiksimulation entfernt werden müssen, da sonst die Ergebnisse der Molekulardynamiksimulation verfälscht werden. Daher wurde zunächst eine Minimierung durchgeführt. Die Geschwindigkeiten wurden zu Beginn alle auf Null gesetzt.

Ausgehend von der Röntgenstruktur wurde also eine Molekulardynamiksimulation mit mikroskopischer Reibung durchgeführt (*simulated annealing*). Bei jedem Integrationsschritt werden die drei Geschwindigkeitskomponenten jedes Atoms mit einem konstanten Faktor, dem Reibungskoeffizienten, der zwischen 0 (keine Bewegung möglich) und 1 (keine Reibung) liegt, multipliziert. Auf diese Weise verliert das Molekül ständig kinetische Energie. Dieser Verlust wird zum Teil durch ein Nachströmen von Energie aus anderen Freiheitsgraden wieder ausgeglichen und teilt sich dadurch allen Freiheitsgraden mit.

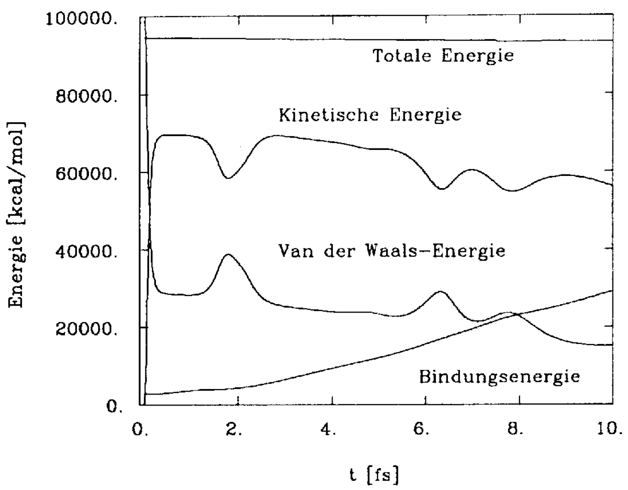


Abbildung 3.6: Minimierung der Röntgenstruktur. Gezeigt ist ein stark gedehnter Ausschnitt aus dem Verlauf von van der Waals-, Bindungs-, kinetischer und totaler Energie zu Beginn der Simulation. Der Zeitschritt beträgt nur 0.01 fs. Der Reibungsfaktor hat einen Wert von 0.99999/Integrationsschritt. Um die Energiewerte zu Beginn der Simulation deutlicher erkennen zu können, ist die Zeitskala leicht nach rechts verschoben.

Die Temperaturen, bei denen die Minimierung stattfindet, können zu Beginn sehr hoch werden, da die Verspannungen, welche sich in hohen Werten der van der Waals-Energie äußern, schnell auch zu hohen Werten der kinetischen Energie (und damit der Temperatur) führen (siehe Abbildung 3.6; zu Beginn der Simulation liegt die van der Waals-Energie bei etwa 100 000 kcal/mol.). Diese hohen Werte, zusammen mit Fluktuationen der kinetischen Energie, erlauben das Überwinden von Potentialbarrieren in der Energielandschaft. Dadurch wird es möglich, tieferliegende Minima der potentiellen

Energie zu finden, als dies mit Gradientenabstiegsmethoden möglich ist, da letztere keine Energiebarrieren überwinden können. Zwar hat das Auftreten von derart hohen Temperaturen auch den Nachteil, daß das Protein in Zustände übergehen kann, die in natürlicher Umgebung nicht zu erreichen sind, doch sind dem folgende Argumente entgegenzuhalten:

- der Zeitraum, über den das Protein derart hohen Temperaturen ausgesetzt ist, ist sehr kurz (siehe Abbildung 3.7), so daß nicht zu erwarten ist, daß hohe Barrieren

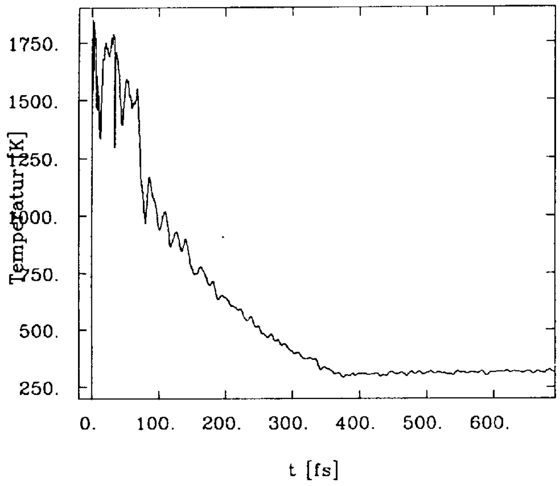


Abbildung 3.7: Verhalten der Temperatur über den gesamten Simulationszeitraum.

  überwunden werden.

- ein strukturelles Verbleiben nahe der Ausgangsstruktur ist nur dann wünschenswert, wenn diese die natürliche Struktur des Proteins gut widerspiegelt. Daß dies nicht unbedingt der Fall ist (Verspannungen), wurde schon erläutert.

- die gesamte Energie für das Aufheizen des Proteins stammt aus seiner potentiellen Energie. Falls sich die Ausgangsstruktur nur wenig von der natürlichen Struktur unterscheiden würde, so käme es im Verlauf einer Dynamiksimulation nicht zu hohen Temperaturen.

- eine Vergleichsrechnung von A. Windemuth mit dem Programm MD [69] (unter Verwendung von „partieller Elektrostatik“) über die ersten 150 fs zeigte ähnlich große Abweichungen von der Röntgenstruktur an denselben Stellen, an denen sie auch bei unserer Simulation auftraten. Bei dieser Vergleichsrechnung wurde, ausgehend von der Röntgenstruktur, sofort eine Equilibrierung (siehe Abschnitt 2.4.4) durchgeführt. Die Temperatur schwankte daher nur geringfügig um

300 K. Ein Übergang der Proteinstruktur in gewöhnlich unzugängliche Bereiche scheint daher ausgeschlossen.

Daher sind wir der Auffassung, daß die in dieser Arbeit verwendete Methode der Minimierung gerechtfertigt ist.

Bei der verwendeten Methode können, vor allem zu Beginn der Minimierung, einzelne Atome hohe Geschwindigkeiten erreichen. Damit das Integrationsverfahren nicht instabil wird, empfiehlt es sich daher, einen kleinen Integrationszeitschritt zu wählen. Wir benutzten während der ersten 7.5 fs einen Zeitschritt von nur 0.01 fs. Für einzelne, besonders schnelle Atome könnte auch dieser Zeitschritt noch zu groß sein. Daher wurde in das Programm noch ein weiterer Parameter eingeführt: die maximale Schrittweite (Verschiebung) je Integrationsschritt. Die Bedeutung dieses Parameters ist im folgenden beschrieben.

Abhängig von der Form der Potentialfunktion kann es vorkommen, daß ein Atom während eines Integrationsschritts Strecken zurücklegt, die es ihm erlauben, Potentialbarrieren ohne Energieaufwand zu überwinden. Abbildung 3.8 zeigt einen Potentialtopf mit sehr steilen Wänden. Ist die in einem Schritt zurückgelegte Strecke größer als die Wandzone des Potentials, so überwindet das Teilchen die Potentialwand, ohne sie zu bemerken. Damit geht lokal ein Gewinn an potentieller Energie ohne Verlust kinetischer Energie einher, was zum Verlust der Erhaltung der Gesamtenergie führt.

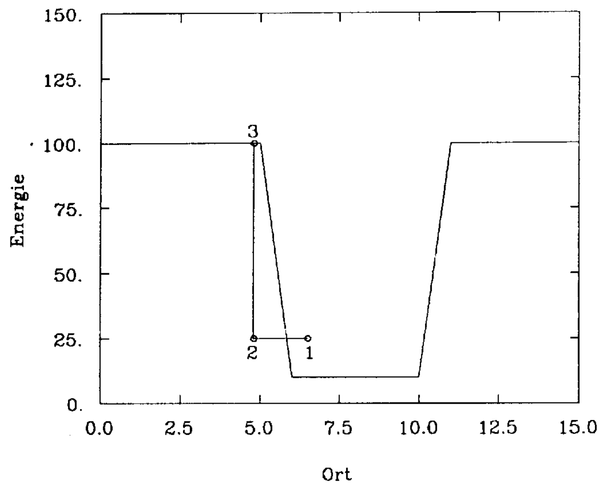


Abbildung 3.8: Beispielpotential zu den Erläuterungen im Text. Ort und Energie in beliebigen Einheiten.

Dies ist in Abbildung 3.8 illustriert: ein Teilchen bei Position 1, das in *einem* Integrationsschritt Position 2 erreicht, „spürt“ von der Potentialschwelle nichts; an keiner

Position wirkt eine rücktreibende Kraft, da bei allen Positionen die Steigung des Potentials verschwindet. Im darauffolgenden Berechnungszyklus (siehe Seite 54) wird die Energiefunktion erneut ausgewertet und dem Teilchen wird eine Energie gemäß Position 3 zugewiesen. Die entsprechende Energiezunahme drückt sich in einem Ansteigen der Gesamtenergie aus.

Um solche Effekte zu vermeiden, läßt sich im Programm EGOdie maximale Schrittweite pro Integrationsschritt begrenzen. Der Wert der Obergrenze hängt, wie erwähnt, von der Potentialform ab und wird in [69, Seite 43] mit 0.08 Å angegeben. Dieser Wert wurde auch in unseren Simulationen verwendet.

Der Reibungsfaktor wurde mit einem Wert von 0.99999/Integrationsschritt zu Beginn der Simulation sehr klein (d.h. sehr nahe an 1) gewählt, was man in Abbildung 3.6 an der geringen mittleren Steigung der Kurven erkennen kann. Es liegt damit ein nur schwach gedämpftes System vor.

Gut zu erkennen ist die Korrelation von kinetischer und van der Waals-Energie. Sie besteht aber nur zu Beginn der Simulation, da hier sehr große Energiebeträge von der van der Waals-Energie in die kinetische Energie übergehen und damit deren Verlauf bestimmen.

Im Gegensatz zu den bisher betrachteten Energiebeiträgen steigt die Bindungsenergie im Verlauf der Simulation an, da in der Röntgenstruktur alle Bindungslängen nahe dem Gleichgewichtsabstand gehalten sind, was nicht den natürlichen Verhältnissen entspricht.

3.2.2.1 Verlauf der Minimierung

Im Verlauf der Minimierung wurden die Parameter Reibungskonstante und Zeitschritt mehrfach geändert. In Abbildung 3.9 ist der Verlauf der Gesamtenergie während der gesamten bisher durchgeführten Simulation zu beobachten. Die erkennbaren Abschnitte (mit 1–6 numeriert) entsprechen jeweils verschiedenen Parameterwerten.

Abschnitt 1 ist in Abbildung 3.6 gedehnt dargestellt und wurde oben diskutiert. Die Minimierung wurde in dieser ersten, kritischen Phase sehr vorsichtig vorgenommen: der Zeitschritt betrug 0.01 fs und der bei jedem Integrationsschritt angewendete Reibungsfaktor hatte den Wert 0.99999/Integrationsschritt.

Nach 7.5 fs wurde der Zeitschritt von 0.01 fs auf 0.05 fs verfünffacht, da die größten Energieschwankungen schon stattgefunden hatten. Wie Abbildung 3.9 zeigt, nimmt die Gesamtenergie während Abschnitt 2 langsamer ab, als während Abschnitt 1, da die effektive Reibung pro Zeiteinheit um einen Faktor Fünf kleiner ist.

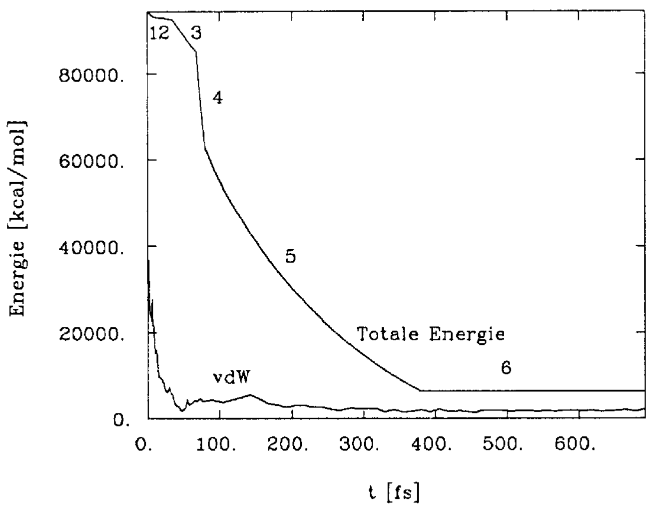


Abbildung 3.9: Dargestellt ist der zeitliche Verlauf der van der Waals- und der totalen Energie während der Simulation. Die Kurve der Gesamtenergie zerfällt deutlich in 6 verschiedene Abschnitte, die im Text erläutert werden.

In Abschnitt 3 (ab 35 fs) wurde die Reibung erhöht, d.h. der Reibungsfaktor pro Integrationsschritt wurde von 0.99999 auf 0.9999 verkleinert, um die Abkühlung zu beschleunigen. Dies drückt sich in der schnelleren Abnahme der Gesamtenergie aus. Der Zeitschritt betrug unverändert 0.05 fs.

Aus demselben Grund wurde der Reibungsfaktor in Abschnitt 4 (ab 67.5 fs) auf 0.999/Integrationsschritt erhöht. Der Zeitschritt betrug 0.05 fs.

Vom Beginn des Abschnitts 5 an ist zu erkennen, daß die van der Waals-Energie (bis auf Fluktuationen) einen konstanten Wert erreicht hat. Deshalb wurde nun der Integrationsschritt auf 0.25 fs erhöht. Wegen der entsprechend kleineren effektiven Reibung flacht die Kurve hier erneut ab. Mit diesen Parameterwerten wurden die nächsten 312.75 fs gerechnet. Man erkennt in diesem Teil der Kurve den exponentiellen Abfall der Energie, der aus der konstanten Reibung resultiert.

In Abschnitt 6 (ab 380.25 fs) wurde die Reibung abgeschaltet, da die Zieltemperatur von 300 K erreicht war. Unter Beibehaltung des Integrationsschritts wurde Dynamik gerechnet, um die Relaxation aller Freiheitsgrade zu überprüfen. Die Gesamtenergie ändert sich in diesem Abschnitt nicht mehr.

3.2.3 Verhalten des Moleküls nach der Minimierung

Wie schon im Abschnitt 2.4.4 erwähnt, sind die einzelnen internen Freiheitsgrade des Moleküls nach einer Minimierung nicht unbedingt im Gleichgewicht. Um dies beim

Reaktionszentrumzu überprüfen, wurde im Anschluß an die Minimierung eine Dynamiksimulation ohne Reibung durchgeführt.

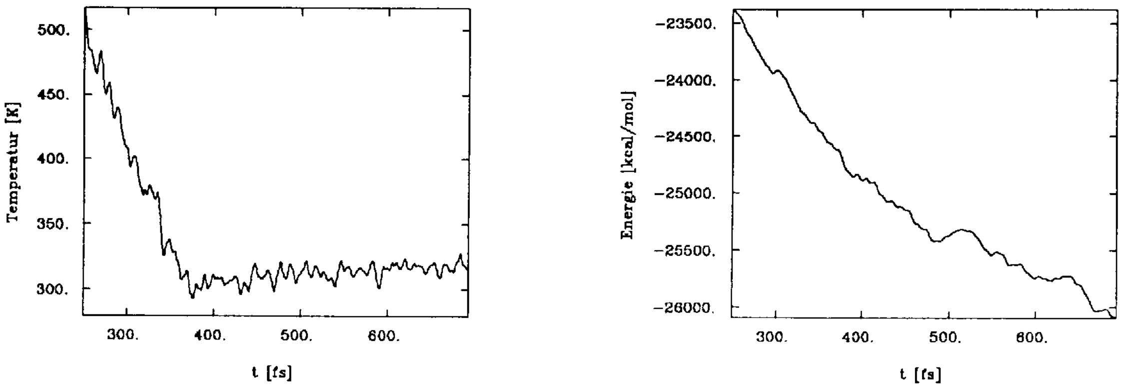


Abbildung 3.10: Temperatur (siehe Gleichung (3.13)) und elektrostatische Energie nach der Minimierung.

Die Ergebnisse dieser Simulation sind in Abbildung 3.10 dargestellt. Mit Abschalten der Reibung bei 380.25 fs ist auch das Absinken der Temperatur beendet; die Temperatur steigt sogar von 300 K wieder auf etwa 320 K an. Die dazu erforderliche Energie stammt aus Freiheitsgraden der potentiellen Energie, die noch nicht im Gleichgewicht sind. Dazu zählt vor allem die elektrostatische Energie, wie man aus dem rechten Bild in Abbildung 3.10 ersehen kann: trotz Abschalten der Reibung sinkt die elektrostatische Energie nahezu unbeeinflußt weiter. Dieser Energiebeitrag ist es, der die Erhöhung der kinetischen Energie bewirkt.

Aus dieser Untersuchung läßt sich folgern, daß noch nicht alle Freiheitsgrade im Molekül den Gleichgewichtszustand erreicht haben. Daher muß sich an die Minimierung noch eine Equilibrierung mit festgehaltener Temperatur anschließen, die genügend lange dauert, um auch der elektrostatischen Energie Gelegenheit zu geben, einen Gleichgewichtswert zu erreichen. Die Durchführung dieser Equilibrierung ist z.Zt. noch nicht abgeschlossen.

3.2.4 Geschwindigkeitsverteilung

Im Abschnitt 2.2 wurde schon darauf hingewiesen, daß neben einem Satz von Anfangskoordinaten auch ein Satz von Anfangsgeschwindigkeiten benötigt wird. Da für letztere keine experimentellen Daten verfügbar sind, legt man für die Geschwindigkeiten meist eine Maxwellverteilung [siehe Gleichung (3.15)], die der Zieltemperatur entspricht, zugrunde. Wir sind aus zwei Gründen davon abgegangen und haben die Anfangsgeschwindigkeiten zu Null gewählt:

1. Die Zieltemperatur (300 K) ist wesentlich niedriger als die sich nach wenigen Integrationsschritten einstellende Temperatur ($\approx$ 1750 K). Daher sollte die Verteilung der Anfangsgeschwindigkeiten von untergeordneter Bedeutung sein.

2. Falls als Anfangsgeschwindigkeit Null gewählt wird, schwingen zu Beginn sehr viele Atome in Phase. Dies kann, wie gezeigt werden wird, auch an den Mittelwerten von Temperatur und Bindungsenergie beobachtet werden (siehe Abschnitt 3.2.5). Durch die von Atom zu Atom unterschiedliche Kopplung der verschiedenen Freiheitsgrade geraten die Atome dann mehr und mehr außer Phase und es sollte sich ganz automatisch die „richtige“ Geschwindigkeitsverteilung einstellen.

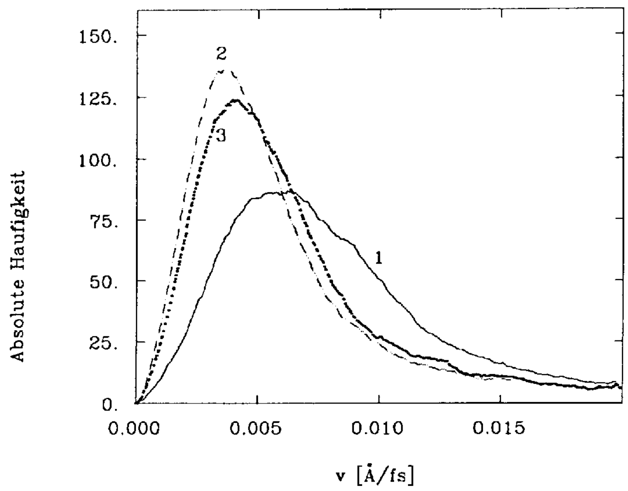


Abbildung 3.11: Zeitliche Entwicklung der Geschwindigkeitsverteilung. Kurve 1 wurde bei 105 fs, Kurve 2 bei 380 fs und Kurve 3 bei 642.5 fs aufgenommen. Nach oben sind die absoluten Häufigkeiten der Geschwindigkeiten je Geschwindigkeitsintervall (66 Å/ns) angegeben. Die Kurven wurden durch gleitende Mittelwertbildung über je 20 Datenpunkte geglättet.

Um die sich einstellende Geschwindigkeitsverteilung zu untersuchen, wurden zu drei verschiedenen Zeitpunkten (105 fs, 380 fs, 642.5 fs) Geschwindigkeitsprofile aufgenommen. Die Kurven sind in Abbildung 3.11 dargestellt.

Kurve 1 zeigt die Verteilung, nachdem große Energiebeträge aus der van der Waals-Energie in kinetische Energie übergegangen sind. Man erkennt neben einer Verschiebung des Maximums zu hohen Werten auch eine Verbreiterung der Kurve gegenüber den anderen beiden Kurven. Da die Verspannungen örtlich lokalisiert sind, gibt es zu Beginn der Simulation einige sehr schnelle Atome, die einen Teil ihrer Energie aber (durch Kopplung über die verschiedenen Potentiale, hauptsächlich über Bindungen) rasch an andere Atome abgeben, die dadurch ebenfalls hohe Geschwindigkeiten erhalten.

Kurve 2 wurde nach Abschluß der Abkühlphase aufgezeichnet. Der Entzug an kinetischer Energie und damit das Absinken der Temperatur bewirkt eine Verschiebung des Maximums in Richtung kleinerer Geschwindigkeitswerte. Auch ist die Verteilung nun schmaler, da sich die Geschwindigkeiten einander angeglichen haben.

Kurve 3 wurde nach Ablauf der Dynamiksimulation (Abschnitt 6 der Kurve der Gesamtenergie in Abbildung 3.9) aufgezeichnet. Die Unterschiede zu Kurve 2 sind gering. Die leichte Verschiebung des Maximums spiegelt die Erhöhung der Temperatur wider (siehe Abschnitt 3.2.3).

Wir wollen nun die am Ende der Simulation erhaltene Verteilung mit einer Maxwellverteilung vergleichen.

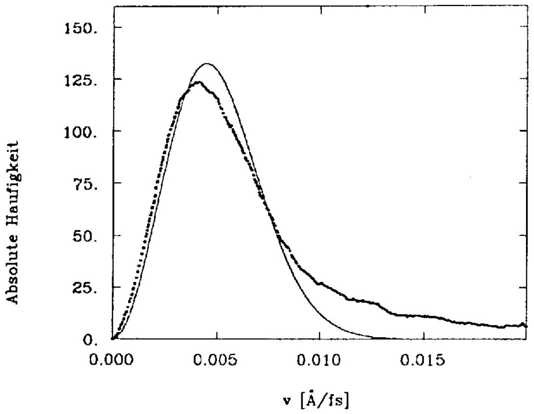


Abbildung 3.12: Die gepunktete Kurve ist identisch zu Kurve 3 in Abbildung 3.11. Die durchgezogene Kurve stellt einen numerischen fit einer Maxwellverteilung dar.

In Abbildung 3.12 ist dieser Vergleich durchgeführt. Dazu wurde eine Maxwellverteilung [57, S. 314]

$$F(v) = av^2 \mathrm{e}^{-bv^2}; \quad a = 4\pi N \left(\frac{m}{2\pi k_B T}\right)^{3/2}; \quad b = \frac{m}{2k_B T} \tag{3.15}$$

durch Variation der Parameter a und b numerisch an die Daten der Kurve 3 aus Abbildung 3.11 angepaßt. Die Hauptunterschiede zwischen beiden Kurven liegen im Bereich der hohen Geschwindigkeiten. Hier liegt die durch Simulation bestimmte Geschwindigkeitsverteilung deutlich über der Maxwellverteilung. Trotzdem kann die Übereinstimmung als gut angesehen werden; sie rechtfertigt nachträglich die Vorgehensweise, alle Anfangsgeschwindigkeiten auf Null zu setzen.

Eine vollständige Übereinstimmung der Geschwindigkeitsverteilung mit einer Maxwellverteilung ist nicht zu erwarten, weil

- die Atome im Molekül nicht frei sind; sie werden durch starke Bindungen an ihre Nachbarn beeinflußt;
- die Atome des Moleküls unterschiedliche Masse haben; daher stellen die Geschwindigkeitsverteilungen in Abbildung 3.11 und 3.12 Überlagerungen Maxwellscher Verteilungen zu Atomen verschiedener Masse dar;
- die simulierten Zeiten noch sehr kurz sind; zu späteren Zeitpunkten kann die Verteilung der kinetischen Energie gleichmäßiger sein und die Zahl der Atome mit hoher Geschwindigkeit kleiner werden.

