FORSCHUNGSZENTRUM JÜLICH GmbH
Jülich Supercomputing Centre
D-52425 Jülich, Tel. (02461) 61-6402

Interner Bericht

Implementierung des Proteinkraftfeldes ECEPP/3 auf der Cell Broadband Engine

Lauretta Schubert

FZJ-JSC-IB-2009-03

August 2009

(letzte Änderung: 12.08.2009)

Hochschule Aachen

Campus Jülich

Fachbereich Medizintechnik und Technomathematik

Bachelor-Arbeit im Studiengang Scientific Programming

Implementierung des Proteinkraftfeldes ECEPP/3 auf der Cell Broadband Engine

Lauretta Schubert

Jülich Supercomputing Centre (JSC)

JÜLICH
FORSCHUNGSZENTRUM

Jülich, den 13. August 2009

Die vorliegende Bachelorarbeit wurde in Zusammenarbeit mit der Forschungszentrum Jülich GmbH, am Jülich Supercomputing Centre (JSC) angefertigt.

Sie wurde betreut von:
Referent: Prof. Dr. rer. nat. Christof Schelthoff
Koreferent: Dr. Jan H. Meinke

Diese Arbeit ist von mir selbständig angefertigt und verfasst. Es sind keine anderen als die angegebenen Quellen und Hilfsmittel benutzt worden.

Lauretta Schubert

Inhaltsverzeichnis

Abkürzungsverzeichnis

ALU Arithmetic Logical Unit

CPU Central Processing Unit

DMA Direct Memory Access

DNA Desoxyribonukleinsäure

ECEPP Empirical Conformational Energy for Peptides and Proteins

EIB Element Interconnect Bus

HS Hauptspeicher

LS Local Store

MFC Memory Flow Controller

MIC Memory Inferface Controller

MPI Message Passing Interface

mRNA Messenger-Ribonukleinsäure

NMR Magnetresonanzanalyse

PPE PowerPC Processing Element

RISC Reduced Instruction Set Computing

SDK Software Development Kit

SIMD Single Instruction Multiple Data

SMMP Simple Molecular Mechanics for Proteins

SMP Symmetrisches Multiprozessorsystem

SMT Simultaneous Multithreading

SPE Synergistic Processing Element

tRNA Transfer-Ribonukleinsäure

1 Motivation

Proteine gehören zu den wichtigsten Bausteinen des Körpers. Sie verleihen den Zellen ihre Struktur und übernehmen wichtige Funktionen wie z. B. den Sauerstofftransport oder die Biokatalysation. Um diese Funktionen ausüben zu können, muss jedes Protein seine ganz spezielle drei-dimensionale Form eingenommen haben. Diese erhält es wenige (Milli-)Sekunden nach seiner Synthese, indem es sich faltet. Eine falsche Faltung kann bedeuten, dass das Protein seine Aufgabe im Körper überhaupt nicht oder nicht richtig ausüben kann. Die Folge können Krankheiten wie Alzheimer oder BSE sein.
Deshalb ist es von höchster Bedeutung den Faltungsvorgang zu verstehen. Doch die Faltung ist nicht einfach zu beobachten, denn dabei müsste die Bewegung einzelner Atome sichtbar gemacht werden - und das auch noch innerhalb sehr kurzer Zeitspannen. Deshalb verfolgt die Faltungssimulation *in silicio* verschiedene Methoden, die auf energetischen Berechnungen aufbauen. Das Proteinkraftfeld ECEPP/3 setzt dabei auf Coulomb-, Van-der-Waals-, Wasserstoffbrückenbindung- und Torsionswechselwirkungen. Mittels empirisch bestimmter Parameter berechnet es die Energie einer Konformation. Auf dieser Basis kann mit Hilfe von Monte Carlo Algorithmen der Konformationsraum eines Moleküls untersucht werden, um mögliche Faltungswege aufzuzeigen.
Bei der Berechnung dieser Wechselwirkungen tritt die Berechnung des Abstandes zweier wechselwirkender Atome in den Vordergrund. Der Rechenaufwand wächst für eine einzelne Energieberechnung quadratisch zur Problemgröße an, da jedes Atom mit jedem Atom wechselwirkt. Außerdem sind für eine gute Abdeckung des Konformationsraumes viele Berechnungen nötig, sodass sich ein enormer Rechenbedarf ergibt.
Heutige Supercomputer bewältigen solche Aufgaben durch Parallelisierung über Mehrkernprozessoren. Die Cell Broadband Engine stellt in diesem Feld allein als Multicore-Prozessor, der aus zwei wesentlichen Komponenten besteht. Zum einem dient ein PowerPC als Steuereinheit und acht einfache Co-Prozessoren als treibende Recheneinheiten. Die Co-Prozessoren verfügen über eine SIMD-Recheneinheit, mit der vier Floating-Point-Operationen parallel berechnet werden können. Eine effiziente Ausnutzung verspricht