3.2.5 Bindungsschwingungen

Da als Anfangsbedingung für die Geschwindigkeiten der Wert Null gewählt wurde, beginnen alle Atome mit ihrer Schwingung in Phase. Zwar schwingen die Atome je nach Masse, Kraftkonstante und molekularer Umgebung mit verschiedenen Frequenzen, doch lassen sich stets ganze Gruppen von Atomen mit ähnlichen Frequenzen finden. Dieses kollektive Verhalten drückt sich in periodischen Schwankungen makroskopischer Parameter wie Bindungsenergie und Temperatur aus. Im Verlauf der Simulation geraten die Atome mehr und mehr außer Phase, was sich in einem Rückgang der Amplitude der makroskopischen Schwankungen zeigt.

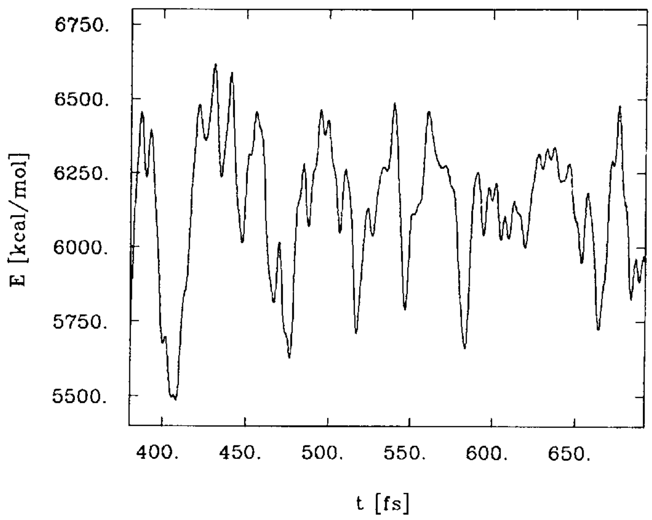


Abbildung 3.13: In y-Richtung stark gedehnter Ausschnitt der Bindungsenergie.

Die Schwankungen der Bindungsenergie sind in Abbildung 3.13 durch starke Dehnung der Energieskala sichtbar gemacht. Um eine Zuordnung zu Frequenzen zu ermöglichen, wurde eine Fouriertransformation ($\sin^2 + \cos^2$) durchgeführt. Das erhaltene Spektrum zeigt die durchgezogene Kurve in Abbildung 3.14.

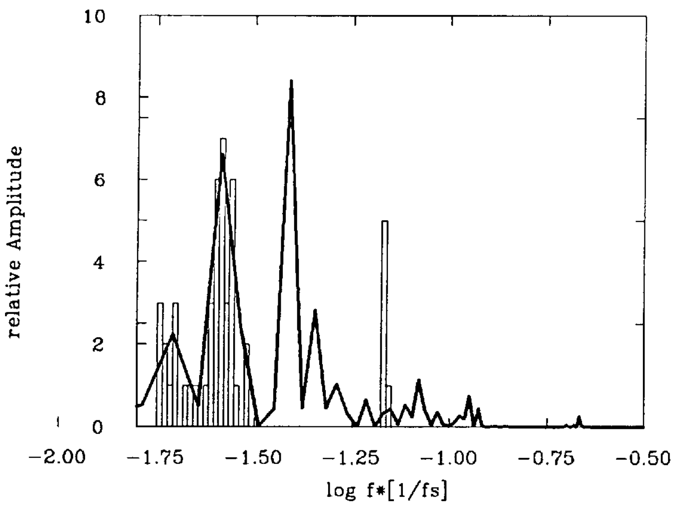


Abbildung 3.14: Gegenüberstellung des Fourierspektrums der Bindungsenergie (Zeitintervall: 380.25 fs bis 693.0 fs, Abschnitt 6 der Kurve in Abbildung 3.9) und eines Histogramms möglicher Bindungsschwingungsfrequenzen.

Es wurde noch eine Zuordung der beobachteten Frequenzen zu einzelnen Bindungsschwingungsfrequenzen versucht. Dazu wurde aus den Kraftkonstanten K_b und der Liste aller Bindungen im Reaktionszentrumeine Tabelle möglicher Bindungsschwingungsfrequenzen f angefertigt.

$$f = \frac{1}{2\pi}\sqrt{\frac{K_b}{m^*}} \tag{3.16}$$

Es ist dabei die reduzierte Masse $m^* = (m_1 \cdot m_2)/(m_1 + m_2)$ der an der Bindung beteiligten Atome zu verwenden. Aus den Daten dieser Tabelle wurde dann das in Abbildung 3.14 unterlegte Histogramm erzeugt. Für eine genauere Analyse müßten die einzelnen Frequenzen noch mit der Anzahl der Bindungen des jeweiligen Typs gewichtet werden, doch ging es hier nur darum, zu zeigen, daß ein Teil der beobachteten Frequenzen im Prinzip Bindungsschwingungen zugeordnet werden kann. Für den peak bei $10^{-1.4}$ fs^{-1} findet sich keine Entsprechung im Histogramm. Als mögliche Ursache kommt eine Frequenzverschiebung der Bindungsschwingungen durch Einflüsse der molekularen Umgebung in Frage.

Kapitel 4

Der Rechner und das Programm

In diesem Kapitel sind der von uns entwickelte und gebaute Rechner ALTER EGO–12 und das ebenfalls neu entwickelte Parallelprogramm EGObeschrieben. Erst diese beiden Komponenten ermöglichten es, die im Kapitel 3 aufgeführten Berechnungen ohne cut-off (⇒G) durchzuführen. Übliche Programme zur Simulation der Dynamik von Makromolekülen wie z.B. XPLOR, sind bei Molekülen von der Größe des Reaktionszentrums nur durch Modifikationen in der Lage, ohne cut-off zu arbeiten. Doch der Zeitaufwand solcher Rechnungen und die damit verbundenen hohen Kosten lassen ihre Durchführung mit herkömmlicher Rechnerausrüstung nicht sinnvoll erscheinen.

Seiner Natur nach muß dieses Kapitel sehr technisch sein und dafür möchte der Autor schon an dieser Stelle um Verständnis bitten. Aufbau und Arbeitsweise von Rechner und Programm werden ausführlich beschrieben, um dem interessierten Leser die Möglichkeit zu geben, alle rechnerinternen Vorgänge, die während einer Molekulardynamiksimulation ablaufen, verstehen zu können. Wie schon in der Einleitung erwähnt, mußte der Großteil der für diese Diplomarbeit zur Verfügung stehenden Zeit auf die in diesem Kapitel beschriebenen Arbeiten verwendet werden. Auch aus diesem Grund scheint eine detaillierte Beschreibung geboten. Eine vereinfachte und gekürzte Darstellung dieses Kapitels findet sich in [37]. Eine englischsprachige Version dieses Kapitels wurde bei INMOS zur Veröffentlichung in den *technical notes* eingereicht. Eine englische Kurzfassung dieser Arbeit wurde zur Veröffentlichung in *molecular simulation* eingereicht.

4.1 Die Eigenentwicklung — Anforderungen und Leistungen

Vor der Konzeption eines neuen „Gespanns“ aus Rechner und Programm sind die zu erfüllenden Anforderungen klarzulegen („Pflichtenheft“). Diese lauten bei uns wie folgt:

1. Parallelrechnerkonzept
2. Leichte Erweiterbarkeit bzw. stufenweiser Ausbau
3. Sehr leistungsfähige einzelne Knoten, um mit Supercomputern konkurrieren zu können:
   - Großer Hauptspeicher; mindestens 1 MByte/Knoten
   - Gewährleistung der Datensicherheit
   - Schneller Mikroprozessor
   - Einfache Vernetzbarkeit
   - Hohe Rechenleistung auch bei floating point-Operationen
4. Optische Anzeige der Prozessorauslastung
5. Hohe Rechenleistung pro Volumen
6. Geringe Kosten
7. Flexible Topologie (⇒G)

Die von uns vorgeschlagene Lösung, welche alle diese Forderungen erfüllt, läßt sich folgendermaßen zusammenfassen:

- Transputerparallelrechner mit dem chip IMS T800 als Prozessor
- 1 MByte dynamisches RAM (⇒G) als Hauptspeicher pro Knoten
- Überprüfung der Daten aus dem Speicher durch einen parity-Mechanismus (⇒G)
- Leuchtdiodenanzeige des Betriebszustands jedes Knotens
- Sechs Rechenknoten auf einer Doppeleuropakarte, 10 Karten in einem 19-Einschub

- Drei-Lagen multilayer-Platine mit einer Leiterbahnbreite und einem Leiterbahnabstand von nur 0.2 mm
- Herstellung im Eigenbau zu Selbstkosten
- Austauschbare backplane (⇒G) (ermöglicht das Verändern der Topologie)

Transputer (⇒G) als Rechenelemente wurden gewählt, um die Anforderungen 1. und 3. zu erfüllen; der T800 ist zur Zeit einer der schnellsten Mikroprozessoren auf dem Markt. Außerdem gewährleistet das Transputerkonzept eine leichte Vernetz- und Erweiterbarkeit (Anforderung 2.). Anforderung 7. wurde nicht so stark berücksichtigt, weil sie für die primär beabsichtigte Anwendung des Rechners ohne Bedeutung ist. Sie ist als Option für die Verwendung des Rechners bei anderen Aufgaben gedacht. Dazu kann unsere festverdrahtete backplane (⇒G) durch eine, mittels link switches, elektronisch umschaltbare ersetzt werden (siehe Abschnitt 4.3.3.2).

4.2 Problem und Topologie (⇒G)

Ausgehend vom Problem, große Biopolymere zu simulieren, gilt es, eine geeignete Topologie (⇒G) zu entwickeln. Um die Bewegung der einzelnen Atome berechnen zu können, ist es notwendig, die Kraft zu kennen, die auf jedes einzelne Atom wirkt. Diese setzt sich aus den in Abschnitt 2.1 genannten Anteilen zusammen: Elektrostatische Kräfte, van der Waals-Kräfte, Bindungskräfte, Winkelbiegungskräfte, Dihedralkräfte, Extraplanarkräfte, Kräfte von Wasserstoffbrücken und Zwangskräfte.

Die elektrostatischen Kräfte wirken zwischen zwei Atomen und sind abhängig von deren Ladungen q_i und q_j sowie ihrem Abstand $\vec{r}_{ij}$

$$\vec{F}_{ij} = \frac{q_i q_j}{4\pi\varepsilon r_{ij}^3} \cdot \vec{r}_{ij} \; ; \quad \vec{r}_{ij} = \vec{r}_i - \vec{r}_j \; ; \quad r_{ij} = |\vec{r}_{ij}| \quad . \tag{4.1}$$

Die Kraft, die auf Atom j wirkt, ist der Kraft, die auf Atom i wirkt, entgegengesetzt gleich, d.h. es gilt das Newtonsche Prinzip ACTIO = REACTIO

$$\vec{F}_{ji} = -\vec{F}_{ij} \quad . \tag{4.2}$$

Da an jedem Atom die Coulombkräfte **aller** anderen Atome angreifen, ist es notwendig, $\vec{F}_{ij}$ bzw. $\vec{F}_{ji}$ für alle Atom**paare** auszuwerten und zur Gesamtkraft $\vec{F}_i$, die auf Atom i wirkt, aufzuaddieren:

$$\vec{F}_i = \sum_{\langle ij \rangle,\ i \neq j} \vec{F}_{ij} \quad . \tag{4.3}$$

Unter Berücksichtigung von Gleichung (4.2) sind für N Atome $N(N-1)/2$ Paarkräfte $\vec{F}_{ij}$ zu bestimmen. Bei einer Molekülgröße von 12 637 Atomen, einer Zahl, die dem photosynthetischen Reaktionszentrum entspricht, sind die Kräfte von 79 840 566 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 und die, vom Standpunkt der Berechnung her, gleich zu behandelnde van der Waals-Wechselwirkung konzentrieren[1]. Der Rechenaufwand für alle anderen Wechselwirkungen wächst nur linear mit der Zahl der Atome und ist daher für solch große Atomzahlen zu vernachlässigen (siehe Tabelle 4.5).

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. Ein Schlüsselproblem bei der parallelen Auswertung der Coulombkräfte ist, daß eigentlich jeder Rechenknoten die Koordinaten aller Atome kennen muß, weil jedes Atom mit allen anderen in Wechselwirkung steht, ein einzelner Knoten aber nicht genug Speicherplatz besitzt, um alle Koordinaten abzuspeichern.

Es bieten sich verschiedene Strategien zur Parallelisierung an. Naheliegend wäre etwa eine Aufteilung der Arbeit auf die einzelnen Prozessoren, wie sie in Abbildung 4.1 illustriert ist: jedem Prozessor wird ein Raumbereich (z.B. Würfel*Wuerfeltopologie*) mit den darin eingeschlossenen Atomen zur Bearbeitung zugeteilt. Diese Vorgehensweise hat jedoch gravierende Nachteile:

- Jeder der von uns verwendeten Transputer [41] besitzt nur 4 links zur Kommunikation mit anderen Transputern (⇒G). Da an jeden Würfel sechs weitere Würfel angrenzen, mit denen Daten ausgetauscht werden müssen, wären zum Aufbau dieser Topologie (⇒G) sechs links nötig[2].
- 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 brächte.
- 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 beteiligten Prozessoren.
- Da die Kommunikation unkoordiniert erfolgt und die Daten auf dem Weg zum Zielknoten mitunter viele Transputer (⇒G) zu passieren haben, kann es an einzel-

[1]Beide Wechselwirkungen erfordern bei der Berechnung eine Summation über (dieselben) Atompaare. Einzig die funktionale Form der Summanden ist unterschiedlich. Siehe hierzu Abschnitt 2.1

[2]Zukünftige Transputertypen sollen über sechs oder sieben links verfügen, so daß eine entsprechende Rechnertopologie leicht zu realisieren wäre.

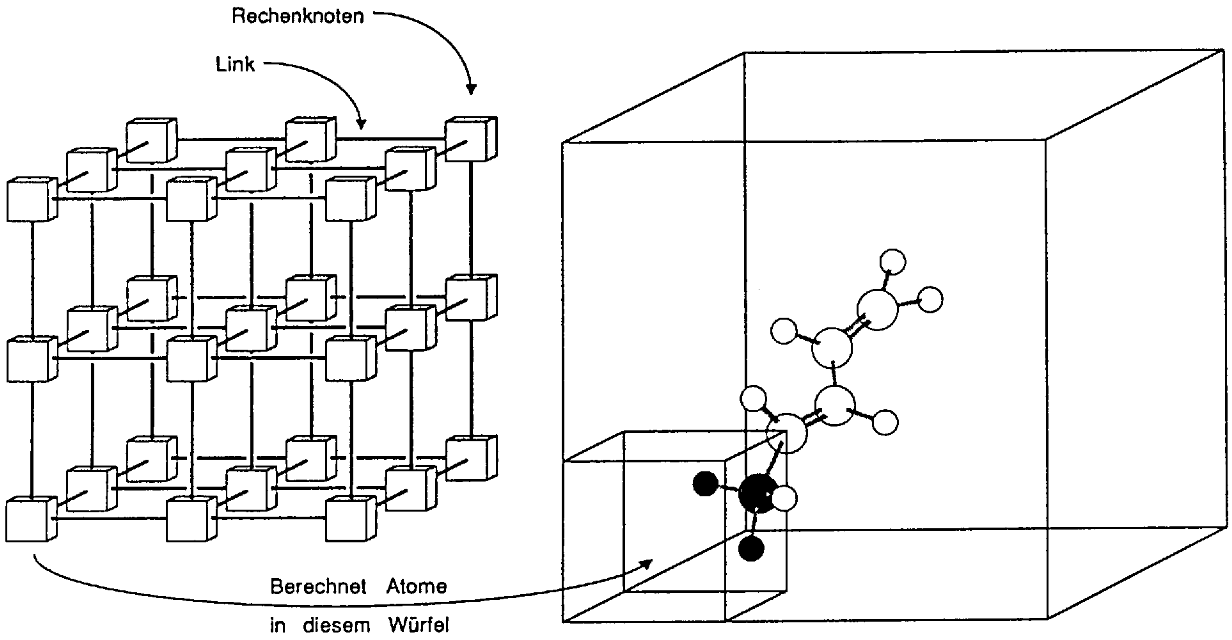


Abbildung 4.1: Würfel-Topologie.

nen Knoten zu Stauungen kommen. Dadurch werden andere Transputer, die auf die Daten warten müssen, in ihrer Arbeit behindert. So sinkt die Gesamtrechenleistung des Netzwerkes, was sich in einer geringeren Effizienz (⇒G) ausdrückt.

Für Probleme mit kurzreichweitigen Kräften dagegen ist die Topologie (⇒G) in Abbildung 4.1 hervorragend geeignet, da dort die Kommunikationswege sehr kurz sind.

Die aufgezählten Probleme vermeidet eine andere Topologie, die in Abbildung 4.2 vorgestellt wird und die wir daher verwenden. In dieser Topologie sind die Transputer zu einem Ring verbunden (diese Anordnung wurde auch schon in [43, 52] vorgeschlagen). 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 ihm „fremde“ Atome bezeichnet. Um sich gegenseitig über die Koordinaten der jeweils „fremden“ Atome zu informieren, bringt jeder Transputer Kopien seiner „eigenen“ Datensätze im Ring in Umlauf. Dies ist in Abschnitt 4.4.3.4 näher beschrieben.

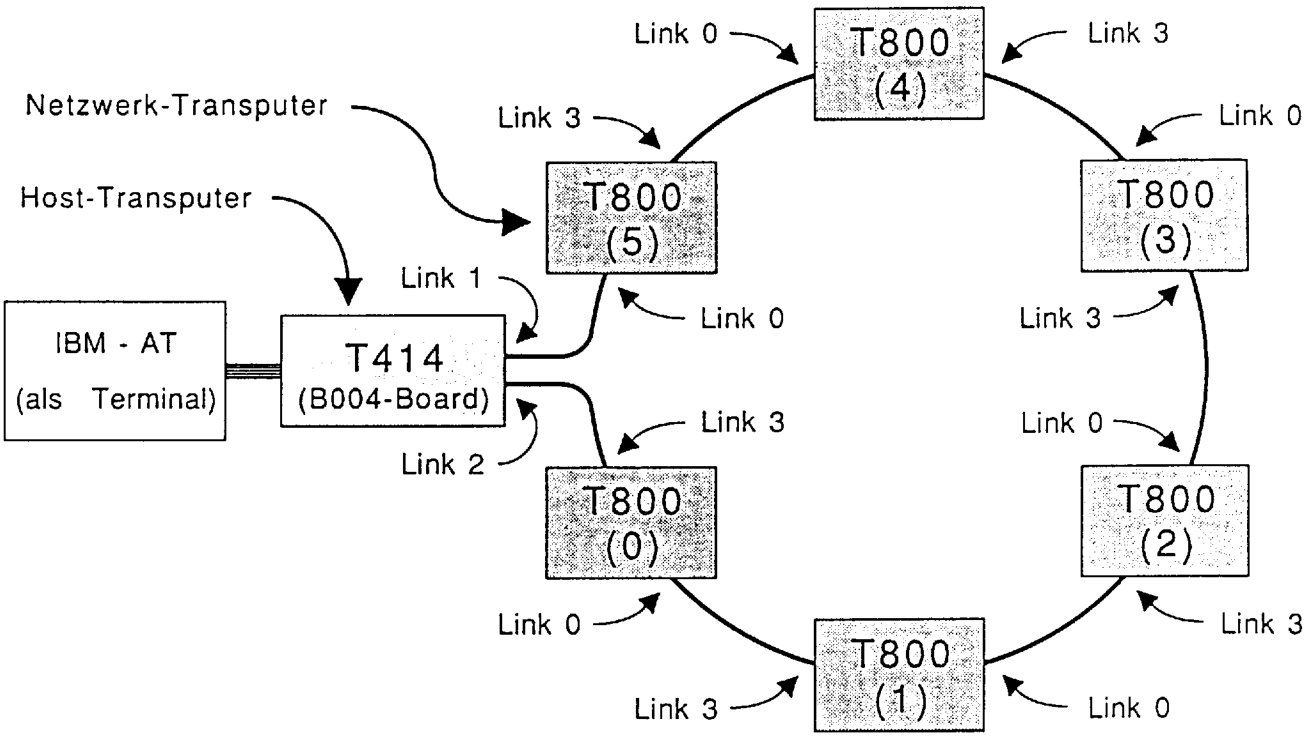


Abbildung 4.2: Ring-Topologie.

4.3 Hardware

Bevor wir uns näher mit dem Programm und seiner Lösungsstrategie für das Schlüsselproblem (jeder Knoten muß alle Atomkoordinaten kennen, ohne dafür genügend Speicherplatz zu haben) beschäftigen, beschreiben wir zunächst die von uns entwickelte Hardware, auf der das Programm abläuft.

4.3.1 Der Transputer

> „A transputer is a microcomputer with its own local memory and with links for connecting one transputer to another transputer.“ [10, Seite 5]

Dieser Kernsatz aus den Herstellerunterlagen beschreibt wohl am besten, was einen Transputer vor anderen Mikroprozessoren auszeichnet. Zu diesen Merkmalen können, je nach Typ, noch andere hinzu kommen. Der Transputer bestimmt mit seinen Besonderheiten und Möglichkeiten als zentrales Element des Rechners wesentlich dessen

Design. Daher erfolgt hier eine sehr geraffte Darstellung der beiden verwendeten Prozessortypen T800 und T414.

4.3.1.1 Der Transputer IMS T800

Im Oktober 1985 [33] brachte die englische Firma INMOS den ersten Transputer auf den Markt. Die Produktfamilie hat sich in den letzten drei Jahren bis zu ihrem jüngsten Mitglied, dem IMS T800G20S entwickelt, der Anfang 1987 verfügbar wurde [29]. Er vereint einen 32 bit 10 MIPS[3] Prozessor (CPU), vier serielle Kommunikationskanäle (links), 4 KByte internes RAM (⇒G) mit sehr kurzer Zugriffszeit (nur ein Prozessorzyklus [4]) und eine 64 bit Gleitkommaeinheit (FPU) mit einer Dauerleistung von 1.5 MFLOPS[4] auf einem einzigen chip. Im Blockdiagramm 4.3 kann man alle diese Einheiten und ihre internen, 32 bit breiten Verbindungswege gut erkennen. Auch die Verbindungen nach außen sind 32 bit breit: im Zeitmultiplex liegen 32 bit Adressen und 32 bit Daten auf den Ein-/Ausgabeanschlüssen. Eine zukünftige Transputerversion (vorläufige Bezeichnung: T801) wird jeweils eigene Anschlüsse für Daten und Adressen erhalten und so ohne Zeitmultiplex auskommen, was noch höhere Datentransferraten vom und zum Speicher erlaubt.

Die zur Zeit erhältlichen Transputerversionen sind für eine Taktfrequenz von 20 MHz spezifiziert. Diese wird intern mittels einer PLL[5] aus dem zugeführten 5 MHz Takt erzeugt (dies erlaubt, verschieden schnelle Komponenten mit einem gemeinsamen Takt zu versorgen).

Der interne Speicher ist nicht, wie sonst üblich, als cache (⇒G) organisiert, sondern belegt den untersten Teil des linearen Adreßraumes. Im Gegensatz zum cache kann der Programmierer durch geschickte Programmierung die Verwendung des on-chip RAMs beeinflussen. Da im on-chip RAM die lokalen Variablen eines Prozesses abgelegt werden können, ersetzt er die großen Registerbänke, die man bei anderen Prozessoren antrifft. Darüber hinaus kann dort auch Programmcode abgelegt werden, der dann sehr schnell ausgeführt werden kann. Davon machen wir in unserem Programm EGOGebrauch.

Die vier seriellen Kommunikationskanäle (*links*) können mit bis zu 1.7 MByte pro Sekunde (unidirektional; 2.35 MBytes bidirektional) [17] Daten mit anderen Transputern austauschen. Da die links über eigene DMA-Maschinen (⇒G) verfügen, kann die Kommunikation parallel zur Berechnung erfolgen.

[3]**M**ega **I**nstructions **p**er **S**econd; ein grobes Maß für die Leistungsfähigkeit eines Prozessors.

[4]**M**ega **Fl**oating point **O**perations **p**er **S**econd; ein grobes Maß für die Gleitkommaleistungsfähigkeit eines Prozessors.

[5]**P**hase **L**ocked **L**oop: phasenstarre Verkopplung eines (internen) Oszillators mit einer (von außen zugeführten) Referenzfrequenz.

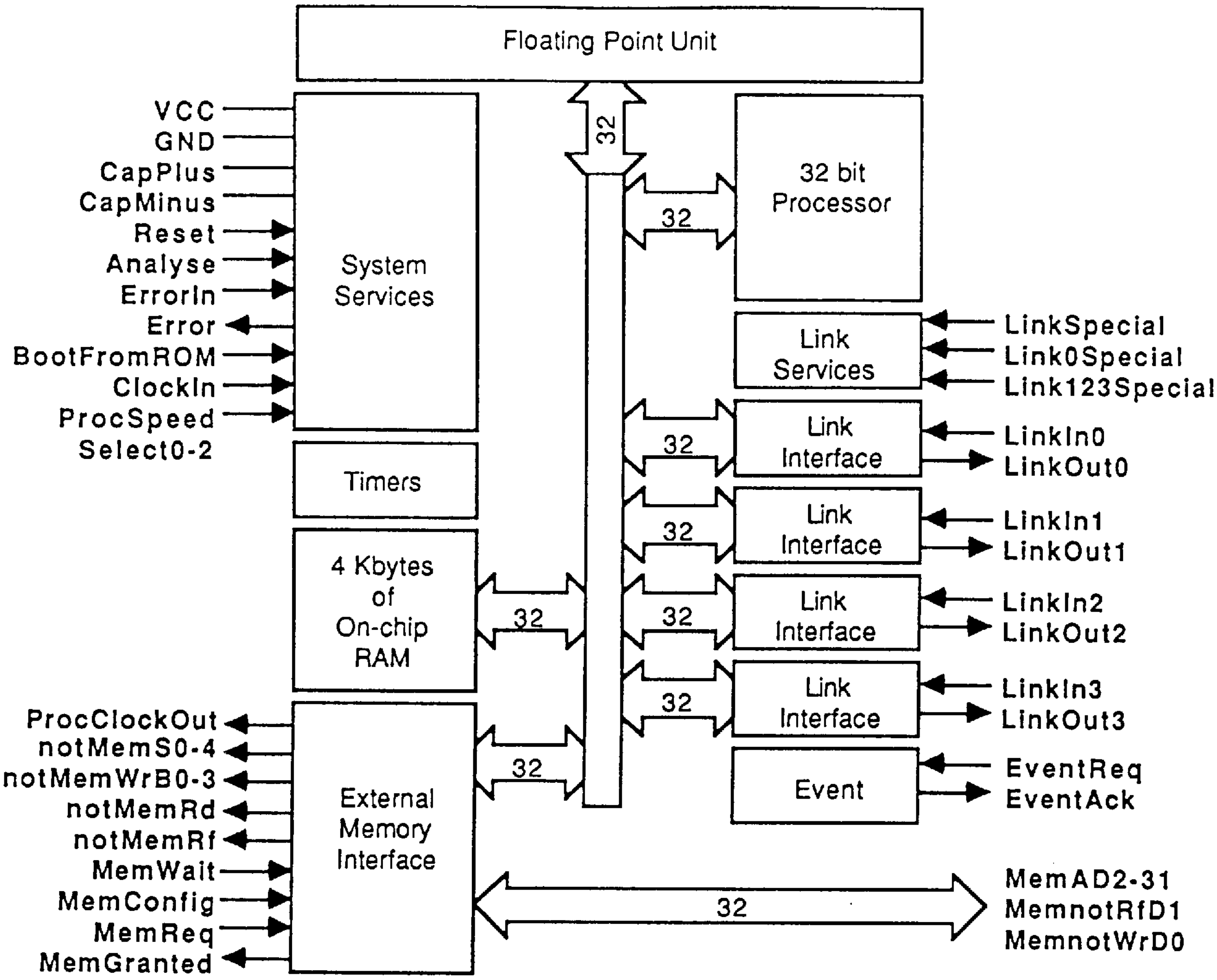


Abbildung 4.3: Blockdiagramm des IMS T800, aus [4, Seite 2].

Auch die FPU kann mit der CPU parallel arbeiten. Während die FPU einfach- oder doppeltgenaue Gleitkommaarithmetik ausführt, kann die CPU mit der Integerarithmetikeinheit weiterarbeiten, etwa um aus einem Feldindex eine Speicheradresse zu berechnen.

Viele weitere interessante Details wären noch anzuführen (etwa die enorm speicherplatzsparende Befehlscodierung oder der in Hardware realisierte Prozessscheduler (⇒G)), doch würde dies den Rahmen dieser Arbeit sprengen. Daher möchten wir hier nur auf die umfangreiche Literatur verweisen. Unsere Hauptinformationsquelle war das Datenblatt [4]. Nicht ganz so konzentriert und daher leichter verständlich sind die Informationen in Zeitschriftenartikeln wie [28, 29, 40, 41]. Von hoher Qualität sind auch die *technical notes* von INMOS [3, 16, 17, 35, 38, 59, 60].

4.3.1.2 Der Transputer IMS T414

Der T414 ist der Vorgänger des T800. Wie schon im Blockdiagramm 4.4 ersichtlich, fehlt gegenüber dem T800 die FPU; der on-chip-Speicher ist nur halb so groß (2 KByte). Des weiteren wurden im „Inneren" des Transputers beim Übergang zum T800 einige

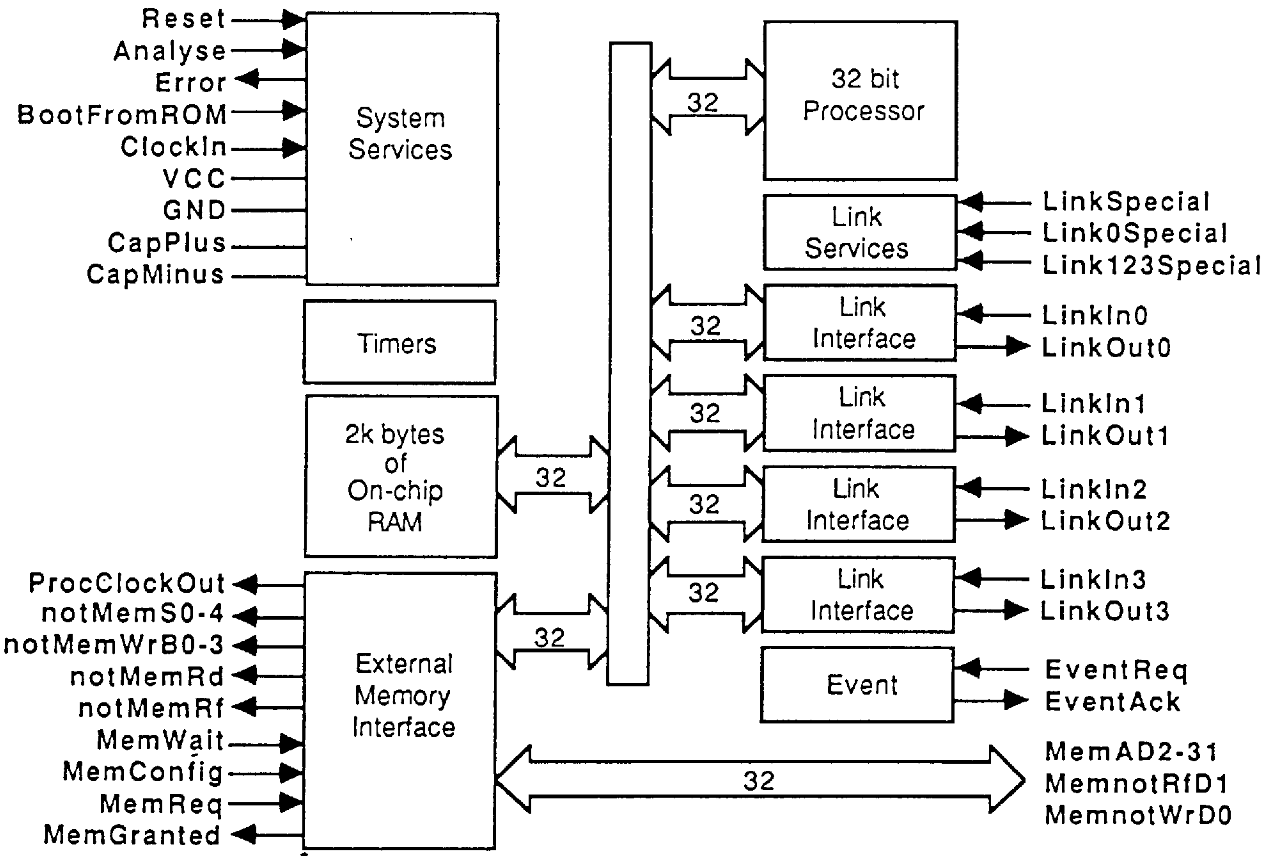


Abbildung 4.4: Blockdiagramm des IMS T414, aus [2, Seite 1].

Verbesserungen vorgenommen, die aber weitgehend technischer Natur sind und den Benutzer kaum betreffen (z.B. ist beim T414 noch kein überlappender Datentransfer auf den links möglich, was zu einer niedrigeren effektiven Datenrate führt). Software-mäßig besteht vom T414 zum T800 vollständige Aufwärtskompatibilität. Das pin-out (⇒G) der Transputer wurde von Anfang an so zukunftsweisend gestaltet, daß es oft möglich ist, alte T414 durch neue T800 zu ersetzen, ohne an der Schaltung etwas ändern zu müssen. Leider ist dies nicht immer ohne weiteres möglich, wie aus Anhang A hervorgeht.

4.3.2 Aufbau eines Rechenknotens

Die Aufgabenstellung dieser Diplomarbeit erforderte es, einen neuartigen Parallelrechner zu entwerfen und zu bauen. Diese Aufgabe wurde von Helmut Grubmüller und dem Autor gemeinsam gelöst. Um das, auch für uns neue Bauelement „Transputer“ (⇒G) ausreichend kennenzulernen und zu erproben, wurde von uns zuerst ein Prototyp der selbst entwickelten Schaltung angefertigt, an dem Änderungen des Schaltplans erprobt werden konnten. Den auf diese Weise praktisch getesteten Schaltplan setzte dann Helmut Grubmüller in ein Layout für eine gedruckte Schaltung um. An Arbeitsmitteln stand ihm dabei nur ein MacIntosh II mit einem Mal- und Zeichenprogramm (MacDraw) und ein Laserdrucker mit einer Auflösung von 300 dpi zur Verfügung. Während dieser Monate konnte der Autor unter Benutzung des Prototypen eine erste Version des Simulationsprogrammes EGOschreiben und austesten. Der Prototyp erfüllte also einen doppelten Zweck: zum einen sollte er die Funktionsfähigkeit der Schaltung sicherstellen und zum anderen diente er als Rechenknoten für die Programmentwicklung.

Diese so wichtige und von uns selbst entwickelte „kleinste Einheit“ unseres Parallelrechners, der einzelne Rechenknoten, soll im folgenden näher beschrieben werden. Ein Rechenknoten*Rechenknoten* besteht aus folgenden Komponenten, die im Blockschaltbild (Abbildung 4.5) zu erkennen sind:

- Transputer IMS T800G20S (⇒G)
- 1 MByte dynamisches RAM (⇒G) als Lokalspeicher für Programm und Daten, aufgebaut aus 4 SIP-Modulen (⇒G) mit je 9 256 Kbit x 1 chips (100 ns)
- Byte-weiser parity check-Mechanismus (⇒G)
- Treiber
- Leuchtdiodenanzeigen (LEDs) für den Betriebszustand

Es fällt auf, daß in der Liste kein ROM (⇒G) auftaucht. Grund dafür ist, daß wir das Transputerfeature ‚boot (⇒G) from link‘ verwenden und das Programm zu Beginn der Berechnungen einmal in das RAM (⇒G) jedes Knotens geladen wird. Dies erhöht nicht nur die Flexibilität des Rechners für andere Anwendungen, sondern erleichtert auch die Programmentwicklung erheblich, da ein geändertes Programm sofort auf den Rechner übertragen und getestet werden kann.

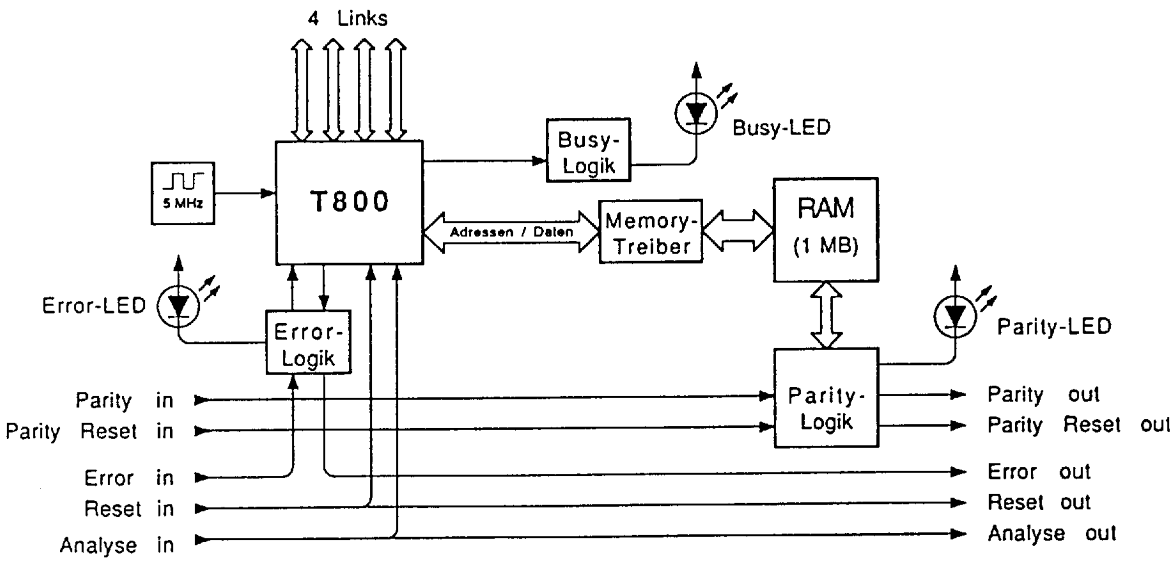


Abbildung 4.5: Blockschaltbild eines Rechenknotens.

Die Größe des Hauptspeichers von 1 MByte je Knoten 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 den gegebenen Mitteln mehr Rechenknoten realisiert werden.

Da unser System sehr viele Knoten enthält (bis zu 60 Prozessoren in einem 19" Gehäuse), muß der Erkennung von Speicherfehlern*Speicherfehler* besondere Aufmerksamkeit geschenkt werden, wie die folgenden Überlegungen zeigen. Bei der Betrachtung von Speicherfehlern ist zwischen *hard errors* und *soft errors* zu unterscheiden (siehe [31]). Erstere werden durch die endliche Lebensdauer des Speicher-chips hervorgerufen (Bauteil irreparabel zerstört) und sind wegen ihrer geringen Wahrscheinlichkeit zu vernachlässigen. Die *soft errors* dagegen sind mit 100–1000 FIT[6] dominant. Sie werden durch einen Ladungsverlust eines Kondensators (ca. 50 fF) im DRAM (⇒G) durch äußere Einflüsse (Alphastrahlung aus dem chip-Gehäuse, Störspannungen [9]) verursacht und zerstören das Bauteil nicht. Definitive Aussagen über die Fehleranfälligkeit ihrer Bauteile (FIT-Zahlen) sind von den Speicherfabrikanten nur schwer zu erfahren. Einzig Hitachi gab telefonisch an, daß für 256 Kbit chips die Ausfallwahrscheinlichkeit bei etwa 100 FIT liegt. Bei einem Computer mit 120 MByte Speicher aus solchen chips ist alle paar Monate mit einem soft error zu rechnen (100 FIT · 36 chips · 120 Knoten =

[6]**F**ailure **I**n **T**ime, 1 FIT ist ein Ausfall des Bauteils in einer Milliarde Bauelementestunden.

$4.32{\cdot}10^{-4}$ Ausfälle/h, das entspricht einer MTBF[7] von 13 Wochen). Solche Rechenzeiten sind für Molekulardynamik durchaus realistisch und ein einziges falsches bit (⇒G) im Speicher kann die ganze Rechnung unbrauchbar machen. Daher wird für jedes byte (der Transputer erlaubt byte-weisen Speicherzugriff) ein zusätzliches Prüf-bit (parity (⇒G)) mit abgespeichert und beim Lesen überprüft. So lassen sich alle ein-bit-Fehler (welche nahezu ausschließlich auftreten) erkennen. Ein solcher Fall wird dem Programm EGO- mitgeteilt, welches dann in der Lage ist, den letzten (fehlerhaften) Rechenschritt mit korrekten Daten zu wiederholen. Daher erübrigt sich eine aufwendige Korrektur auf Hardwareebene [wozu 7 Prüf-bits für 32 Daten-bits, d.h. ein Wort (⇒G) nötig wären].

Je drei Leuchtdioden (LEDs) auf der Frontplatte geben einen Überblick über den momentanen Betriebszustand eines jeden Knotens:

- Die rote error-LED ist direkt an den error-Ausgang des Transputers angeschlossen. Sie leuchtet bei Fehlern wie z.B. Division durch Null, arithmetischem Überlauf sowie falscher Feldindizierung auf. Außerdem kann sie durch den occam II-Befehl `STOP` explizit gesetzt werden.
- Die gelbe parity LED signalisiert einen Speicherfehler. Ihr Zustand wird (in einem flip flop) gespeichert (*parity flag*).
- 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 (siehe Abschnitt 4.4.3.4).

Wie in Abschnitt 4.3.1.1 erwähnt, sind die Transputer T800G20S für eine interne Taktfrequenz von 20 MHz ausgelegt. Durch Anlegen von logischen Spannungspegeln kann diese interne Frequenz in diskreten Stufen (17.5 MHz, 20.0 MHz, 22.5 MHz, 25.0 MHz, 30.0 MHz, 35.0 MHz) geändert werden. Durch Versuche am Prototypen und später auch an ALTER EGO–12 konnten wir feststellen, daß der T800G20S bis zu einer Frequenz von 25 MHz stabil arbeitet, falls man die Toleranzen für Versorgungsspannung und Temperatur enger wählt. Da auch der externe Speicher bei einer Taktfrequenz von 22.5 MHz noch einwandfrei arbeitet (Speicherfehler würden sofort zu einer parity-Anzeige führen) und wir außerdem für hervorragende Kühlung und sehr stabile Versorgungsspannungen gesorgt haben, arbeiten bei uns alle Rechenknoten mit einer Taktfrequenz von 22.5 MHz. Dadurch arbeitet der Rechner nun um 1/8 schneller als ursprünglich geplant.

[7]**M**ean **T**ime **B**etween **F**ailure; mittlere Zeit, die zwischen zwei Fehlern liegt.

4.3.3 Die Platine

4.3.3.1 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 4.6 und Abbildung 4.7).

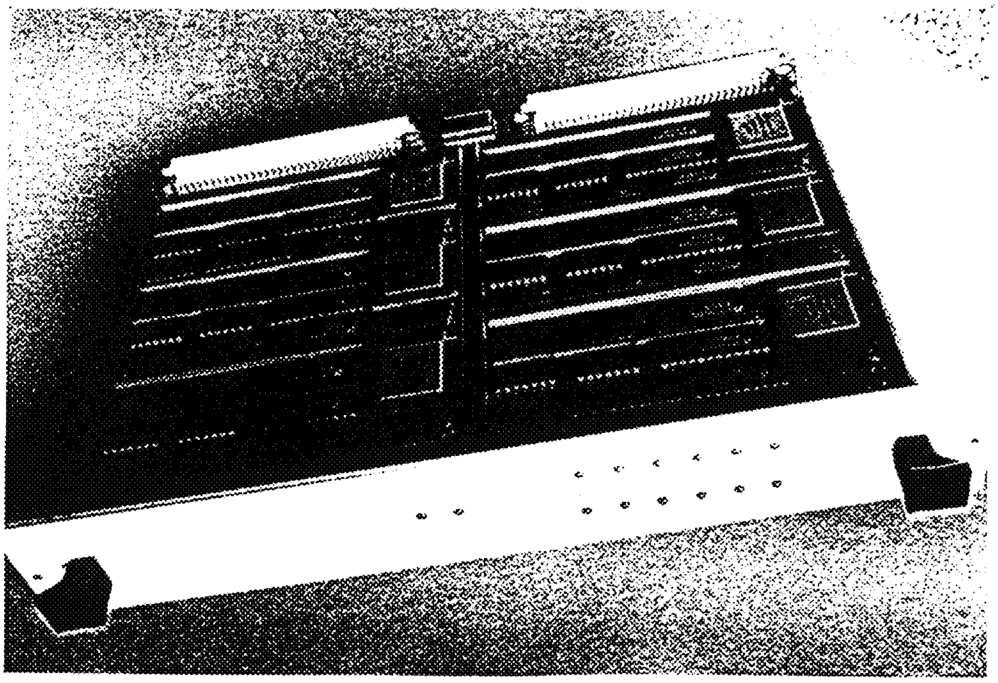


Abbildung 4.6: Fotografie einer Platine des selbstgebauten Parallelrechners

Deshalb wurde beim Platinenlayout eine hohe Packungsdichte angestrebt, um sechs Transputer (⇒G) auf einer Doppeleuropakarte unterzubringen. Dies wurde durch folgende Maßnahmen erreicht:

- Verwendung einer dreilagigen Platine in Verbindung mit einer Leiterbahnbreite und einem Leiterbahnabstand von nur 0.2 mm.
- Verwendung von passiven SMD[8]-Bauteilen.

[8] **S**urface **M**ount **D**evice: die Anschlußdrähte dieser Bauteile werden nicht, wie früher üblich, durch Bohrungen in der Platine gesteckt und verlötet, sondern die Anschlüsse werden direkt auf der Oberfläche der Platine verlötet. Besonders bei mehrlagigen Platinen (*multilayer*) ist dies von Vorteil, da dann die Leiterbahnführung auf den inneren Lagen nicht durch Bohrungen beeinträchtigt wird.

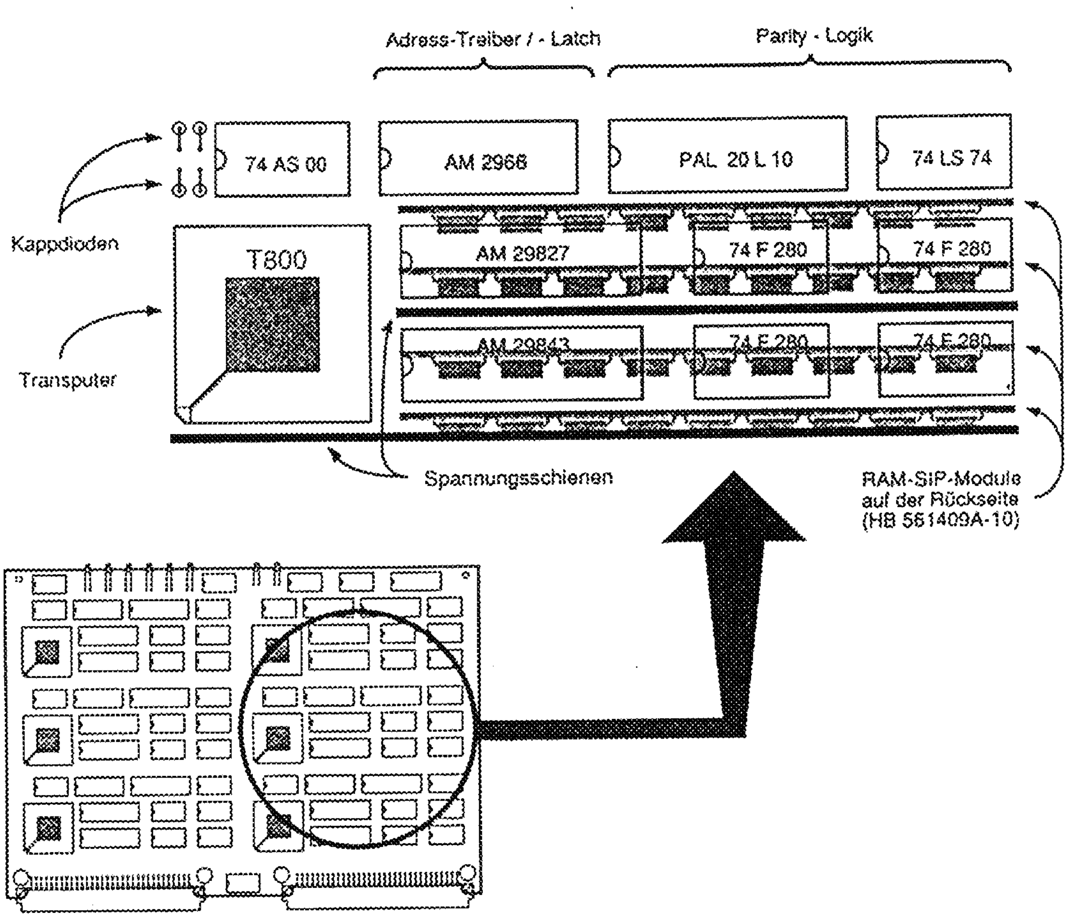


Abbildung 4.7: Schematische Zeichnung der Platine mit vergrößerter Darstellung eines Rechenknotens.

- Beidseitige Bauteilbestückung (Abbildung 4.7):

  Auf der Platinenoberseite wurden Transputer (⇒G), parity-Logik sowie Treiber-ICs montiert. Die Verwendung von speziellen IC-Fassungen ermöglichte es dann, die DRAM-SIP-Module (⇒G) auf der Platinenunterseite einzulöten. Diese Maßnahme erlaubte eine extrem kurze Leiterbahnführung, woraus sehr saubere Signalpegel und eine reduzierte Störanfälligkeit resultieren. Dies konnte durch Oszillographieren sehr gut bestätigt werden.

- Pufferung der Versorgungsspannung durch Verwendung hochkapazitiver Spannungsschienen („Q-Pack"), die stehend zwischen den DIL[9]-Reihen plaziert sind. Eine Pufferung der Versorgungsspannungen ist nötig, um Stromspitzen, die von

[9]**D**ual **i**n **L**ine, übliche Gehäuseform für integrierte Schaltkreise.

den DRAMs (⇒G) verursacht werden, abzublocken, d.h. Spannungseinbrüche zu vermeiden. Durch Verwendung dieser Spannungsschienen lassen sich Abblockkondensatoren einsparen, für die auf der Platine kaum Platz ist.

Wie erwähnt, ist die Rechner-Topologie (⇒G) 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 heraus auf die backplane (⇒G) zu führen (siehe auch Abschnitt 4.3.3.2).

4.3.3.2 Zehn Platinen in einem 19"-Einschub

Jede 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*, z.B. IMS C004). 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 link-Verbindungen ist auf der backplane auch noch eine Interruptlogik untergebracht, die es ermöglicht, jeden der 60 Transputer getrennt anzusprechen. Ein 1-aus-16 Dekoder (`74154`) selektiert eine der 10 Platinen und ein 1-aus-8 Dekoder (`AM26LS2538`) auf jeder Platine wählt dann einen der sechs Transputer auf der Platine aus.

Eine Platine beansprucht vier Tiefeneinheiten in einem 19"-Einschub; das bedeutet, daß 10 Platinen oder 60 Transputer in diesem Einschub Platz finden. Die verbleibenden zwei Tiefeneinheiten werden für reset-Taster und Betriebsanzeige genutzt.

Weitere Einzelheiten sowie die Beschreibung einer verbesserten Hardware mit einer sechslagigen Platine und 4 MByte Speicher je Transputer findet sich in [36].

4.4 Software

Kommen wir nun zur Beschreibung der Software und damit zur Lösungsstrategie für das Schlüsselproblem (jeder Knoten muß alle Atomkoordinaten kennen, hat aber für deren Speicherung zu wenig Hauptspeicher) zurück. Die Beschreibung des Programms ist nicht ganz einfach, da der Vorgang des „Lesens“ von Natur aus sequentiell ist. Der wesentliche Punkt des Programms aber ist seine Parallelität, die sich nur schwer in einer sequentiellen Darstellungsform ausdrücken läßt. Es sei dem Autor daher erlaubt, jeweils die verschiedenen parallelen Prozesse zu einem Aspekt [Laden von Programm und Daten (Abschnitt 4.4.3.3), Kommunikation der Koordinaten (Abschnitt 4.4.3.4), Kommunikation der Kräfte (Abschnitt 4.4.3.5)] gemeinsam zu betrachten, um anschließend dieselben Prozesse unter einem anderen Aspekt nochmals zu untersuchen. Der Autor hofft, dadurch der Parallelität am besten gerecht zu werden.

4.4.1 Die Programmiersprache occam II

Im allgemeinen wird die Wahl der Programmiersprache von der Problemstellung und nicht von der zugrundeliegenden Hardware abhängen (etwa LISP für AI-Anwendungen, FORTRAN für wissenschaftliche Berechnungen). Dies ist hier nicht so: es gibt eine Sprache, deren Primitive (⇒G) genau auf die Maschinensprache des Transputers (⇒G) abgestimmt sind, und die sich daher besonders effizient übersetzen läßt: occam II! Motiviert vom Wunsch nach besonders schnellen und kompakten Programmen erweist sich daher occam II [7, 49, 56, 62] für Transputer als die Sprache der Wahl.

Mit occam II wird ein Programm als eine Sammlung paralleler Prozesse beschrieben, die untereinander und mit „dem Rest der Welt" über Kanäle kommunizieren. C. A. R. Hoare hat dieses Modell kommunizierender sequentieller Prozesse in [44] theoretisch beschrieben. occam II ist seine praktische Realisation. occam II-Programme sind aus drei primitiven Prozessen aufgebaut [54, Seite 3]:

`variable := expression`	Zuweisung des Wertes des Ausdrucks an die Variable
`channel ? variable`	Einlesen eines Wertes von einem Kanal in eine Variable
`channel ! expression`	Ausgabe des Wertes eines Ausdrucks in einen Kanal

Jeder occam II-Kanal bildet eine unidirektionale Verbindung zwischen zwei parallelen Prozessen. Die Kommunikation erfolgt synchronisiert und ungepuffert. Werden diese primitiven Prozesse kombiniert (für sequentielle Abfolgen z.B. mit den Konstruktoren `SEQ`, `IF`, `CASE` und `WHILE`, für parallele Programme z.B. mit den Konstruktoren `PAR` und `ALT`), so sind die entstehenden Konstrukte selbst wieder Prozesse, die kombiniert werden können. Die Kommunikation der mit `PAR` erzeugten parallelen Prozesse erfolgt dabei *ausschließlich* über Kanäle.

Diese Art der Kopplung von Prozessen — und damit auch von Prozessoren — über Kanäle ist es, die eine (fast) lineare Leistungsteigerung bei Verwendung mehrerer Prozessoren ermöglicht. Dem steht ein anderes Konzept gegenüber, bei dem mehrere Prozessoren auf einen *gemeinsamen* Speicher zugreifen (*shared memory*). Eine spezielle Verteilungslogik (*bus arbiter*) sorgt dafür, daß es zu keinen Konflikten kommt, falls zwei (oder mehrere) Prozessoren *gleichzeitig* auf den Speicher zugreifen wollen. Und genau hierin liegt auch der Nachteil dieses Konzeptes: es kann immer nur *ein* Prozessor auf den Speicher zugreifen; gegebenenfalls müssen die anderen Prozessoren warten, bis der Speicher für sie verfügbar wird. Genau dieses „Warten“ aber gilt es zu vermeiden, was mit dem Kanalkonzept von occam II erreicht wird. Im Gegensatz zum shared memory ist der Speicher hier *verteilt*, und 100 Prozessoren können darauf genausoschnell zugreifen, wie ein einzelner, da jedem ein *eigener* Speicher zur Verfügung steht. Auch die Kommunikationsbandbreite eines Transputernetzwerks nimmt mit wachsender Transputerzahl zu: jeder Transputer bringt vier zusätzliche links ein, die die Bandbreite erhöhen.

Es ist sowohl möglich, auf *einem* Transputer *mehrere* parallele Prozesse laufen zu lassen, als auch diese auf mehrere Transputer zu verteilen. Wie diese Verteilung aussieht, wird nicht im Programm selbst, sondern in einem speziellen Konfigurationsteil festgelegt. So läßt sich auch sehr einfach eine Umverteilung vornehmen.

4.4.2 TDS-Entwicklungssystem und folding editor

An dieser Stelle sind einige Anmerkungen zur Entwicklungsumgebung TDS (**T**ransputer **D**evelopment **S**ystem) von INMOS, in der das Programm entstand und auch abläuft, notwendig. Zentraler Bestandteil des TDS [1, 5, 6, 8, 11, 12, 13, 14, 15] ist der folding editor, aus dem heraus auch der compiler und eigene EXEs (das ist die übliche Form eines Benutzerprogramms) 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 (Abbildung 4.8). Wird dagegen das fold entfaltet, so gibt es seinen gesamten Inhalt preis, der ebenfalls wieder folds enthalten kann (siehe Programm-listing, Anhang C). Geöffnete folds erkennt man an drei geschweiften Klammern ({{{ Titelzeile). Ihr Ende wird entsprechend markiert (}}}). So ist mit Hilfe der folds eine übersichtliche und gut strukturierte Gestaltung des Programmtextes möglich. Alle editor-Funktionen, der Compiler und auch Benutzerprogramme (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 Moleküldaten-files mit Hilfe eines EXEs in ein fold mit hierarchischer Sub-fold-Struktur 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

```
... Vereinbarungen
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)
    Buffer.out (scr.in[2], output, to.net.out, stopper[1])

  PAR
    ... Buffer fuer Analyse
    Analyse (scr.in[3], to.Ana[11], P.error)

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

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

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

Abbildung 4.8: Übersichtliche Darstellung der Grobstruktur des Programms *Controller* mit geschlossenen folds.

vorzunehmen, ohne das TDS verlassen zu müssen. Eine Simulation wird nun einfach durch Aufruf des entsprechenden EXEs auf dem fold mit den Daten des gewünschten Moleküls gestartet.

4.4.3 Das Programm

4.4.3.1 Lokale oder verteilte Datenhaltung

Eng mit der Festlegung der Kommunikationsstruktur und damit der Topologie (⇒G) eines Mehrprozessorsystems, ist eine Entscheidung über die Speicherung der Daten im Netzwerk verknüpft. Hierbei gilt es, zwischen den beiden folgenden Möglichkeiten eine grundsätzliche Entscheidung zu treffen:

1. Daten werden im Ring „herumgereicht", d.h. jeder Knoten hat nur einen Ausschnitt aus dem Gesamtdatensatz zur Verfügung. Durch das „Herumreichen" aller dieser Ausschnitte aber kann doch jeder Prozessor auf die Gesamtheit aller Da-

ten zugreifen. Dieser Zugriff kann nicht wahlfrei erfolgen, sondern der verfügbare Ausschnitt wandert wie ein Fenster über den gesamten Datensatz.

2. Daten werden lokal gespeichert, d.h. jeder Prozessor kann immer wahlfrei auf den gesamten Datensatz zugreifen. Besondere Maßnahmen müssen dabei die Konsistenz der Datensätze der verschiedenen Knoten sicherstellen (z.B. update-Schritte).

Der Vorteil einer Entscheidung für 1. ist eine Einsparung von Speicherplatz. Diese Einsparung bringt aber den Nachteil von zusätzlicher Kommunikation mit sich. Im Fall einer Entscheidung für 2. sind Vorteil und Nachteil umgekehrt: es wird mehr Speicherplatz benötigt, dafür steht diese gespeicherte Information allen Transputern (⇒G) immer zur Verfügung. Es ist eine detaillierte Abwägung zwischen verfügbarem Speicherplatz, Zeitersparnis durch Reduktion des Kommunikationsaufwandes und der Notwendigkeit einer lokalen Speicherung erforderlich. Probleme, die einen wahlfreien Zugriff (*random access*) auf alle Daten erfordern, machen eine lokale Speicherung unumgänglich. Oft läßt sich dies aber durch eine andere Problemlösungsstrategie umgehen.

Bei unserem Problem, der Molekulardynamiksimulation, insbesondere der Berechnung der elektrostatischen Wechselwirkung, ist ein Zugriff auf alle Atomkoordinaten erforderlich, da sich die Summation in Gleichung (4.3) über alle Paare erstreckt. Die Reihenfolge, in der die Summation ausgeführt wird, spielt aber wegen der Kommutativität der Addition keine Rolle. Da ferner nur die Koordinaten von jeweils zwei Atomen gleichzeitig benötigt werden[10], ist die Wahlfreiheit des Datenzugriffs keine unbedingte Voraussetzung; der Zugriff kann vielmehr in beliebiger Reihenfolge erfolgen (daher können die Atome *beliebig* auf die Transputer verteilt werden).

Bei Berechnung der anderen Wechselwirkungen, wie Winkel- oder Dihedralwechselwirkung ist dies anders (daher ist es *günstig*, aber nicht notwendig, benachbarte Atome demselben Knoten zuzuweisen). An diesen Wechselwirkungen sind mehr als zwei Atome zugleich beteiligt (im Beispiel der genannten Wechselwirkungen drei bzw. vier Atome). Die zusätzlichen Koordinaten müssen lokal gespeichert werden, um zumindest in der benötigten Untermenge aller Koordinaten wahlfreien Zugriff zu ermöglichen. Dieser Punkt ist im Abschnitt 4.4.3.6.1 näher beschrieben.

Die Berechnung der Wasserstoffbrücken fällt in keine dieser Kategorien, da zu ihrer Berechnung wahlfreier Zugriff auf viele Koordinaten erforderlich ist. Eine Lösungsmöglichkeit ist in Abschnitt 4.7 angedeutet.

[10] wobei die Koordinaten eines Atoms zu den „eigenen“ zählen, während die Koordinaten des anderen Atoms „fremd“ sind.

Bei unserem Problem, der Molekulardynamiksimulation, ist die Zeitersparnis bei lokaler Speicherung wegen der allgemein geringen Zahl zu übertragender bytes ($\Rightarrow$G) und der unabhängig parallel arbeitenden DMA-Maschinen ($\Rightarrow$G) nicht sehr groß, zumal auch bei lokaler Speicherung ein Zugriff auf ein (lokales) Feld erfolgen muß (der ebenfalls Zeit kostet). Folgende Abschätzung zeigt dies: die Berechnung eines Integrationsschrittes bei 12 637 Atomen und 12 Rechenknoten dauert etwa 387 s. Dabei wird über jedes link etwa ein halbes Megabyte an Koordinaten übertragen, was bei 1.7 MByte/s Übertragungsgeschwindigkeit 0.3 s dauert; das sind 0.08% der Rechenzeit und damit vernachlässigbar. Andererseits ist Speicherplatz knapp, denn unser Ziel lautet: Simulation von möglichst vielen Atomen auf wenigen Transputern. Daher werden in der momentanen Implementation von EGOdie Atomdaten (Koordinaten, Ladungen, Typklasse) *nicht* lokal, sondern im Ring verteilt gespeichert.

Weil die Rechenzeit quadratisch, die Datentransferzeit aber nur linear mit der Zahl der Atome wächst, ist das Verhältnis Rechenzeit/Transferzeit für kleinere Moleküle schlechter. Für deren schnelle Simulation wäre die Entscheidung für eine verteilte Speicherung von Atomdaten nochmals zu überdenken, zumal dann auch pro Atom mehr Speicherplatz verfügbar ist. Insbesondere wäre dann auch zu überlegen, alle Daten des ganzen Moleküls lokal in jedem Knoten zu halten und die Positionen in einem gesonderten Schritt anzugleichen. Dafür wäre dann anstelle der Ringtopologie die verwandte Torustopologie besser geeignet, da sich dort die Kommunikationszeit um $\sqrt{2}$ verkürzt (höhere Konnektivität). Zu bedenken ist freilich, daß dann nur Moleküle mit maximal ca. 4 700 Atomen simuliert werden können, unabhängig von der Zahl der Transputer; hierfür ist einzig die Größe des Speichers pro Knoten von Bedeutung.

Bei der verteilten Speicherung, wie sie von EGOverwendet wird, können schon mit 12 Transputern bis zu 40 000 Atome simuliert werden (weil es sich um Paarkräfte handelt, müssen die Koordinaten stets in zwei Prozessoren zugleich gespeichert werden, daher ist hier nicht die 12-fache Zahl an Atomen im Vergleich zur lokalen Speicherung zu erzielen). Diese Zahl steigt linear mit der Zahl der Rechenknoten.

Daneben gibt es noch unveränderliche Daten, die für ganze *Gruppen* von Atomen gleich sind. So sind beispielsweise die van der Waals-Parameter oder die Massen der Atome eines Atomtyps (z.B. NH_3, C, N, H, ...) alle gleich und müssen nur einmal für die ganze Gruppe gespeichert werden. Solche Daten werden stets lokal gespeichert, da die Gruppierung das Datenvolumen auf bis zu 1/1000 reduzieren kann.

4.4.3.2 Aufgliederung in Module

Das erstellte Programmpaket ist aus drei Modulen, *Interface*, *Controller* und *Program*, aufgebaut[11]. Das Modul *Interface* bildet eine Schnittstelle 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 wie XPLOR benutzt. Das Modul *Controller* beinhaltet neben der Datenversorgung des eigentlichen Rechenprogrammes auch Überwachungs- und Analysefunktionen. Das Modul *Program* enthält den Code für die Molekulardynamiksimulation und läuft auf jedem Netzwerktransputer ab. Die Trennung *Controller/Program* ergibt sich aus den verschiedenen Anforderungen und der Einbindung in die INMOS-Entwicklungsumgebung. Der *Controller* läuft auf dem host-Transputer ab [bei uns ein T414 auf einem etwas umgebauten B004-board von INMOS (siehe Anhang A) in einem IBM-kompatiblen AT] und ist daher in Form eines EXEs geschrieben. Er hat damit Zugriff auf die hierarchische folding-Struktur. Das Rechenprogramm *Program* läuft auf mehreren Transputern ab und hat daher die Form eines PROGRAMs [1, 13].

Zur Durchführung einer Molekulardynamikrechnung startet man das EXE *Controller* auf dem gewünschten Daten-fold. Alle Steuerdaten, wie die Länge des Integrationszeitschritts, die Zahl der Atome im Molekül, die Häufigkeit der Datenaufzeichnung, Parameter für die Minimierung und die Equilibrierung sind, gruppiert in einem Subfold (dem Steuerungs-fold), darin enthalten.

Die nachfolgende Programmausführung besteht aus der Ladephase und der Berechnungsphase. In beiden Phasen arbeiten die Module *Controller* bzw. *Program* parallel auf dem T414- bzw. den 12 T800-Transputern.

4.4.3.3 Laden des Netzwerkes

Nachdem das Modul *Interface* das Daten-fold erzeugt hat, kann der *Controller* darauf starten. Zuerst startet er auf jedem T800 die eigenständigen Parallelprogramme *clear*, welche alle parity bits (⇒G) der externen Speicher zurücksetzen, um eine definierte Ausgangslage herzustellen. Der *Controller* führt anschließend folgende Schritte (im folgenden zur Unterscheidung als „*Controller*-Schritte“ bezeichnet) nacheinander aus:

1. Laden des parallelen Rechenprogramms *Program* in den Speicher jedes Rechenknotens.

[11] Programmnamen beziehen sich auf den Programmquelltext, Anhang C. Sie bezeichnen das jeweilige Modul oder ein Unterprogramm und sind *kursiv* gedruckt.

2. Zusammenstellen und Laden der Datensätze für jeden Rechenknoten.

3. Beenden der Ladephase und Umschalten des gesamten Netzwerks in die Berechnungsphase.

4. Durchführung der Berechnungsphase: die hier zu leistenden Aufgaben des Moduls *Controller* sind im Abschnitt 4.4.3.7 zusammen mit den ausführenden Prozessen beschrieben.

Wir wollen nun die einzelnen *Controller*-Schritte während der Ladephase näher erläutern, dabei wird stets auf Abbildung 4.2 Bezug genommen.

Im 1. *Controller*-Schritt erfolgt das Laden des Programms in die einzelnen Knoten, da, wie in Abschnitt 4.3.2 erwähnt, in den einzelnen Knoten kein ROM vorhanden ist. Zu diesem Zweck wurde das von INMOS im Rahmen des TDS gelieferte und im Quelltext vorhandene Ladeprogramm umgeschrieben und in den *Controller* integriert. Auf Seiten des Benutzers bringt dies die Vereinfachung des automatischen Programmladens; vom Standpunkt der Rechnerkonstruktion aus war es notwendig, um ein automatisches Rücksetzen der parity flags (Abschnitt 4.3.2) zu ermöglichen[12].

Dieses Ladeprogramm bekommt seine Informationen über die Topologie des Netzwerks, die Zahl der Transputer und die genaue Verschaltung der Knoten aus dem Konfigurationsteil (Abschnitt 4.4.1) des Rechenprogramms *Program*. In dieser Phase des Ablaufs wird jedem T800 auch seine „Identität", d.h. eine Nummer zugewiesen: das Laden des Programms erfolgt ausgehend von link 1 des T414 im Uhrzeigersinn entlang der Ringstruktur. Dabei bekommt der erste T800 die höchste Nummer (in Abbildung 4.2 die Nummer 5), der nächste die nächstkleinere usw. bis der letzte T800 die Nummer 0 erhält. Das Laden selbst ist ein mehrstufiger Vorgang, der in [12, Kapitel 6: Loading transputer networks] beschrieben ist.

Im 2. *Controller*-Schritt übernimmt nun das soeben geladene Programm *Program* die Kontrolle über jeden T800 und unterstützt die Phase des Ladens der Daten (oberer Teil im Flußdiagramm, Abbildung 4.10). Der *Controller* berechnet zuerst, wieviele Atome jeder Transputer als ihm „eigen" betrachten kann. Damit die Lastaufteilung möglichst gleichmäßig ist, darf der Unterschied in der Zahl der eigenen Atome zweier Transputer

[12]Da auf dem INMOS-board B004 kein frei programmierbarer Ausgang vorhanden ist, wird das Rücksetzen der parity flags von der reset-Leitung gesteuert, mit der auch alle T800 verbunden sind. Nun greift der T800 aber nach Empfang eines reset-Signals 36 mal in einem Modus auf den Speicher zu, der für diesen nicht geeignet ist, was ein sofortiges erneutes Setzen des parity flags zur Folge hat. Dies läßt sich nur verhindern, wenn mit der reset-Leitung auch die analyse-Leitung des Transputers aktiviert wird (sog. „analyse mode"). Dies aber war mit dem original INMOS-Ladeprogramm nicht möglich.

maximal 1 sein (im allgemeinen wird die Zahl der Atome **kein** ganzzahliges Vielfaches der Transputerzahl sein). Dann wird aus dem Daten-fold die entsprechende Zahl von Daten eingelesen und über das link 2 des T414 an das Netzwerk übermittelt. Dabei werden gleichartige Daten (z.B. x-Koordinaten) in einem Vektor zusammengefaßt und gemeinsam übermittelt. Dies bringt den Vorteil, daß im fold alle Daten eines Atoms beieinander stehen, was für manuelle Änderungen mit dem editor sinnvoll ist, während es für die Berechnung günstiger ist, wenn alle artgleichen Daten zusammenstehen. Die Daten werden entgegen dem Uhrzeigersinn, ausgehend von link 2 des T414, geladen. Den ersten Datensatz erhält auf diese Weise T800 Nummer 0, den zweiten T800 Nummer 1 usw. bis alle Daten ausgeteilt sind[13], denn das Programm *Program* ist so aufgebaut, daß es den ersten eintreffenden Datensatz als „eigene“ Daten lokal abspeichert und von da an alle Datenpakete an den Nachbarn weiterreicht.

Abschließend werden noch die globalen Daten (diese werden vom Programm *Program* an einem speziellen tag (⇒G) erkannt und können dadurch von den lokalen Daten unterschieden werden), wie Energiekonstanten und Integrationszeitschritt (die für alle Knoten gleich sind) übertragen. Das Auftreten von letzterem signalisiert den Rechenknoten auch das Ende der Ladephase (*Controller*-Schritt 3.), woraufhin sie in die Berechnungsphase (*Controller*-Schritt 4.) übergehen.

4.4.3.4 Austausch von Atomkoordinaten

Wie geht nun während der Berechnung die Kommunikation im Ring vor sich? Alle wesentlichen Aspekte lassen sich schon bei Betrachtung der elektrostatischen Wechselwirkung erläutern, weshalb wir uns vorläufig auf sie beschränken. Ferner nehmen wir an, es seien alle Paare zu berücksichtigen, d.h. jedes Atom wirkt auf jedes andere ein[14].

Wir sahen in Abschnitt 4.2, daß jeder Transputer über einen Satz „eigener“ Atome (d.h. deren Koordinaten, Geschwindigkeiten, Ladungen, etc.) verfügt. Dabei zeichnet die in einem Satz zusammengefaßten Atome keine weitere gemeinsame Eigenschaft aus; die Zuteilung der Atome zu Transputern kann beliebig erfolgen.

Die Coulombenergie E_i für die „eigenen“ Atome i setzt sich also aus folgenden zwei Anteilen zusammen

$$E_i = \sum_{\substack{\text{eigene} \\ \text{Atome } j,\, j \neq i}} \frac{q_i q_j}{4\pi\varepsilon r_{ij}} + \sum_{\substack{\text{fremde} \\ \text{Atome } k}} \frac{q_i q_k}{4\pi\varepsilon r_{ik}} \quad , \tag{4.4}$$

[13]Aufgrund einer technischen Beschränkung des TDS (es ist nur gleichzeitiger Zugriff auf max. 4 folds möglich) erfolgt diese Verteilung der Daten in zwei aufeinanderfolgenden Stufen.

[14]Tatsächlich sind manche Paare von der elektrostatischen Wechselwirkung ausgeschlossen; siehe Seite 7.

einer Summation über alle anderen ($j \neq i$) „eigenen“ Atome j und einer Summation über alle „fremden“ Atome k.

Die Ausführung dieser Summationen und aller dazu notwendigen Hilfsschritte sowie die anschließende Integration der Newtonschen Gleichungen (2.2) wird als ein *Berechnungszyklus* bezeichnet. Er beginnt mit der Berechnung der Kräfte und Energien und endet mit der Abfrage der Energiewerte. Die wiederholte Ausführung eines Berechnungszykluses kennzeichnet die Struktur des Programms EGO.

Bevor wir einen Berechnungszyklus genauer schildern, geben wir einen Überblick über die wichtigsten in den Netzwerktransputern und im host-Transputer (T414) parallel ablaufenden Prozesse. Es sind dies in den Netzwerktransputern die Prozesse *Berechnung*, *Routing*, *Add.buffer* und *Multiplexer*, deren Funktion und Zusammenwirken in Abschnitt 4.4.3.6 genauer erläutert werden wird, und im host-Transputer der Prozeß *Master*, auf den wir im Abschnitt 4.4.3.7 näher eingehen. In einem Berechnungszyklus arbeiten alle diese Prozesse parallel zusammen. Ein Berechnungszyklus besteht aus folgenden Schritten (zur Unterscheidung im folgenden „Zyklusschritte“ genannt), die sequentiell auf jedem Transputer ablaufen:

1. Jeder Transputer berechnet die Wechselwirkungen der ihm zugeteilten Atome untereinander (linker Term in Gleichung (4.4)).

2. Nun berechnet er die Wechselwirkungen der ihm zugeteilten Atome mit allen anderen Atomen im Ring. Dazu werden die folgenden Schritte (zur Unterscheidung „Subschritte“ genannt) mehrfach ausgeführt:

   (a) Jeder Transputer kopiert (d.h. die Werte bleiben lokal gespeichert) die Koordinaten „seiner“ Atome im Uhrzeigersinn (Abbildung 4.2) zu seinem Nachbarn weiter. Ebenso empfängt er von seinem anderen Nachbarn dessen Koordinaten („fremde“ Koordinaten; für deren vorübergehende Speicherung steht zusätzlicher Speicherplatz zur Verfügung, d.h. die Koordinaten der „eigenen“ Atome bleiben gespeichert).

   (b) Jetzt werden alle Wechselwirkungen dieser Atomgruppen („eigene“ Atome mit „fremden“ Atomen) untereinander berechnet (ein Teil des rechten Terms in Gleichung (4.4)).

   (c) Die „fremden“ Koordinaten werden zum nächsten Transputer weitergereicht.

   Wie oft dieser Zyklusschritt innerhalb eines Berechnungszykluses wiederholt wird, und welche seiner Subschritte dabei ausgeführt werden, wird in den folgenden Absätzen diskutiert. Dabei wird die erste Ausführung des Zyklusschritts 2. als

2.1. bezeichnet, die zweite als 2.2. usw. Die bisher genannten Schritte werden von den Prozessen *Berechnung* und *Master* durchgeführt und organisiert.

3. Die Kräfte auf die „eigenen“ Atome, welche nicht selbst, sondern von anderen Knoten berechnet wurden, werden jetzt zu den selbst berechneten Kräften addiert (dies ist in Abschnitt 4.4.3.5 näher erklärt). An diesem Zyklusschritt sind die Prozesse *Berechnung* und *Add.buffer* beteiligt.

4. Da nun die Gesamtkraft auf jedes einzelne „eigene“ Atom bekannt ist, kann ein Integrationsschritt ausgeführt werden. Dabei werden die Positionen aller Atome neu berechnet (*Berechnung*).

5. Abschließend werden vom Prozeß *Master* im Modul *Controller* die Energiewerte (Coulombenergie, van der Waals-Energie, etc.) abgefragt. Dieser Vorgang ist näher am Ende des Abschnitts 4.4.3.6.1 erläutert.

Damit ist ein Berechnungszyklus abgeschlossen und ein neuer beginnt.

Noch offen ist die Beantwortung der Frage, wie oft die innere Schleife (Zyklusschritt 2.) ausgeführt wird. Um die Übersicht zu behalten, nehmen wir dazu einen Rechner mit sechs Transputern mit der in Abbildung 4.2 dargestellten Ringtopologie an. Ferner wollen wir das Newtonsche Prinzip, Gleichung (4.2), benutzen, um die Rechenzeit zu halbieren.

Falls *jeder* Transputer die Wechselwirkungen seiner Atome mit *allen* anderen Atomen berechnete, so würden alle Wechselwirkungen doppelt berechnet[15]. Jeder Transputer darf also nur mit der Hälfte der anderen „wechselwirken“, und zwar so geschickt, daß trotzdem alle Paare berechnet werden. Als erste Lösung bietet sich die Aufteilung nach Tabelle 4.1 an. Dabei bedeutet die Nomenklatur *x rechnet y*, daß Transputer Nummer x die Wechselwirkung „seiner“ Atome mit denen des Transputers Nummer y ausrechnet (der Austausch der Kräfte ist hier noch nicht berücksichtigt; er wird erst in Abschnitt 4.4.3.5 behandelt). Bei dieser Aufteilung wird keine Wechselwirkung mehr doppelt gerechnet. Doch bei genauerem Hinsehen erkennt man, daß die Wechselwirkung zwischen den Atomen des Transputers 0 und denen von Transputer 3 fehlt. Die richtige Aufteilung ist in Tabelle 4.2 angegeben. Unschwer erkennt man, daß während des letzten Schritts, Zyklusschritt 2.3., nur die Hälfte der Transputer rechnet. Dieses Verhalten muß sich auch in der Anzeige der (grünen) busy-LEDs widerspiegeln. Man erwartet, daß während eines Zyklusschritts 2. die Hälfte von ihnen dunkel ist, während

[15] Transputer Nummer 0 rechnet die Wechselwirkung „seiner“ Atome mit den Atomen von Transputer Nummer 3 und Transputer Nummer 3 rechnet die Wechselwirkung „seiner“ Atome mit den Atomen von Transputer Nummer 0, was nach Gleichung (4.2) aber die gleichen Paarwechselwirkungen, nur mit umgekehrtem Vorzeichen, beinhaltet.

Transputer		Zyklusschritt 1.	2.1.	2.2.
0	rechnet	0	4	5
1	rechnet	1	5	0
2	rechnet	2	0	1
3	rechnet	3	1	2
4	rechnet	4	2	3
5	rechnet	5	3	4

Tabelle 4.1: Falsche Aufteilung der Rechenlast

Transputer		Zyklusschritt 1.	2.1.	2.2.	2.3.
0	rechnet	0	3	4	5
1	rechnet	1	4	5	0
2	rechnet	2	5	0	1
3	rechnet	3	1	2	
4	rechnet	4	2	3	
5	rechnet	5	3	4	

Tabelle 4.2: Richtige Aufteilung der Rechenlast

sonst alle leuchten. Die realen Verhältnisse sind aber etwas komplizierter und in Tabelle 4.3 dargestellt.

Im einzelnen ist der Ablauf wie folgt:
In Zyklusschritt 1. berechnet jeder Transputer die Wechselwirkung seiner „eigenen“ Atome untereinander und reicht deren Koordinaten anschließend im Uhrzeigersinn (vgl. Abbildung 4.2) an seinen Nachbarn weiter. Dies erkennt man am gleichmäßigen Leuchten aller sechs busy-LEDs.
In Zyklusschritt 2.1. werden keine Berechnungen ausgeführt, da die „fremden“ Koordinaten bei keinem Transputer die richtige Identität (siehe Tabelle 4.2) haben; sie werden nur zum Nachbarn weitergereicht, was das Symbol $\nearrow$ andeutet. Dies geht sehr schnell, bedingt aber häufige Speicherzugriffe: die busy-LEDs blitzen kurz hell auf.
Die Vorgänge in Zyklusschritt 2.2. sind identisch zu denen in Zyklusschritt 2.1.
Der Zyklusschritt 2.3. ist zweigeteilt: in Zyklusschritt 2.3.I. beginnen die Rechenknoten 0, 1 und 2 gemäß Tabelle 4.2 mit der Berechnung externer Wechselwirkungen. Die Knoten 3, 4 und 5 aber schieben die erhaltenen „fremden“ Koordinaten nur weiter, da

	Zyklusschritt							
Transputer	1.	2.1.	2.2.	2.3.I.	2.3.II.	2.4.	2.5.	1.
0	0	↗	↗	3	3	4	5	0
1	1	↗	↗	4	4	5	0	1
2	2	↗	↗	5	5	0	1	2
3	3	↗	↗	↗	1	2	□	3
4	4	↗	↗	↗	2	□	3	4
5	5	↗	↗	↗	□	3	4	5

Tabelle 4.3: Reale Aufteilung der Rechenlast. Das Zeichen □ symbolisiert einen wartenden Rechenknoten, das Zeichen ↗ bedeutet, daß die Koordinaten ohne Berechnung weitergereicht werden (Zyklusschritt 2a.). Nach rechts sind die sechs Zyklusschritte (1., 2.1.–2.5.), die innerhalb eines Berechnungszykluses ausgeführt werden, aufgetragen; nach unten ist die Nummer des Rechenknotens aufgetragen. Die Ziffern im Inneren der Tabelle bezeichnen die Indentität des „fremden“ Koordinatensatzes.

deren Identität nicht paßt. Durch diesen Schiebeschritt erhalten die Knoten 3 und 4 Koordinaten, mit denen eine Berechnung durchgeführt werden kann. Einzig Knoten 5 erhält keine neuen Koordinaten, da Knoten 0 noch mit der Berechnung beschäftigt ist (wie in Abschnitt 4.4.3.1 angedeutet, ist die Transferzeit wesentlich kürzer als die Berechnungszeit). Daher erlischt nun die busy-LED von Knoten 5 (Symbol □), während alle anderen LEDs brennen und aktive Knoten anzeigen. Erwähnenswert ist noch, warum Knoten 3 den Schiebeschritt 2.3.I. ausführen kann, obwohl Knoten 2 noch nicht zur Aufnahme neuer Daten bereit ist (da er noch rechnet). Dies ist nur möglich, weil ein in allen Netzwerktransputern parallel laufender Pufferprozeß *Buffer* (siehe Abbildung 4.9) die Koordinaten vorübergehend aufnimmt. Am Ende von Zyklusschritt 2.3.II. schieben alle aktiven Knoten ihre „fremden“ Koordinaten im Uhrzeigersinn weiter.
Dadurch erhält Knoten 5 von Knoten 0 den Koordinatensatz Nummer 3 und beginnt seine Berechnungen. Knoten 4 hat von Knoten 5 keine neuen Koordinaten erhalten und kann daher in Zyklusschritt 2.4. keine Berechnungen ausführen; seine busy-LED wird dunkel. Am Ende von Zyklusschritt 2.4. schieben alle aktiven Knoten ihre „fremden“ Koordinaten im Uhrzeigersinn weiter.
Die Vorgänge in Zyklusschritt 2.5. sind identisch zu denen in Zyklusschritt 2.4., mit dem einzigen Unterschied, daß während Zyklusschritt 2.5. Knoten 3 inaktiv ist.
Mit dem Ende von Zyklusschritt 2.5. erhalten alle Knoten wieder ihre eigenen Koordinaten zurück, die aber nicht gebraucht werden (da sie schon lokal vorhanden sind). Sie werden daher gelöscht.

Soweit es den Transfer von Koordinaten anbelangt, ist der Berechnungszyklus nun beendet (die Besprechung des Krafttransfers erfolgt in Abschnitt 4.4.3.5); ein neuer Zyklus schließt sich mit Zyklusschritt 1. an. Am „Wandern“ der dunklen busy-LED kann man erkennen, in welchem Stadium des Berechnungszykluses sich das Programm gerade befindet bzw. abschätzen, wieviel Zeit noch bis zum Ende des Zykluses gebraucht wird.

Bei einer *ungeraden* Zahl von Transputern läßt sich eine bessere Aufteilung der Rechenlast auf die Transputer erreichen, als im soeben besprochenen Fall der *geraden* Transputerzahl (bei den Überlegungen in diesem Paragraphen wird real vorhandener overhead außer Acht gelassen). Betrachten wir dazu zuerst die Zeit, die *ein* Transputer zur Berechnung eines Berechnungszykluses bei einem Molekül aus N Atomen benötigt:

$$t_1 = \frac{N(N-1)}{2} \cdot T_0 \quad . \tag{4.5}$$

Dies ist die Zeit T_0, die für die Berechnung einer (Coulomb-) Paarwechselwirkung benötigt wird (in unserem Fall ergibt sich T_0 zu $T_0 = 5.15 \cdot 10^{-5}$)[16], multipliziert mit der Zahl dieser Paare (siehe Abschnitt 4.2). Im Idealfall erwartet man beim Einsatz vom M Transputern für dieselbe Aufgabe eine Ausführungszeit von

$$t_M^{\mathrm{ideal}} = t_1/M \quad . \tag{4.6}$$

Dies entspricht einer Beschleunigung (⇒G) $B_M^{\mathrm{ideal}} = M$ und einer Effizienz (⇒G) $E^{\mathrm{ideal}} = 1$ (100%). Im Fall einer *ungeraden* Transputerzahl M kommt man mit dem von uns verwendeten und im Abschnitt 4.4.3 beschriebenen Algorithmus diesen Idealwerten recht nahe. Anhand einer Tabelle ähnlich Tabelle 4.2 kann man sich schnell klar machen, daß M Zyklusschritte für einen Berechnungszyklus erforderlich sind, wobei in jedem Zyklusschritt *alle* Transputer arbeiten. Falls man annimmt, daß N ein Vielfaches von M ist[17], so werden jedem Transputer N/M Atome zugeteilt. Im Zyklusschritt 1 (siehe Seite 54) werden die Wechselwirkungen der „eigenen“ Atome untereinander berechnet. Es sind dies $\frac{N}{M} \cdot \left(\frac{N}{M} - 1\right) \cdot 1/2$ Paarwechselwirkungen. Somit beträgt die Zeit t_M^{odd} für einen Berechnungszyklus

$$M \text{ ungerade: } t_M^{\mathrm{odd}} = \underbrace{\frac{N}{M} \cdot \left(\frac{N}{M} - 1\right) \cdot 1/2 \cdot T_0}_{\text{Zyklusschritt 1}} + \underbrace{\overset{\substack{\text{Zyklusschritte}\\ \Downarrow}}{\frac{M-1}{2}} \cdot \overset{\substack{\text{Paare}\\ \Downarrow}}{\left(\frac{N}{M}\right)^2}}_{\text{andere Zyklusschritte}} \cdot T_0 \quad . \tag{4.7}$$

---

[16] Diese Zahl wurde aus den Laufzeitwerten für einen T800 ($M = 1$) und einen Molekülausschnitt von 3634 Atomen ($N = 3634$) (siehe Tabelle 4.4) unter Benutzung von Gleichung 4.7, bzw. Gleichung 4.5 errechnet.

[17] Falls dies nicht der Fall ist, so ist in den Formeln für N der nächstgrößere Wert einzusetzen, der durch M teilbar ist. Die Rechenzeiten verlängern sich dadurch (vor allem für $N \gg M$) kaum.

Dies läßt sich zu $t_M^{\mathrm{odd}} = t_M^{\mathrm{ideal}}$ vereinfachen, was auf die maximal erreichbare Beschleunigung

$$B_M^{\mathrm{odd}} = M \tag{4.8}$$

und die maximal erreichbare Effizienz

$$E^{\mathrm{odd}} = 1 \tag{4.9}$$

führt.

Im Falle einer *geraden* Zahl von Transputern lautet der entsprechende Ausdruck

$$M \text{ gerade: } t_M^{\mathrm{even}} = \underbrace{\frac{N}{M} \cdot \left(\frac{N}{M} - 1\right) \cdot 1/2 \cdot T_0}_{\text{Zyklusschritt 1}} + \underbrace{\overset{\substack{\text{Zyklusschritte} \\ \Downarrow}}{\frac{M}{2}} \cdot \overset{\substack{\text{Paare} \\ \Downarrow}}{\left(\frac{N}{M}\right)^2}}_{\text{andere Zyklusschritte}} \cdot T_0 =$$

$$\frac{N\,(N-1)}{2M} \left(1 + \frac{1}{M - \frac{M}{N}}\right) \cdot T_0 \quad . \tag{4.10}$$

Im Grenzfall $\frac{M}{N} \to 0$ ergibt sich für die maximal erreichbare Beschleunigung (⇒G)

$$B_M^{\mathrm{even}} = \frac{M^2}{M+1} \quad , \tag{4.11}$$

und die maximal erreichbare Effizienz (⇒G) errechnet sich zu

$$E^{\mathrm{even}} = \frac{M}{M+1} \quad . \tag{4.12}$$

Betrachtet man nun den Beschleunigungsunterschied B_M^{diff} zwischen M, M gerade, und $M-1$ Transputern

$$M \text{ gerade: } B_M^{\mathrm{diff}} = B_M^{\mathrm{even}} - B_{M-1}^{\mathrm{odd}} = \frac{1}{M+1} \quad , \tag{4.13}$$

so wird dieser schon für mittlere Transputerzahlen (etwa $M = 12$) vernachlässigbar klein. Da, wie in Abschnitt 4.3.3 beschrieben, bei uns stets nur geradzahlige Transputerzahlen auftreten können (sechs Transputer auf einer Platine), hat man sozusagen noch einen Rechenknoten frei. Dieser Knoten könnte in einer zukünftigen Version von EGOfür die Berechnung der Wasserstoffbrücken abgestellt werden.

4.4.3.5 Austausch von Kräften

Da ein bidirektionales (Hardware-) link zwischen zwei Transputern (⇒G) *zwei* unidirektionalen occam II channels entspricht, können im Ring Daten in beiden Richtungen ausgetauscht werden. In der einen Richtung kreisen über den einen occam II channel die Koordinatenpakete, während parallel dazu über den anderen occam II channel in die entgegengesetzte Richtung Kräfte versandt werden. Wir nennen den ersten Kanal Koord[18] und den zweiten Force.

Hier sei nun erklärt, wie die Kräfte den richtigen Knoten erreichen:
Jeder Transputer kann aus seiner eigenen Nummer und der Zahl der schon erfolgten Zyklusschritte 2. die Identität der „fremden" Koordinaten, die gerade in seinem Speicher stehen, berechnen. Nachdem die Wechselwirkungen berechnet wurden, müssen die Kräfte in umgekehrter Richtung und wegen Gleichung (2.2) mit umgekehrtem Vorzeichen an den Transputer mit dieser Identität geschickt werden.

Dazu wird der Kanal Force genutzt, da es bei Verwendung nur eines Kanals für Koordinaten und Kräfte zu deadlock-Situationen kommen könnte (siehe Abschnitt 4.4.3.9). Um dies zu vermeiden, wurde das Versenden der Kräfte so weit wie möglich von den Koordinatenmanipulationen entkoppelt. Während das Durchreichen der Koordinatenpakete synchron erfolgt (bei jedem Schiebevorgang synchronisiert*Synchronisation* sich der ganze Ring immer wieder neu, indem die ‚schnelleren' Knoten, welche ihre zugeteilte Rechenarbeit als erste beenden, auf die ‚langsameren' Knoten warten, was man sehr gut an den grünen busy-LEDs beobachten kann), geschieht das Versenden und Empfangen von Kraftdaten asynchron.

Sind alle Kräfte zu einem „fremden" Koordinatensatz berechnet, so wird ihnen die Nummer des Zieltransputers vorangestellt und die Kräfte werden vom Prozeß *Berechnung* zum Prozeß *Multiplexer* (siehe Abbildung 4.9) übertragen. Dieser reiht sie in den Kraftkanal ein (siehe Abschnitt 4.4.3.6) und schickt sie gegen den Uhrzeigersinn zum Zieltransputer auf den Weg. Die bei den dazwischenliegenden Transputern und beim Zieltransputer einlaufenden Kraftdaten werden vom Prozeß *Routing* verwaltet. Dieser Prozeß vergleicht die Zielnummer des Datenpakets mit der eigenen Nummer. Bei den dazwischenliegenden Transputern (Zielnummer und eigene Nummer stimmen nicht überein) schickt er die Kräfte über den *Multiplexer* weiter. Erst beim Zieltransputer gibt der *Routing*-Prozeß den Weg über den Kanal Force.buff zum Prozeß *Add.buffer* frei. Der *Add.buffer* addiert die auf diese Weise ankommenden Kräfte auf. Um den Integrationsschritt ausführen zu können, müssen aber *alle* Kräfte bekannt sein (insbesondere auch die Kräfte, die von anderen Knoten berechnet und versandt wurden); die Kräfte und die Koordinaten müssen synchronisiert werden, um dies sicherzustellen. Dazu wird von jedem *Berechnungs*prozeß, der seine Arbeit abgeschlossen hat, ein *ready-tag* ($\Rightarrow$G) in den Kraftkanal eingespeist. Der *Routing*-Prozeß (siehe Abschnitt 4.4.3.6) zählt diese tags und reicht sie weiter (nur die „eigenen" tags werden nicht weitergereicht, um eine sonst endlose Schleife zu vermeiden). Sind so viele tags gezählt worden, wie Transputer

[18] Namen von Variablen und Kanälen beziehen sich auf den Programmquelltext, Anhang C und sind im Druck hervorgehoben.

im Ring sind (bei ALTER EGO–12 sind das 12 tags), so ist sichergestellt, daß alle Transputer die Kraftberechnung beendet haben und alle „fremden“ Kräfte nun bekannt sind. Daraufhin schickt der *Routing*-Prozeß ein Zeichen (tag (⇒G)) zum *Add.buffer*, welcher dadurch alle aufsummierten Kräfte über den Kanal Force.from zum Prozeß *Berechnung* schickt. Dieser kann die „fremden“ Kräfte (rechter Term in Gleichung (4.4)) zu seinen „eigenen“ (linker Term in Gleichung (4.4)) addieren und erhält so die Gesamtkraft F_i auf jedes einzelne „eigene“ Atom. Anschließend erfolgt der Integrationsschritt.

4.4.3.6 T800-Rechenknoten (Modul *Program*)

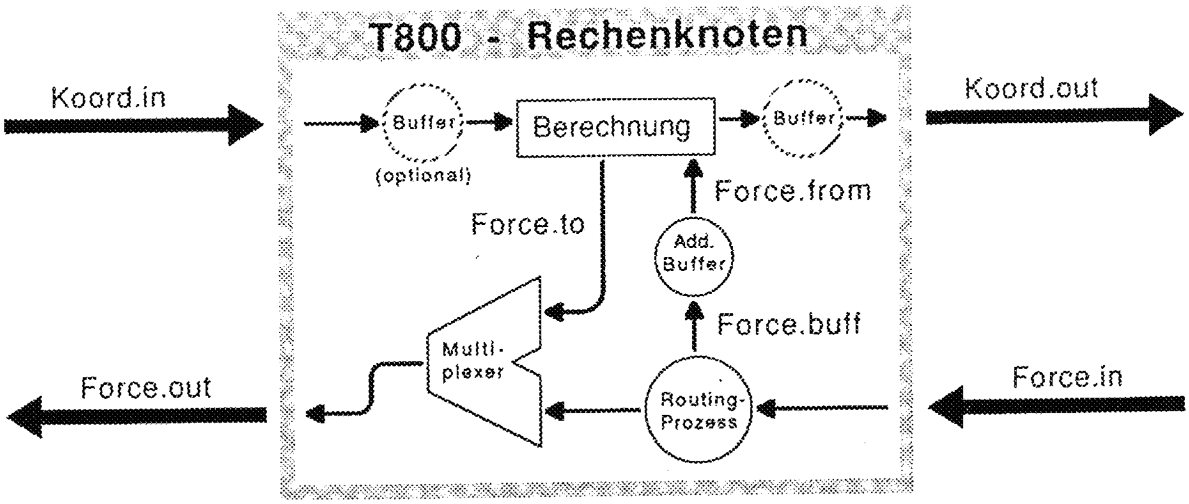


Abbildung 4.9: T800 Prozeßstruktur.

Wir wollen nun die auf den Netzwerktransputern ablaufenden Parallelprozesse genauer beschreiben. Wie in Abbildung 4.9 zu sehen, laufen auf jedem T800 im Modul *Program* fünf Prozesse ab.

- Der *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 II-Kanal an den Prozeß *Add.buffer* geleitet. Im anderen Fall passieren sie einen *Multiplexer* und erreichen schließlich den nächsten Transputer (⇒G).

- Der Prozeß *Add.buffer*, der die Kräfte so lange zwischenspeichert und aufaddiert[19], bis der Prozeß *Berechnung* bereit ist, sie weiterzuverarbeiten. Dieser Puffer ist wichtig für die Vermeidung von deadlock-Situationen (siehe Abschnitt 4.4.3.9).

- Der Prozeß *Multiplexer* faßt die Kraftdaten aus dem *Routing*-Prozeß und die neu berechneten Kräfte zusammen, um sie an den folgenden Transputer zu übermitteln. Dabei hat das routing gegenüber dem Neueinspeisen Vorrang, um eine Überlastung des Kraftkanals zu verhindern.
- Der Prozeß *Berechnung* schließlich führt die eigentliche Dynamiksimulation aus und verarbeitet sowohl die Kraftdaten als auch die Koordinatensätze von den anderen Transputern. Auf ihn wird im Abschnitt 4.4.3.6.1 ausführlich eingegangen.
- Auch die Koordinaten werden gepuffert: der Prozeß *Buffer* leistet dies.

Zusätzlich können auch noch weitere 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 (in EGOrealisiert).

4.4.3.6.1 Flußdiagramm T800 Betrachten wir den Prozeß *Berechnung*, der auf jedem T800 abläuft, etwas näher (Abbildung 4.10):

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. Dies wurde schon im Abschnitt 4.4.3.3 ausführlich beschrieben. Ein spezielles Umschaltsignal (die Übertragung des Integrationszeitschritts) steuert den Übergang in die eigentliche Berechnungsphase. Dieser Programmteil terminiert nicht, doch kann der host-Transputer (T414) ein reset-Signal auslösen, welches das gesamte Netzwerk neu startet.

Die Berechnung selbst ist ein sequentielles Programm, das aber über die eingezeichneten Kanäle (Abbildung 4.9) 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, die zu den „eigenen“ Atomen passen, besitzen. Dieser Teil unterscheidet

[19]Hierfür ist ein eigenes Speicherfeld nötig, da occam II-Prozesse keinen gemeinsamen Speicher (shared memory) kennen. Die Zusammenführung „fremder“ und „eigener“ Kräfte erfolgt, wenn alle Berechnungen abgeschlossen sind, über den Kanal Force.from.

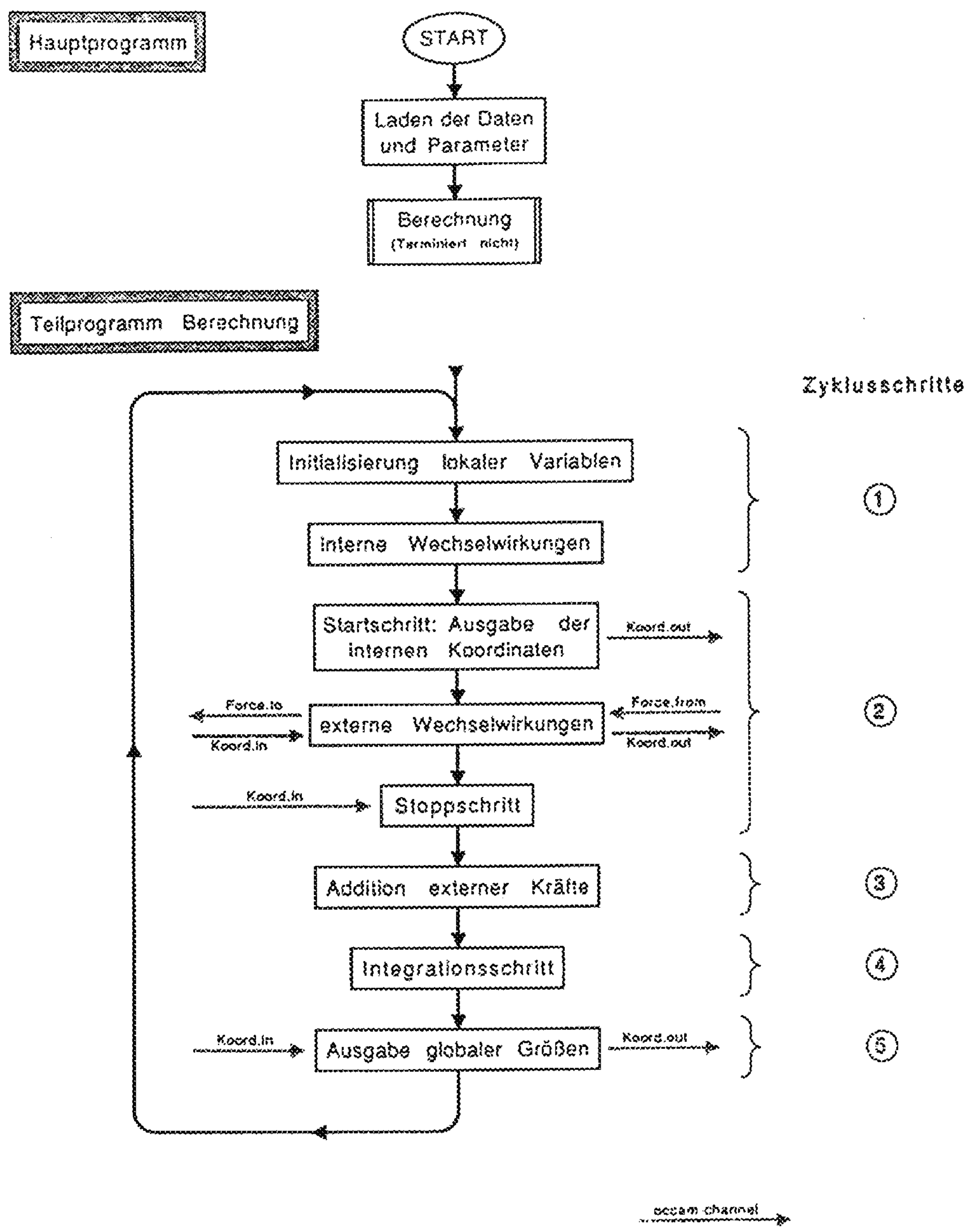


Abbildung 4.10: Flußdiagramm eines T800-Rechenknotens.

sich kaum von den herkömmlichen, sequentiellen Dynamikprogrammen. Der folgende *Startschritt* ist neu: der Transputer reicht „seine“ Koordinaten an seinen rechten Nachbarn weiter. Wie im Abschnitt 4.4.3.4 erläutert, werden zur Berechnung der externen Wechselwirkungen nun die Koordinaten vom linken Nachbarn verwendet. Die Elektrostatikberechnung etwa ist nun einfach eine Doppelschleife (unter Berücksichtigung der Ausschlußlisten, vgl. Seite 7) über alle „eigenen“ und alle „fremden“ Atome.

Die hier berechneten Kräfte werden zum einen auf die lokalen Kraftakkumulatoren addiert, zum anderen werden sie mit umgekehrtem Vorzeichen (siehe Gleichung (4.2)) an den linken Nachbarn zurückgegeben, so daß dieser sie ebenfalls auf seine Kraftakkumulatoren addieren kann. Dann werden die „fremden" Koordinaten an den rechten Nachbarn weitergegeben, und der Vorgang 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").

Die Berechnung der anderen Wechselwirkungen ist nicht ganz so einfach:
Für die Bindungskräfte existieren in jedem Knoten zwei Listen der gebundenen Atome: eine Liste nur für die Wechselwirkung „eigener" Atome untereinander (interne Liste) und eine Liste für die Wechselwirkung mit „fremden" Atomen (externe Liste).

Die interne Liste besitzt für jedes eigene Atom einen Eintrag, in dem die an dieses Atom gebundenen „eigenen" Atome aufgelistet sind. Diese Liste kann in Zyklusschritt 1. ohne Schwierigkeiten abgearbeitet werden. Die externe Liste ist nach „fremden" Atomen geordnet und besitzt zu jedem „fremden" Atom, das an „eigene" Atome gebunden ist, Einträge, mit welchen Atomen es gebunden ist. Bei jedem Satz „fremder" Koordinaten wird überprüft, welche Bindungen bestehen. Diese werden dann ausgerechnet. Die Kräfte werden gemeinsam mit denen aus der Elektrostatikberechnung verschickt.
An den anderen Wechselwirkungen sind mehr als zwei Atome beteiligt, so daß mindestens ein Atom zwischengespeichert werden muß. Wieder sind die Wechselwirkungen über interne und externe (diese enthält je Eintrag mindestens ein „fremdes" Atom) Listen definiert. Die Auswertung der internen Listen bereitet keine Schwierigkeit. Bei jedem Satz „fremder" Koordinaten wird nun überprüft, welche Atome in den externen Listen (auch diese Listen sind geordnet) vorkommen. Deren Koordinaten werden dann zwischengespeichert (einschließlich der Nummer des Transputers, von dem sie stammen). Hierbei ist es sehr vorteilhaft, wenn die Zuteilung der Atome zu den Transputern gruppenweise (Seitenketten, Aminosäuren, etc.) erfolgte, da dann die externen Listen <u>sehr</u> kurz werden (typisch 10–50 Einträge je Transputer bei 12 000 Atomen und 12 Transputern), was Speicherplatz und Suchzeit spart.

Die Berechnung erfolgt erst, wenn alle „fremden" Atome „vorbeigezogen" sind, da erst dann sichergestellt ist, daß alle Referenzen der Listen befriedigt sind. Die Kräfte werden dann einzeln zu den Zieltransputern geschickt.

Da nun die Gesamtkraft auf jedes interne Atom bekannt ist, können in einem Integrationsschritt (nach *Verlet*, [67], siehe Gleichung (2.13)) 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. Dazu addiert jeder Netzwerktransputer zu den Energiewerten, die er über link 0 empfängt, seine eigenen Beiträge hinzu und gibt die neuen Werte an den Nachbarn weiter. Begonnen und beendet wird dieser Schritt vom Prozeß *Master* aus dem Modul *Controller*.

4.4.3.7 T414-Rechenknoten (Modul *Controller*)

Im folgenden Abschnitt werden die Überwachungsfunktionen des Moduls *Controller* skizziert. In Abbildung 4.11 erkennen wir folgende parallele Prozesse:

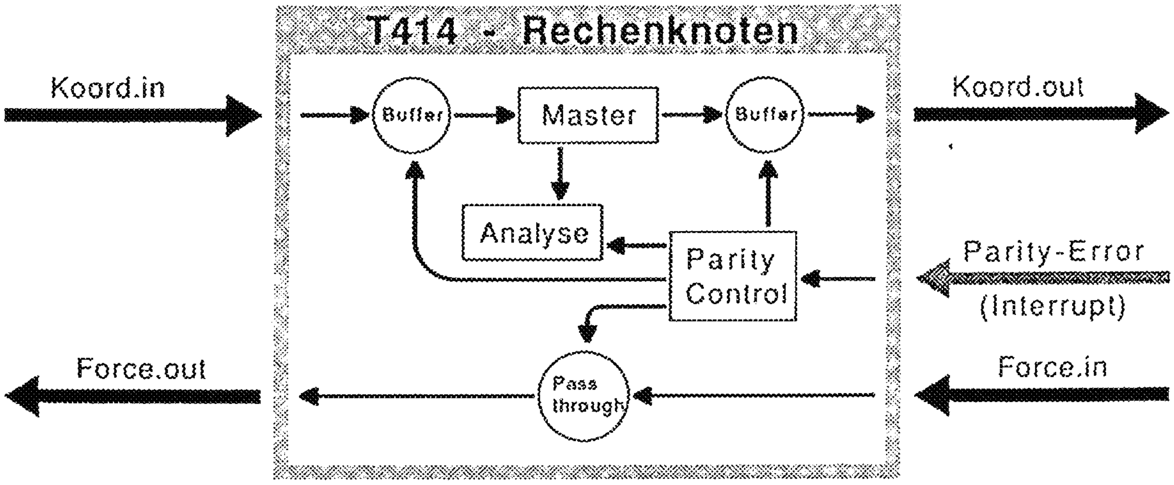


Abbildung 4.11: T414 Prozeßstruktur.

- Ein *Parity*-Prozeß überwacht den Interrupteingang des T414, der mit dem parity error-Ausgang des Netzwerkes verbunden ist. Tritt irgendwo ein parity-Fehler (⇒G) auf, so teilt dies der Prozeß auch allen anderen Prozessen mit (entweder direkt über einen Kanal oder über ein spezielles *stop tag* (⇒G), das der linke *Buffer* einspeist), die daraufhin terminieren. Dies ist wichtig, damit keine fehlerhaften Daten in den Prozeß *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 (⇒G) ist noch nicht alles verloren: ein Aufsetzen auf den letzten gültig berechneten Daten ist möglich. Dies war auch ausschlaggebend bei der Abwägung parity check (⇒G) oder error-detection-and-correction: was in Software geht, muß man nicht in Hardware machen.

- Der *Pass through*-Prozeß schließt einfach den Kraftkanal vom Eingang zum Ausgang kurz, denn Kraftdaten werden hier nicht benötigt.

- 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 genau dann, wenn dessen DMA-Maschine (⇒G) soeben über das link Werte an den T414 überträgt. Da die Nachricht des parity-Fehlers (⇒G) den *Master* oder die *Analyse* erst nach einer gewissen Verzögerung erreicht, könnte es ohne Puffer schon zu spät sein: fehlerhafte Daten stehen im restart file! Gerade dies verhindert der Eingangspuffer (im Abbildung 4.11 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 auch dann noch einen restart zu ermöglichen, benutzen die Puffer spezielle Ein-/Ausgaberoutinen, die jederzeit unterbrochen werden können. Dazu ist aber eine Wandlung des Protokolls nötig, die ebenfalls von den Puffern geleistet wird.

- Der *Master*-Prozeß hat alle Fäden in der Hand:

  Er zählt die Umläufe der Koordinaten, reicht sie gegebenenfalls zur *Analyse* weiter, fragt globale Energiewerte ab und hält den Benutzer über den Bildschirm auf dem Laufenden.

- Das *Analyse*-Programm in seiner einfachsten Version schließlich speichert die berechnete Trajektorie ab, bestimmt die Rechenzeit und hält die restart files auf dem neuesten Stand.

Eine zusätzliche Pufferung der Datenübergabe an das *Analyse*-Programm entkoppelt diese und die eigentliche Berechnung auf den T800 (Modul *Program*) weiter voneinander, so daß beide ohne gegenseitige Behinderung parallel ablaufen können.

4.4.3.8 `SEQ` versus `PAR`

Hier drängt sich nun die Frage auf, warum diese grobe Art der Parallelisierung (auf jedem Transputer (⇒G) läuft ein im wesentlichen sequentielles Programm) gewählt wurde. Bei strikter Anlehnung an die Natur würde man für jedes Atom einen (parallelen) Prozeß kreïeren und derartige Prozesse dann auf die verfügbaren Transputer verteilen. Dies würde das Programm vereinheitlichen (keine „eigenen“ und „fremden“ Atome mehr), brächte aber einen Geschwindigkeitsverlust, wie im folgenden aufgezeigt:

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

2. Die 4 KByte on-chip RAM (⇒G) stellen eine wertvolle Resource dar, da sie einen vier mal schnelleren Zugriff erlauben, als ihn der externe Speicher gestattet. Bei nur einem Prozeß je Transputer kann dieser *alle* seine skalaren Variablen im on-chip RAM ablegen und auf sie mit maximaler Geschwindigkeit zugreifen. Bei mehreren (10–100) parallelen Prozessen je Transputer, die alle ihren eigenen workspace (⇒G) benötigen, finden nur die Variablen sehr weniger Prozesse im on-chip RAM (⇒G) Platz; kostbare Zeit verstreicht ungenutzt beim Zugriff auf das externe DRAM (⇒G).

3. Ein Transputer für sich ist ein sequentieller Rechner. Daher kann von den vielen parallelen Prozessen 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, wie folgende Überlegung zeigt:

   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 solange zwischengespeichert werden, bis alle externen Atome an dem Prozeß vorbeigekommen sind (siehe Abschnitt 4.4.3.6.1). Dies wiederum kann für alle Wechselwirkungen (für Dihedral- und Extraplanarwechselwirkung muß die Koordinateninformation von je *zwei* zusätzlichen Atomen zwischengespeichert werden) 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 für etwa 100 Atome zusätzlicher Speicherplatz bereitgehalten werden muß. Dies ist nur 1/15 des Bedarfs der vollständig parallelen Lösung.

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

4.4.3.9 Deadlock-Problem

Als völlig neue Qualität eines parallelen Programms ist die Möglichkeit eines *deadlocks*deadlock anzusehen: parallele Programmteile blockieren sich gegenseitig und jegliche Aktivität im Netzwerk kommt zum Erliegen. Bei ungeschickter Programmierung kann eine Situation wie in Abbildung 4.12 auftreten:
keiner der beiden parallelen Prozesse in Abbildung 4.12 kann hier weiterarbeiten, weil

```
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
```

Abbildung 4.12: Verklemmung zweier Prozesse: deadlock

der eine jeweils auf den anderen wartet. Und genau dieser Fall kann auch im Programm auftreten, falls der Pufferprozeß *Add.buffer* zur Kraftaddition fehlt; nur ist die Lage dort deutlich komplizierter. Ohne auf Details einzugehen: es liegt daran, daß der *routing*-Prozeß nicht mehr arbeiten kann, wenn er Kraftdaten zum Prozeß *Berechnung* schicken will, dieser aber gerade mit anderem beschäftigt ist. Und genau da greift der Puffer ein: er addiert die Kräfte so lange auf, bis die *Berechnung* zu ihrer Aufnahme bereit ist. Der *Routing*-Prozeß ist damit von ihnen befreit und kann ungehindert arbeiten. So wird hier die Gefahr eines deadlocks gebannt.

Auch die Verwendung nur eines gemeinsamen Kanals für Kräfte und Daten kann zu einer Verklemmung führen. Falls sich an einem Knoten ein Koordinatenpaket staut, behindert es bei Verwendung desselben Kanals für Kräfte und Koordinaten auch den Krafttransport. Dies kann dazu führen, daß ein Knoten keine Kräfte mehr abschicken kann, weil „alles verstopft ist“. Dadurch wird er aber in seinem Berechnungszyklus angehalten und kann keine neuen Koordinatenpakete mehr aufnehmen. Dies kann auf den zuerst erwähnten Prozeß zurückwirken und diesen ebenfalls anhalten — der deadlock ist da!

4.4.4 Weiterverarbeitung der Daten

Die im Laufe einer Dynamiksimulation anfallende Datenmenge ist sehr groß: bei 13 000 Atomen umfaßt ein Datensatz (aus nur einem Integrationsschritt) etwa ein halbes Megabyte. So fallen bei einer Simulation von 20 ps Dynamik schon 200 MByte Daten an, selbst wenn nur jeder 50. Integrationsschritt aufgezeichnet wird. Solche Datenmengen lassen sich mit dem PC weder speichern noch verarbeiten. Daher laufen auch auf dem IBM-PC zwei parallele Prozesse:

- Der server für das TDS, um diesem Zugriff auf Platte, Bildschirm und Tastatur zu gewähren.
- Kermit$_{\textit{Kermit}}$, ein Dateitransferprogramm.

Über den server schreibt das EXE *Controller* Teile der Trajektorie auf die PC-Platte. Die so entstehenden Files wirken gleichzeitig als Semaphoren (⇒G) für das parallel laufende Kermit-Programm (welches von einem Turbo-Pascal-Programm kontrolliert wird), das dann die files über eine normale Terminalleitung (V.24, 9600 Baud) an einen Großrechner (CONVEX C1) überträgt. Dort steht ausreichend Massenspeicherplatz und eine Bandstation zur Archivierung der Daten zur Verfügung. Außerdem werden die Daten dort in ein XPLOR-lesbares Format umgewandelt (vergleiche Abschnitt 4.7) und komprimiert.

Mit den so entstehenden Trajektorienfiles kann anschließend durch XPLOR „off-line“ eine Analyse vorgenommen werden. Die durchgeführten Auswertungen sind in Kapitel 3 besprochen.

4.5 Benchmark-Ergebnisse

Will man die Leistung zweier Computersysteme vergleichen, so benutzt man dafür benchmark-Programme. Bei diesen Programmen handelt es sich um 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[20] und MFLOPS[21] 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 4.5 und vergleiche C-Compiler (MD) und FORTRAN-Compiler (XPLOR)]. Es ist also bei benchmark-Werten stets Vorsicht geboten [61].

Für eine Anwendung wirklich aussagekräftige Werte über das Gespann Hardware und Software erhält man nur, wenn man die Ausführungszeiten des entsprechenden 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: manche Probleme lassen sich eben gut parallelisieren, andere Probleme nicht. Aus diesem Grund erscheinen in Tabelle 4.4 die Laufzeitangaben für echte Molekulardynamikrechnungen, wobei die Vergleichsprogramme (Tabelle 4.5) für die jeweiligen Computer optimiert sind.

	Alanin ($\Rightarrow$G), 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 4.4: Benchmark-Ergebnisse für Molekulardynamik. Angegeben ist jeweils die Rechenzeit für einen Dynamikschritt in Sekunden für Alanin ($\Rightarrow$G), PTI (Pancreatic Trypsin Inhibitor) und RC (Ausschnitt aus dem photosynthetischen Reaktionszentrum von *Rhodopseudomonas viridis*).

An dieser Tabelle sind einige Eigenschaften sehr schön zu erkennen:

- Die Rechenzeit steigt nahezu quadratisch mit der Zahl der Atome.

[20] **M**ega **I**nstructions **p**er **S**econd; ein grobes Maß für die Leistungsfähigkeit eines Prozessors.

[21] **M**ega **Fl**oating point **O**perations **p**er **S**econd; ein grobes Maß für die Gleitkommaleistungsfähigkeit eines Prozessors.

- Die Rechenzeit für die Elektrostatik- und vdW-Berechnungen beträgt zwischen 50% und 99% der gesamten Rechenzeit. Dieser Anteil wird umso größer, je mehr Atome das Molekül hat.
- Die Beschleunigung (⇒G) von 12 Transputern gegenüber einem Transputer 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 (⇒G) liegt mit in der Struktur des Programmes begründet: bei nur fünf Atomen je Transputer macht sich der Zusatzaufwand (*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). 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. Die Beschleunigung (⇒G) von 10.75 liegt außerdem sehr nahe am theoretisch möglichen Wert von 11.08 (der Wert 12 kann nicht erreicht werden, siehe Gleichung (4.11)). Dieses gute Skalierungsverhalten rechtfertigt auch eine Extrapolation von 12 Transputern auf 50 Transputer.Man erkennt aus Tabelle 4.5, daß

VAX 11–750	1 x T800	CONVEX C1		12 x T800	CRAY-XMP	50 x T800
XPLOR	EGO	MD [70]	XPLOR	EGO	XPLOR	EGO
2 Stunden (geschätzt)	339.7 s	350 s	147 s (hochgerechnet)	31.6 s	7.8 s (hochgerechnet)	7.6 s (extrapoliert)

Tabelle 4.5: Benchmark-Ergebnisse für Molekulardynamikrechnungen an einem Ausschnitt des Reaktionszentrums (RC, 3634 Atome). Angegeben ist jeweils die Rechenzeit in Sekunden für einen Dynamikschritt bei Berücksichtigung aller Wechselwirkungen. Einige Werte konnten nicht direkt gemessen werden, sondern mußten hochgerechnet werden, da der zur Verfügung stehende Speicher der aufgeführten Rechner für die Berechnung aller Wechselwirkungen mit XPLOR nicht ausreicht. Hochgerechnete Werte wurden von PTI auf das Reaktionszentrum extrapoliert.

zwischen einer Rechenzeit von 31.6 s auf 12 Transputern und einer Rechenzeit von 7.8 s auf einer CRAY-XMP (am Institut für Plasmaphysik (IPP) in Garching) nur noch ein Faktor 4 liegt! Dieser Faktor ist aber mit etwa 50 Transputern einzuholen.

Neben der reinen Maschinenleistung, die hier verglichen wurde, ist für den Endbenutzer noch eine andere Größe von entscheidender Bedeutung: die Interaktionszeit. Damit ist die reale Zeit gemeint, die der Benutzer auf die Ergebnisse seiner Rechnungen warten muß. Diese Interaktionszeit wird bei Mehrbenutzersystemen nicht nur von der Leistungsfähigkeit des Rechners, sondern in hohem Grad auch von der Zahl der Benutzer bestimmt. So können vom Abschicken eines fünf-Stunden-Auftrags an einer Rechenzentrums-CRAY bis zum Eintreffen der Ergebnisse Tage vergehen.

ALTER EGO–12 ist dagegen als single-user-System ausgelegt und stellt seine gesamte Leistung einem Benutzer zur Verfügung. Dadurch ist die effektive Leistung um ein Vielfaches höher, als die benchmark-Werte vermuten lassen. Oben erwähnter Auftrag wäre auf ALTER EGO–12 mit EGOin weniger als einem Tag fertig gerechnet. So konnte beispielsweise die Simulation von über 40 ps Dynamik von PTI unter Berücksichtigung aller Wechselwirkungen bequem über Nacht erfolgen. Dieselbe Simulation beansprucht auf dem stark ausgelasteten CONVEX C1 Vektorrechner unserer Gruppe einige Tage.

4.6 Kosten für einen Rechenknoten

Neben der Leistungsfähigkeit sind bei einem Rechner natürlich auch die Kosten ausschlaggebend. In Tabelle 4.6 findet sich daher eine Aufschlüsselung der anteiligen Kosten eines Rechenknotens.

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	42.35
Gesamt		1939.41

Tabelle 4.6: Kostenaufstellung für einen Rechenknoten. Die Preise stellen die anteiligen Kosten eines Rechenknotens bei einer Ausbaustufe von 60 Transputern (d.h. ein 19-Einschub) dar; die Mehrwertsteuer ist bereits enthalten.

Der Tabelle 4.6 entnimmt man, daß sich für etwa DM 2.000 pro Knoten ein Netzwerk aus 50 Transputern für DM 100 000 realisieren läßt. Extrapoliert man die vorher (in Abschnitt 4.5) aufgeführten benchmark-Werte, so kommt man zu dem Schluß, daß sich die Rechenleistung einer CRAY-XMP mit 50 Transputern in etwa erreichen läßt — d.h. zu einem Preis von DM 100 000. Insgesamt hat der Rechner mit 12 Transputern etwa DM 24 000 gekostet. Dazu kommen noch einmalige Aufwendungen für das Entwicklungssystem (TDS), den occam II-Compiler, das B004-board von INMOS und die Entwicklung des Prototypen von zusammen etwa DM 15 000. 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 (ITR-880-G9) sowie der Deutschen Forschungsgemeinschaft im Rahmen des Sonderforschungsbereiches SFB 143 „Primärprozesse der bakteriellen Photosynthese“ zur Verfügung gestellt.

4.7 Kompatibilität zu anderen Molekulardynamikprogrammen

Bei der Aufklärung des Zusammenhangs von Struktur und Wirkung von Makromolekülen ist die Dynamiksimulation nur ein Teil der Arbeit — wenngleich sie auch die meiste Computerzeit in Anspruch nimmt. Die durch solche Simulationen gewonnenen Daten müssen auch analysiert werden. Diese Aufgabe kann kein Computer übernehmen, denn dazu bedarf es auch der Intuition und Phantasie des Physikers. Wohl aber kann und muß der Computer dabei helfen. Entsprechende Analysefunktionen beinhalten bekannte Programme wie XPLOR [23] oder CHARMM [21]. Im Gegensatz zur Dynamikberechnung sind die Anforderungen, was die Rechenzeit anbelangt, gering, dagegen sind diese Programmteile, was den Programmcode anbelangt, sehr umfangreich, da sie sehr flexibel gehalten sein müssen, um den Physiker bei seiner Arbeit zu unterstützen und nicht zu behindern. So erscheint es nur natürlich, wenn die von EGObei einer Simulation gelieferten Daten von diesen bereits vorhandenen Analyseprogrammen weiterverarbeitet werden. Tatsächlich sind wir auch diesen Weg gegangen. Die in einem Internformat auf den host-Rechner (z.B. CONVEX C1) übertragenen Daten (siehe Abschnitt 4.4.4) werden dort in ein platzsparendes binäres Format umgewandelt (sog. .crd-File), das etwa von XPLOR als Trajektorien-file eingelesen werden kann. Der Umweg über das Internformat und die Wandlung auf dem host-Rechner wurden gewählt, weil die binäre Darstellung von host-Rechner zu host-Rechner variieren kann. Erfolgt dagegen die Wandlung in das binäre Format auf dem host-Rechner selbst, so ist die genaue Art der binären Darstellung ohne Belang. Damit steht nun die gesamte Palette an Analysemöglichkeiten zur Verfügung.

Auch auf Seite der Eingangsdaten ist Kompatibilität zu bestehenden Formaten wichtig, um problemlos alle vorhandenen Daten verwenden zu können. Wie zu Beginn des Abschnitts 4.4.3.2 beschrieben, wurde daher von uns ein eigenes Interfaceprogramm geschrieben, welches mehrere Daten-files in XPLOR-lesbarem Format [dazu gehören das

Koordinatenfile (.pdb), das Strukturfile (.psf) und die Parameterfiles] in eine folding-Struktur einliest, welche dann von EGOverarbeitet werden kann.

Damit sind die Schnittstellen wohldefiniert und mit den de-facto-Standards kompatibel.

In diesem Zusammenhang ist noch anzumerken, daß EGOüber keine eigene Kommandosprache verfügt; vielmehr sind die auszuführenden Aktionen in einem Steuerfold festzulegen. Damit ist klar, daß die von XPLOR gewohnten Kommandofiles nicht verwendet werden können. Dies ist kein gravierender Nachteil, da sich die meisten Elemente der Kommandosprache auf die Analyse beziehen und damit weiterverwendet werden können.

Es sind noch einige wichtige Punkte zu erwähnen, in denen sich EGOvon XPLOR unterscheidet. Erstens gibt es noch keine Möglichkeit, nur Teile des Moleküls zu simulieren und diese dazu mit einem stochastischen Rand zu versehen (*stochastic bounderies*). Zweitens ist die Simulation von Zufallskräften, wie sie für Langevin-Dynamik erforderlich ist, noch nicht implementiert. Drittens ist die Verwendung eines „cut off" (⇒G) bei den langreichweitigen Wechselwirkungen noch nicht implementiert. Viertens schließlich gibt es noch keine spezielle Repräsentation von Wasser. All dies liegt weniger an prinzipiellen Schwierigkeiten bei der Parallelisierung, sondern daran, daß ein Jahr, welches im Rahmen der vorliegenden Diplomarbeit zur Verfügung stand, für die Entwicklung von Software und Hardware eine sehr knapp bemessene Zeit ist. Spätere Versionen von EGOwerden diese Möglichkeiten bieten.

Erwähnt werden muß noch, daß die Berechnung der Wasserstoffbrücken *Wasserstoffbrücken* in dem neuen Parallelprogramm EGOnoch nicht enthalten ist. Dies hat drei Gründe:

- Wasserstoffbrücken entstehen und vergehen dynamisch während der Simulation; sie lassen sich also nicht (wie etwa die Atombindungen) in unveränderlichen Listen fassen. Daher sind stets alle möglichen Wasserstoffbrückenbindungen zu betrachten.

- An einer Bindung sind bis zu vier Atome (Donator, Wasserstoff, Akzeptor und Akzeptorantezedent) beteiligt. Deren Koordinaten, die über mehrere Transputer verstreut sein können, müssen erst gesammelt und zwischengespeichert werden, bevor die Berechnung ausgeführt werden kann.

- Die Energiebeiträge der Wasserstoffbrücken lassen sich bis zu einem gewissen Grad durch modifizierte Parametersätze in den anderen Wechselwirkungen berücksichtigen. Solche Parametersätze stehen zur Verfügung und wurden sowohl hier, als auch bei den Vergleichsrechnungen von H. Treutlein [64] und Ch. Niedermeier [51] verwendet.

Beim Reaktionszentrum etwa sind 1942 Donatoren und 1747 Akzeptoren zu berücksichtigen, wobei für jedes Donator-Akzeptor-Paar das Bestehen einer Wasserstoffbrücke untersucht werden muß. Dies bereitet bei der gegenwärtigen Parallelisierungsstrategie Speicherplatzprobleme. Für eine zukünftige Implementation erscheint es daher günstig, der Berechnung dieser Wechselwirkung einen eigenen Rechenknoten zu widmen, der sich nur damit beschäftigt und die Koordinaten aller hieran beteiligten Atome speichern kann. Die Berechnung der Wasserstoffbrückenenergie kann dann in der herkömmlichen, d.h. sequentiellen Weise erfolgen. Dieser Abzug eines Rechenknotens muß dabei nicht unbedingt zu einer anteiligen Verminderung der Rechenleistung führen; insbesondere dann nicht, wenn eine *gerade* Zahl von Rechenknoten zur Verfügung steht, wie auf Seite 59 erläutert[22]. Nachdem ein Knoten für die Berechnung von Wasserstoffbrücken abgestellt ist, verbleibt eine *ungerade* Zahl von Transputern für die Berechnung von Paarwechselwirkungen. Anwendung von Gleichung (4.8) zeigt, daß die maximale Beschleunigung (⇒G) in diesem Fall gleich der Zahl der Transputer (die an der Paarwechselwirkung arbeiten) ist, d.h. kein Transputer bleibt untätig.

[22]Um dies zu überprüfen, wurde vorübergehend ein Transputer (softwaremässig) kurzgeschlossen und das Programm EGOfür 11 Transputer neu compiliert. Die Rechenzeit für einen Berechnungszyklus des gesamten Reaktionszentrums unter Berücksichtigung aller Wechselwirkungen vergrößerte sich dabei von 387 s auf 395 s, die Beschleunigung (⇒G) verringerte sich von 10.75 auf 10.53 und die Effizienz erhöhte sich von etwa 89 % auf 96 %. Diese Werte liegen sehr nahe an den theoretischen Grenzwerten von 92% ($M = 12$, Gleichung 4.12) bzw. 100% ($M = 11$, Gleichung 4.9).

Kapitel 5

Zusammenfassung

In diesem Abschnitt sind die wichtigsten Ergebnisse der vorliegenden Arbeit kurz zusammengefaßt.
Ausgangspunkt ist der Wunsch, den Zusammenhang zwischen Struktur und Wirkung von Proteinen besser zu verstehen. Eine der dazu verwendeten Methoden ist die Molekulardynamiksimulation. Leider schränkt der enorm hohe Bedarf an Rechenleistung bei der Simulation großer Moleküle den Gebrauch dieses Werkzeuges stark ein. Hier versucht die vorliegende Diplomarbeit Abhilfe zu schaffen.

Zu diesem Zweck wurde ein neuartiger Parallelrechner entwickelt und aufgebaut, der die Rechenleistung heutiger Großrechner zu einem Hundertstel des bisherigen Preises liefert. Dazu wurde der Schaltplan für diesen Rechner entwickelt und an einem Prototypen getestet. Die nach eigenem Layout erstellten Leiterplatten wurden bestückt und der Parallelrechner erprobt.

Um auf diesem Rechner Molekulardynamiksimulationen durchführen zu können, mußte außerdem der Code vorhandener Simulationsprogramme vollständig neu geschrieben und parallelisiert werden. Dabei kam die neuartige Programmiersprache occam IIzur Anwendung.

Nachdem durch Testrechnungen sichergestellt war, daß dieses neue „Gespann" aus Hardware und Software zuverlässig richtige Ergebnisse liefert, konnte damit begonnen werden, eine Minimierung und Equilibrierung des gesamten Reaktionszentrums unter Berücksichtigung aller Wechselwirkungen, d.h. ohne cut-off (⇒G) bei den langreichweitigen Kräften, durchzuführen, um eine geeignete Ausgangsstruktur für weiterführende Simulationen zu erhalten. Die Equilibrierung konnte im zeitlichen Rahmen der vorliegenden Arbeit nicht mehr abgeschlossen werden. Trotzdem konnten mit den während der Minimierung erhaltenen Daten schon einige Analysen vorgenommen werden. Dazu zählt die Analyse des Verhaltens ausgewählter Energiebeiträge während der ersten Femtosekunden der Simulation, der zeitlichen Entwicklung der Gesamtenergie während des gesamten Simulationszeitraumes und des Verhaltens des Moleküls nach der Minimierung. Ferner wurde die Geschwindigkeitsverteilung zu drei verschiedenen Zeitpunkten untersucht und mit den gewählten Anfangsbedingungen in Beziehung gesetzt. Abschließend wurde gezeigt, daß die periodischen Schwankungen der Bindungsenergie von der besonderen Wahl der Anfangsbedingungen herrühren und Bindungsschwingungen zugeordnet werden können.

Schon diese erste anwendungsbezogene Rechnung, die Minimierung des Reaktionszentrumvon *Rhodopseudomonas viridis*, zeigt die Tauglichkeit der entwickelten Werkzeuge in der Praxis. Dafür war etwa ein Monat Rechenzeit erforderlich. Weiterführende Simulationen sind geplant und das hier entwickelte System aus Rechner und Programm wird nicht nur den allgemeinen Bedarf an Rechenzeit mildern, sondern auch Simulationen ermöglichen, die bisher nicht durchführbar waren.

Anhang A

Änderungen am B004-board von INMOS

Am B004-board von INMOS, einer Einsteckkarte mit T414-Transputer für IBM PCs, mußten einige Änderungen vorgenommen werden, damit diese Karte unsere Anforderungen erfüllt:

1. Änderung der link-Geschwindigkeit auf den beiden links zu Alter Ego–12 (link 1 und link 2) von 10 Mbit/s auf 20 Mbit/s, ohne die link-Geschwindigkeit auf link 0, das die Verbindung zum PC herstellt, zu verändern (bleibt auf 10 Mbit/s).
2. Herausführen des event pins des T414, das bisher unbenutzt war.

Punkt 1. ist normalerweise einfach durchzuführen: man muß nur den logischen Pegel an einem pin (A4) des Transputers ändern. Leider führte dies hier nicht zum gewünschten Erfolg, weil INMOS den T414 nicht mit den üblichen 5 MHz Taktfrequenz betreibt, sondern ihm 25 MHz zuführt (nicht dokumentiertes Feature!). Dadurch kann die PLL für die link-Anschlüsse nicht mehr mit getrennten Geschwindigkeiten für link 0 und die links 1, 2 und 3 arbeiten. Um dies zu ermöglichen, mußte der Takt aus dem 25 MHz Quarzoszillator durch 5 geteilt und die Beschaltung des T414 so geändert werden, daß seine PLL nunmehr ganz normal arbeitet. Die Schwierigkeit bei diesem Umbau lag vor allem darin, daß es sich um ein *undokumentiertes* Feature des T414 handelte; erst eine telefonische Auskunft von Herrn P. Eckelmann, INMOS, brachte Klarheit.

Punkt 2. wurde nötig, um *jederzeit* und *sofort* auf einen parity-Fehler reagieren zu können. Neben der Verwendung des event pins gäbe es prinzipiell noch die Möglichkeit, den Zustand der parity-Leitung zyklisch per Software abzufragen („polling“). Diese Alternative wurde, da sie ohne Hardwareänderung möglich ist, zuerst erprobt. Um die Reaktionszeit ausreichend kurz zu halten, muß aber eine sehr hohe Abtastfrequenz verwendet werden, was den Prozessor stark belastet und viel Rechenzeit verbraucht. Im praktischen Versuch stellte sich heraus, daß daraus eine nicht mehr tragbare Verlangsamung des Programmablaufs resultiert. Daher mußte der Interrupt verwendet werden. Da das event pin auf einer mittleren Lage des Multilayer-boards B004 fest auf Masse gelegt ist, mußte der T414 „angehoben“ werden (zwei Zwischensockel isolieren das event pin vom Masseanschluß und erlauben so einen externen Zugang). Das Entprellen und Ausfiltern von spikes erfolgt per Software.

Trotz all dieser Änderungen ist es immer noch nicht möglich, den T414 gegen einen T800 auszutauschen, da die niedrigste Taktfrequenz von letzterem 17.5 MHz ist, das B004-board aber für 15 MHz Arbeitsfrequenz ausgelegt ist.

Anhang B

Schaltpläne

Anhang C

Programmquelltext in occam II

Der Quelltext des Programms umfaßt mittlerweile mehr als ein halbes Megabyte, das sind ausgedruckt über 500 Seiten. Auf einen Abdruck des Programms in der Diplomarbeit wurde wegen dieses Umfangs verzichtet. Ein weiterer Grund, der gegen einen Abdruck spricht, ist, daß die Übersicht, die durch den folding editor am Bildschirm leicht zu realisieren ist, auf einem Papierausdruck nicht geschaffen werden kann. Es verwundert damit auch nicht, daß während der Erstellung des Programms nicht ein einziger Papierausdruck erfolgte.

Es wurde daher ein anderer Weg der Programmdokumentation beschritten: einigen Exemplaren dieser Diplomarbeit liegen Disketten im IBM-PC-Format bei, die den Quelltext in occam IIund die folding-Information enthalten. Sie können direkt mit dem folding editor gelesen werden.

Anhang D

Danksagung

Für das Thema, das, wie ich glaube, meinen Fähigkeiten und Neigungen optimal entgegenkommt, und die herzliche Betreuung, sowie die angenehme Arbeitsatmosphäre danke ich Herrn **Prof. Klaus Schulten**. Auch für das hohe Maß an Vertrauen, das Herr Schulten mir entgegenbrachte, als er für diese Diplomarbeit den gewöhnlichen Rahmen weit übersteigende Mittel zur Verfügung stellte, möchte ich mich bedanken. Er zeigte nie Ungeduld oder Nervosität, auch wenn das Projekt auf der Stelle zu treten schien oder es kleinere Rückschläge gab. Vielmehr verstand er es, die Motivation stets aufs neue anzufachen.

Danken möchte ich auch sehr **Helmut Grubmüller**, dessen hervorragende Unterstützung wesentlich zum Gelingen dieses Projekts beigetragen hat. Mit dem eigenständigen Design der Leiterplatten nahm er sehr viel Arbeit auf sich, was es mir ermöglichte, die Entwicklung des Programms voranzutreiben. In vielen konstruktiven Diskussionen wurde so manches Problem, nicht nur aus dem Hardwarebereich, gelöst. Ohne seinen hochmotivierten Arbeitseinsatz wäre sicher nicht nur der Zeitplan nicht einzuhalten gewesen.

Außerdem möchte ich mich bei meiner Mutter **Frieda Heller** für das liebevolle Verständnis und die moralische Stärkung sowie die materielle Unterstützung recht herzlich bedanken.

Dank gebührt **Eva Bauernfeind** für ihre Unterstützung während der „heißen Phase“ der Diplomarbeit.

Ferner darf **Andreas Windemuth** nicht unerwähnt bleiben, an dessen C-Version von CHARMM ich mich orientieren konnte, und der mir so manchen wertvollen Hinweis (Datenformate, Testdatensätze) gab.

Bei **Markus Tesch**, der auch das Konvertierungsprogramm aus dem EGOInternformat in das binäre XPLOR Format geschrieben hat, und bei **Christoph Niedermeier** bedanke ich mich für die Hilfe beim Umgang mit XPLOR und PLTI[1], sowie für wertvolle Erfahrungswerte aus dem Bereich der Molekulardynamiksimulation.

Michael Großjean erleichterte mir den Einstieg in das Betriebssystem VMS auf unserem Gruppenrechner, einer VAX 11/750; **Wolfgang Bauer** unterstützte mich auf einer Leiter in schwindelerregender Höhe beim Verlegen der V.24-Datenleitung, über die unser Rechner schließlich seine Daten an den CONVEX C1 Großrechner übermittelte, und **Herbert Treutlein** hatte stets ein offenes Ohr für meine Fragen aus dem Bereich der Molekulardynamik.

Paul Tavan half, an schwierigen Stellen der Arbeit mehr Klarheit in die Darstellung zu bringen.

Beim Elektroniklabor von **E22** bedanke ich mich herzlich für die unbürokratische Unterstützung (meistens fehlt ein Widerstand um 22 Uhr).

[1]mit diesem Zeichenprogramm wurden die xy-Diagramme dieser Arbeit angefertigt

Anhang E

Zitate

„Sie arbeiten ja gar nicht, Sie essen ja!“

Prof. Schulten als er gegen Abend ins Zimmer schaute und ich, am Rechner sitzend, einige Brote (für das ausgefallene Mittagessen) vertilgte (Mitte Februar 1988).

„Bin ich Dein on-line manual?“

Robert Bittl auf meine Frage, wie man unter VMS ein subdirectory erzeugt (im Oktober 1987).

„Wie heißt das Mädchen?“

Prof. Schulten, der nur mit einem halben Ohr zugehört hatte, bei unserem ersten persönlichen Kontakt, nachdem ich ihm erzählt hatte, daß ich wegen eines *PDP11-Rechners* die Quantenmechanikvorlesung (bei ihm) geschwänzt hatte, was mir 2 Semester zusätzliche Studiendauer einbrachte (1985).

„Das kannst Du also auch!“

F. Gebhard in einem privaten Gespräch (Oktober 1987).

„Da müssen noch mehr Lämpchen hin, und die müssen flackern!“

Prof. Schulten bei Besichtigung des Prototypen angesichts einiger LED-Funktionsanzeigen (März 1988).

> „Der Heller ißt immer so komische Sachen; vielleicht sollte ich ihm etwas Gemüse schicken?“

Prof. Schulten zu Helmut Grubmüller, als der ihm mitteilte, daß ich noch weitere 3 Tage im Bett bleiben müßte (ich lag bereits seit 2 Wochen mit eitriger Mandelentzündung im Bett). Mit den „komischen Sachen“ meinte er Nougat und Marzipan, die ab und zu ein Abendessen ersetzen müssen (13.6.1988).

> „So hat Klaus auch angefangen.“

Frau Schulten, als ich ihr einen Teil meines Programms auf Englisch erklärte und bei der Winkelwechselwirkung immerzu von der angel-Wechselwirkung sprach und sie mich partout nicht verstand. Angles sind halt keine angels (Juni 1988).

> „Die Ergebnisse sind zwar völlig falsch, aber—die 12 Transputer rechnen parallel!!“

Der Autor, als EGOdas erste mal parallel die Elektrostatik berechnete. Prof. Schulten bezeichnete dies dann als das ‚Bonmot des Tages‘ (11.8.1988).

Glossar

Alanin: Dieses Molekül wurde in unserer Biophysik-Arbeitsgruppe im Rechner „zusammengebaut“. Es handelt sich dabei um eine α-Helix mit 10 Alanin-Seitengruppen; insgesamt 66 Atome. Aufgrund seiner geringen Größe ist es besonders für die schnelle Ausführung von Testrechnungen geeignet. Die Struktur bleibt auch bei Simulationen über längere Zeiträume stabil.

Backplane: Die einzelnen Platinen, aus denen der Rechner ALTER EGO–12 aufgebaut ist, sind in einem sog. Einschubgehäuse untergebracht. Die Verbindung der Platinen untereinander (z.B. links) und mit der Umgebung (z.B. Stromversorgung) erfolgt über je zwei Stecker, die auf einer Längsseite der Platine angebracht sind. Passend zu diesen Steckern sind im Gehäuse Buchsen vorhanden. Die Verbindung der Buchsen erfolgt nun nicht mit Drähten (diese Art der Verbindung wäre in der Herstellung sehr aufwendig, da Handarbeit, und damit auch sehr fehleranfällig), sondern über Leiterbahnen auf einer Platine. Weil diese Platine auf der Rückseite des Gehäuses sitzt, nennt man sie *backplane*.

Beschleunigung: t_1 sei die Zeit, die von *einem* Rechner für die Lösung einer gegebenen Aufgabe benötigt wird; t_M die Zeit, die M Rechner für die Lösung derselben Aufgabe brauchen. Dann ist die Beschleunigung B eines Transputernetzwerkes durch $B = t_1/t_M$ gegeben.

Bit: Kleinste Informationseinheit; sie kann nur die Werte „wahr“ oder „falsch“ annehmen. In der Technik ist sie meist durch die Spannungspegel 0 V und 5 V repräsentiert.

Boot: Unter *boot* versteht man das Laden eines Programms in einen Rechner nach Einschalten der Versorgungsspannung. Dieses Programm übernimmt dann die weitere Kontrolle. Je nach Herkunft dieses Programms spricht man von „boot from ROM (⇒G)“ oder „boot from link“.

Byte: Ein byte ist aus 8 bit (⇒G) aufgebaut. Diese werden oft gemeinsam interpretiert (z.B. lassen sich mit einem byte die Zahlen von 0 bis 255 darstellen).

Cache: Das cache ist ein besonders schneller Speicher, der zwischen CPU und Hauptspeicher geschaltet wird. Eine spezielle Hardware sorgt dafür, daß die Inhalte von cache und Hauptspeicher einander angeglichen werden, und daß die CPU im cache möglichst oft die gerade benötigten Daten vorfindet; der Programmierer kann dies nicht steuern.

Durch die Zwischenschaltung wirkt das cache wie ein Puffer und erlaubt der CPU wesentlich schnellere Speicherzugriffe als der langsamere Hauptspeicher. Eine besonders schnelle Verarbeitung erreicht man bei Ausführung einer Programmschleife, deren Maschinencode vollständig im cache untergebracht werden kann. Sprünge dagegen verlangsamen oft die Programmausführung, weil das Sprungziel sehr oft außerhalb des cache liegt. In einem solchen Fall muß die CPU warten, bis das cache vom Hauptspeicher mit den benötigten Daten nachgeladen wurde.

Man unterscheidet zwischen Programm- und Daten-cache. Schnelle Prozessoren besitzen beide.

Cut-off: Bei der Molekulardynamiksimulation erfordert die Berechnung der nicht-Bindungswechselwirkungen, wie van der Waals-Wechselwirkung und elektrostatische Wechselwirkung, prinzipiell eine Summation über alle Atompaare. Unter der Annahme, daß weit entfernt Atome nur noch sehr kleine Beiträge zu dieser Summe liefern, können diese ab einer bestimmten Entfernung vernachlässigt werden. Dieses „Abschneiden“ der Wechselwirkung bezeichnet man als *cut-off*. Eine Verbesserung der Näherung erreicht man, indem man nicht abrupt abschneidet, sondern eine weichere Abschneidefunktion verwendet. Weitere Verbesserungen sind möglich.

Durch Verwendung eines cut-off läßt sich die Rechenzeit erheblich (die genauen Werte hängen von der Gestalt des Proteins ab) reduzieren.

DMA-Maschine: Das Konzept des DMA, des **D**irect **M**emory **A**ccess, ist nicht neu. Immer wenn es gilt, ganze Datenblöcke, also Speicherbereiche mit konsekutiven Adressen, im Speicher zu verschieben (z.B. bei Grafikanwendungen), oder nach außen zu übertragen, reicht eigentlich ein einfacher Zähler, der die Speicheradressen inkrementiert, aus. Die CPU mit ihren arithmetischen und logischen Fähigkeiten wird dafür nicht benötigt. Falls man solche Transferaufgaben vom eigentlichen Programm abkoppeln kann, so kann man sie einer speziell dafür gebauten Hardware, der *DMA-Maschine*, übertragen. Solch ein Fall liegt bei der Datenübertragung auf Transputer-links vor: sowohl der sendende als auch der empfangende Prozeß geben die CPU frei, die damit anderen parallelen Prozessen zur Verfügung steht. Dafür übertragen unabhängige DMA-Maschinen die Daten von einem Transputer zum anderen. Diese greifen dabei *direkt* auf den Speicher zu (*DMA*), ohne die CPU zu belasten.

Effizienz: t_1 sei die Zeit, die von *einem* Rechner für die Lösung einer gegebenen Aufgabe benötigt wird; t_M die Zeit, die M Rechner für die Lösung derselben Aufgabe brauchen. Dann ist die Effizienz E eines Transputernetzwerkes durch $E = t_1/(t_M \cdot M)$ gegeben.

Parity: Normalerweise werden die Daten im Speicher eines Computers ohne Redundanz abgelegt. Man kann zwar so in einem Speicher gegebener Größe die meisten Informationen unterbringen, doch gibt es keine Möglichkeit, Fehler in den Daten, die durch Versagen des Speichers entstehen, zu erkennen oder gar zu korrigieren. Legt man besonderen Wert auf hohe Datensicherheit, so führt man ein gewisses Maß an Redundanz ein. Sehr häufig werden 8 bit zu einem byte zusammengefaßt und mit einem neunten *parity* bit versehen. In diesem neunten bit steckt die redundante Information: dieses bit speichert die binäre Quersumme modulo zwei aller bits des bytes. Dies ist die Information darüber, ob die Zahl der 1-bits im byte gerade oder ungerade ist, also die Parität. Beim Schreiben des bytes wird dessen Parität berechnet und im *parity* bit abgelegt.

Ändert nun eines der bits im byte durch einen Speicherfehler seinen Wert, so ändert sich auch die Parität des bytes. Dies kann beim Lesen des bytes überprüft werden: aus den bits des gelesenen bytes wird erneut die Parität berechnet und mit der im *parity* bit gespeicherten verglichen. Zeigen sich hierbei Unterschiede, so kann mit Sicherheit geschlossen werden, daß eines der neun bits fehlerhaft ist.

Mit dem eben Gesagten werden aber auch schon die Grenzen dieser Methode deutlich:

1. es ist nicht möglich, den Fehler zu korrigieren.

2. es könnte auch das parity bit fehlerhaft sein und so ein korrektes byte als fehlerhaft markieren.

3. treten zwei bit-Fehler gleichzeitig auf, so kann dies die parity-Prüfung nicht erkennen und signalisiert ein fehlerfreies byte.

Will man diese Einschränkungen vermeiden, so muß man die Redundanz erhöhen.

Pin-out: Damit bezeichnet man die geometrische Anordnung der Verbindungskontakte auf dem Gehäuse einer integrierten Schaltung sowie die logische Zuordnung der Kontakte zu Signalen.

Primitive: Damit bezeichnet man Sprachelemente, die nicht weiter unterteilt werden können. Aus ihnen sind andere, nicht primitive Konstrukte aufgebaut. Ein Beispiel: in der deutschen Sprache sind die Worte oder Wortteile die kleinsten sinntragenden Einheiten (Primitive), aus denen Sätze und zusammengesetzte Wörter aufgebaut werden.

Proteinstruktur: Bei Proteinen unterscheidet man *Primär-*, *Sekundär-*, *Tertiär-* und *Quartärstruktur*. Als *Primärstruktur* bezeichnet man die Aminosäuresequenz des Proteins. Die Aminosäuren bauen dabei keine gestreckte Makromolekülkette auf, sondern das Protein ist mehr oder weniger stark gefaltet: die häufigsten Strukturen sind die α-Helix und die β-Faltblattstruktur. Die Abfolge dieser Konformationen entlang des Proteinstranges wird als *Sekundärstruktur* bezeichnet. Die genaue räumliche Anordnung der Sekundärstruktur ergibt schließlich die *Tertiärstruktur*, welche die dreidimensionale Struktur des Proteins beschreibt. Schließen sich zwei oder mehrere Polypeptidketten zusammen, die auf die eben beschriebene Art verknäuelt sind, so nennt man die räumliche Anordnung dieses Proteins *Quartärstruktur* [39].

RAM: Bei *dynamischen Speichern* (DRAM) wird die Information als Ladungszustand von kleinen integrierten Kondensatoren (C $\approx$ 50 fF) gespeichert. Um ein bit zu speichern, benötigt man nur einen Kondensator und einen Transistor. Da sich die Kondensatoren mit der Zeit entladen und somit die gespeicherte Information vergessen, muß die Information zyklisch (typisch alle 4–32 ms) aufgefrischt werden („refresh“). Daher die Bezeichnung *dynamisch*. *Statische* Speicher benutzen pro bit ein flip flop aus *zwei* Transistoren als Speicher. Diese brauchen mehr Platz, als die Kombination Transistor-Kondensator. Daher erreicht man mit statischen Speichern keine so hohen Speicherdichten wie mit dynamischen. Genau dies ist der Grund, warum wir dynamische Speicher verwenden. Der Vorteil statischer Speicher liegt zum einen in ihrer kürzeren Zugriffszeit, zum anderen kommen sie *ohne* refresh aus (daher die Bezeichnung *statisch*). Der on-chip Speicher der Transputer ist als statischer Speicher aufgebaut und daher besonders schnell.

RAM steht für **R**andom **A**ccess **M**emory und bezeichnet einen Speicher mit wahlfreiem Schreib- und Lesezugriff, der früher als magnetischer Kernspeicher ausgeführt war, heute aber fast ausschließlich mit integrierten Schaltungen realisiert wird.

ROM: **R**ead **O**nly **M**emory: ein nichtflüchtiger Festwertspeicher, aus dem nur gelesen werden kann. Seine Informationen bleiben auch bei Abschalten der Versorgungsspannung erhalten. Daher werden in einem ROM diejenigen Programmteile abgelegt, die der Rechner ganz zu Beginn, nach Einschalten der Versorgungsspannung, ausführen soll. Im allgemeinen wird dies der Programmcode für einen boot ($\Rightarrow$G) sein.

Scheduler: In einem multitasking-System muß es eine Instanz geben, welche die CPU den einzelnen rechenwilligen Prozessen zuteilt; das ist der *scheduler*. Er kann in Software (UNIX-Betriebssystem) oder Hardware (Transputer) realisiert sein. Die Hardwarelösung ist dabei gegenüber der Softwarelösung immer wesentlich schneller.

Semaphore: Steht in einem multitasking-System ein Betriebsmittel (z.B. Drucker, Plattenspeicher, etc.) für mehrere Prozesse nur einmal zur Verfügung, so muß gewährleistet sein, daß dieses Betriebsmittel zu jedem Zeitpunkt höchstens von einem Prozeß benutzt wird. Zu diesem Zweck werden *Semaphore* eingerichtet.

In unserem Fall muß der Kermit-Prozeß mit der Datenübertragung warten, bis das zu übertragende file vollständig geschrieben ist. Auf ein file darf immer nur ein Prozeß (Kermit oder server) zugreifen; dies gewährleistet eine Semaphore. Hier sind die files selbst Semaphoren: sobald der server beginnt, das nächste file zu schreiben, ist sichergestellt, daß das vorhergehende abgeschlossen ist und vom Kermit-Programm übertragen werden kann.

SIP-Modul: Die Abkürzung „SIP" steht für **S**ingle **I**nline **P**ackage, was bedeutet, daß alle Anschlußkontakte in einer Linie angeordnet sind (siehe Abbildung 4.7). Weitverbreitet bei integrierten Schaltungen ist dagegen das DIP-Gehäuse: **D**ual **I**nline **P**ackage, bei dem *zwei* parallele Anschlußreihen vorhanden sind. *Modul* besagt, daß hier mehrere (9) Speicherbausteine zu einer Einheit zusammengefaßt sind. So läßt sich eine sehr hohe Packungsdichte erreichen.

Tag: Ein *tag* ist ein einzelnes byte, das den Datenblöcken, die über ein link übertragen werden, vorangestellt wird. Am Wert dieses tag kann das Programm feststellen, um welche Art von Daten es sich handelt (z.B. Ortskoordinaten, Ladungen, Energien). Ein tag kann auch für sich alleine stehen (*ready-tag*) und einen Zustand („bin fertig") signalisieren.

Topologie: Unter *Topologie* wird im Kontext dieser Arbeit die Verschaltung der einzelnen Rechenelemente zu einem Netzwerk verstanden. Mögliche Topologien sind Ring-, Würfel-, Torus-, Baumtopologie, ein zweidimensionales Gitter, mehrdimensionale Würfel (hypercube) und viele andere.

Transputer: Dieser Begriff ist in Abschnitt 4.3.1 erklärt.

Workspace: Wörtlich: Arbeitsbereich. Jeder Prozeß benötigt einen Speicherbereich, in dem er lokale Daten wie Zwischenergebnisse und pointer ablegen kann. Auf die Daten in diesem Bereich greift er besonders häufig zu. Daher sollte dafür ein schneller Speicher (wie es der on-chip-Speicher ist) verwendet werden.

Wort: Im Zusammenhang mit bit (⇒G) und byte (⇒G) bezeichnet es die Zusammenfassung von 4 Bytes oder 32 bits zu einer Einheit. Das *Wort* ist eng mit der Breite von Daten- und Adreßbus des Prozessors verknüpft.

Literaturverzeichnis

[1] „Getting Started“, INMOS, April 1987, TDS2.0, #72TDS10900.

[2] „IMS T414 Transputer“, INMOS, February 1987, preliminary Data, #42107801.

[3] „IMS T800 Architecture“, INMOS, March 1987, Technical note 6, #72TCH00601.

[4] „IMS T800 Transputer“, INMOS, April 1987, Preliminary Data, #42108200.

[5] „OCCAM Implementation“, INMOS, April 1987, TDS2.0, #72TDS11500.

[6] „OCCAM Standard Libraries“, INMOS, April 1987, TDS2.0, #72TDS11400.

[7] „OCCAM2 Reference Manual“, Prentice Hall, Cambridge University Press, Cambridge, 1988, #72OCC4501.

[8] „Programming Interface“, INMOS, April 1987, TDS2.0, #72TDS11200.

[9] „Qualität und Zuverlässigkeit—das sollte jeder wissen“, TI, November 1987, Ausgabe 18, S. 5.

[10] „Reference Manual: Transputer Architecture“, INMOS, July 1987, #72TRN04803.

[11] „Server Documentation“, INMOS, April 1987, TDS2.0, #72TDS11300.

[12] „Software Tools“, INMOS, April 1987, TDS2.0, #72TDS11100.

[13] „Transputer Development System“, INMOS, 1988, #72TRN01100.

[14] „Transputer Development System“, INMOS, November 1985, #72TRN03200, #72TDS02901, #72TDS04300, #72TDS04200, #72TDS02100, #72TDS03501, #72TDS02200, #72TDS03800, #72TDS02300, #720CC04000, #72TDS02400, #72TDS04100.

[15] „User Manual“, INMOS, April 1987, TDS2.0, #72TDS11000.

[16] P. Atkin: „Performance Maximisation“, INMOS, March 1987, Technical note 17, #72TCH01700.

[17] P. Atkin and J. Packer: „High performance graphics with the IMS T800“, INMOS, 1988, Technical note 37, #72TCH03700.

[18] P. Batacan: „AT&T and IBM multiprocessors spur molecular dynamics research“, *Computers in Physics*, 9–10 (1988).

[19] J. Breton, J. L. Martin, A. Migus, A. Antonetti, and A. Orszag: „Femtosecond spectroscopy of excitation energy transfer and initial charge separation in the reaction center of the photosynthetic bacterium *Rhodopseudomonas viridis*“, *Proc. Natl. Acad. Sci.*, **83**:5121 (1986), USA.

[20] A. T. Brint and P. Willett: „Identifying 3D maximal common substructures using transputer networks“, *Journal of Molecular Graphics*, (1987).

[21] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus: „CHARMM: a program for macromolecular energy, minimization, and dynamics calculations“, *Journal of Computational Chemistry*, **4**(2):187–217 (1983).

[22] F. R. Brown and N. H. Christ: „Parallel supercomputer for lattice gauge theory“, *Science*, **239**:1393–1400 (1988).

[23] A. T. Brünger: „X-PLOR“, The Howard Hughes Medical Institute and Department of Molecular Biophysics and Biochemistry, Yale University, 260 Whitney Avenue, P.O. Box 6666, New Haven, CT 06511, May 1988.

[24] C. R. Cantor and P. R. Schimmel: „Biophysical Chemistry“, Volume II, W. H. Freeman and Company, New York, 1980.

[25] J. Deisenhofer, O. Epp, K. Miki, R. Huber, and H. Michel: „Structure of the protein subunits in the photosynthetic reaction center of *Rhodopseudomonas viridis* at 3 Å resolution“, *Nature*, **318**:618 (1985).

[26] J. Deisenhofer and H. Michel: „The crystal structure of the photosynthetic reaction center from *Rhodopseudomonas viridis*“, in *The photosynthetic bacterial reaction center: Structure and dynamics*, J. Breton and A. Vermeglio, editors, Plenum Press, London, 1987.

[27] J. Deisenhofer and W. Steigemann: „Crystallographic refinement of the structure of bovine pancreatic trypsin inhibitor at 1.5 Å resolution“, *Acta Cryst.*, **B 31**:238 (1975).

[28] H. Ebert: „Starke Familienbande - die Transputerbausteine“, *c't*, (10):180–196 (1987).

[29] P. Eckelmann: „Transputer der 2. Generation“, *Elektronik*, (1, 2, 3) (1987).

[30] D. Fincham: „Parallel computers and molecular simulation“, *Molecular Simulation*, **1**(1, 2):1–45 (1987).

[31] K. Fleder: „Schaltkreise zur Erkennung und Korrektur von Fehlern in Speichersystemen, EDAC = Error Detection And Correction“, Applikationsbericht EB161, TI, December 1984.

[32] S. Gasiorowicz: „Quantum Physics“, John Wiley & Sons, New York, Chichester, Brisbane, Toronto, Singapore, 1974.

[33] S. Ghee: „IMS B004; IBM PC add-in board“, INMOS, February 1987, Technical note 11, #72TCH01100.

[34] W. K. Giloi: „Der Suprenum-Rechner“, *c't*, (7):92–100 (1988).

[35] T. Gore and D. Cormie: „Designing the IMS T414 and IMS T800 Memory Interface“, INMOS, April 1987, Technical note 9, #72TCH00901.

[36] H. Grubmüller: Diplomarbeit, Technische Universität München, James-Franck-Straße, 8046 Garching bei München, 1989, In Vorbereitung.

[37] H. Grubmüller, H. Heller, and K. Schulten: „Eine CRAY für 'jedermann'“, *mc*, (11):48–65 (1988).

[38] G. Harriman: „Notes on Graphics Support and Performance Improvements on the IMS T800“, INMOS, July 1987, Technical note 26, #72TCH02600.

[39] H. Hassinger and R. Wiebusch: „Experimentelle Enzymologie“, in *Biologie*, Diesterweg, Salle, Sauerländer, Frankfurt am Main, 1977.

[40] G. Häusler: „Die Assemblersprache des Transputers“, *mc*, (7):69–73 (1988), Teil 2.

[41] G. Häusler: „Die mc Transputerkarte“, *mc*, (4):38–44 (1988).

[42] W. D. Hillis: „The Connection Machine“, MIT Press, 1985.

[43] W. D. Hillis and Barnes: „Programming a highly parallel computer“, *Nature*, **326**:27–30 (1987).

[44] C. A. R. Hoare: „Communicating sequential processes“, *Communications of the ACM*, **21**(8):666 (1978).

[45] M. Karplus and J. A. McCammon: „The dynamics of proteins“, *Scientific American*, **4**:30 (1986).

[46] M. Karplus and J. A. McCammon: „Dynamics of proteins: elements and function“, *Ann. Rev. Biochem.*, **53**:263 (1983).

[47] I. Klapper, R. Hagstrom, R. Fine, K. Sharp, and B. Honig: „Focusing of electric fields in the active site of Cu-Zn superoxide dismutase: effects of ionic strength and amino-acid modification“, *Proteins: Structure, Function, and Genetics*, **1**:47–59 (1986).

[48] P. Mattos: „The transputer based Navigation System“, INMOS, February 1987, Technical note 2, #72TCH00200.

[49] D. May: „OCCAM2, Language Definition“, INMOS, February 1987, #72OCC04401, #72OCC04402.

[50] H. Michel and J. Deisenhofer: „X-Ray diffraction studies on a crystalline bacterial photosynthetic reaction center: a progress report and conclusions on the structure of photosystem II reaction center“, Volume 19 of *Encyclopedia of plant physiology: photosynthesis III*, A. C. Staehelin and C. J. Arntzen, editors, Springer, Berlin, 1986.

[51] C. Niedermeier: „Elektrostatische Kontrolle der Primärprozesse im photosynthetischen Reaktionszentrum von Rhodopseudomonas viridis“, Diplomarbeit, Technische Universität München, James-Franck-Straße, 8046 Garching, 1989.

[52] N. S. Ostlund and R. A. Whiteside: „A machine architecture for molecular dynamics: the systolic loop“, in B. Venkataraghavan and R. J. Feldmann, editors, „Macromolecular Structure and Specificity: Computer-assisted Modeling and Applications“, pages 195–208, The New York Academy of Sciences, New York, New York, 1985, Annals of the New York Academy of Sciences.

[53] J. Packer: „Exploiting Concurrency; A Ray Tracing Example“, INMOS, February 1987, Technical note 7, #72TCH00700.

[54] J. Packer: „Simpler real-time programming with the transputer“, INMOS, May 1988, Technical note 51, #72TCH05100.

[55] D. Pountain: „A transputer based Radio-Navigation System“, INMOS, February 1987, Technical note 0, #72TCH00000.

[56] D. Pountain: „A Tutorial Introduction to OCCAM Programming“, INMOS, April 1987, #72TDS11500.

[57] F. Reif: „Statistische Physik und Theorie der Wärme", W. Muschik, editor, Walter de Gruyter, Berlin, New York, 2 edition, 1985, ISBN 3-11-010264-1.

[58] H. Richter: „Multiprozessor mit dynamisch variabler Topologie", IPP-Bericht, Max-Planck-Institut für Plasmaphysik, 8046 Garching, June 1988, IPP R/42.

[59] M. Rygol and T. Watson: „Connecting INMOS links", INMOS, April 1987, Technical note 18, #72TCH01800.

[60] R. Shephard: „Extraordinary use of transputer links", INMOS, April 1987, Technical note 1, #72TCH00101.

[61] R. Shepherd and P. Thompson: „Lies, Damned Lies and Benchmarks", INMOS, February 1987, Technical note 27, #72TCH02700.

[62] R. Steinmetz: „OCCAM2", Hüthig-Verlag, Heidelberg, 1987.

[63] Stoer and Bulirsch: „Numerische Mathematik", Volume II of *Heidelberger Taschenbücher*, Springer, Berlin, Heidelberg, New York, 2 edition, 1978.

[64] H. Treutlein: „Molekulardynamiksimulation des Reaktionszentrums von Rhodopseudomonas viridis", Doktorarbeit, Technische Universität München, Physik-Department, T 30, James-Franck-Straße, 8046 Garching, 1988, In Vorbereitung.

[65] H. Treutlein, K. Schulten, J. Deisenhofer, H. Michel, A. Brünger, and M. Karplus: „Molecular dynamics simulation of the primary processes of the photosynthetic reaction center of rps. viridis", in J. Breton and A. Vermeglio, editors, „The photosynthetic bacterial reaction center: Structure and dynamics", pages 139–150, Plenum Press, London, 1987.

[66] H. Treutlein, K. Schulten, C. Niedermeier, J. Deisenhofer, H. Michel, and D. De-Vault: „Electrostatic control of electron transfer in the photosynthetic reaction center of Rps. viridis", in *The photosynthetic bacterial reaction center: Structure and dynamics*, J. Breton and A. Vermeglio, editors, Plenum Press, London, 1987.

[67] L. Verlet: „Computer 'experiments' on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules", *Physical Review*, **159**(1):98–103 (1967).

[68] A. Warshel and S. T. Russel: „Calculations of electrostatic interactions in biological systems and in solution", *Q. Rev. Biophys.*, **1** (1984).

[69] A. Windemuth: „Dynamiksimulation von Makromolekülen", Diplomarbeit, Technische Universität München, Physik-Department, T 30, James-Franck-Straße, 8046 Garching, August 1988.

[70] A. Windemuth: „MD“, August 1988, Molekulardynamikprogramm, C-Quellcode, private Mitteilung.

[71] W. Wöst: „Wie funktioniert der TX3?“, *c't*, (6):134–146 (1988).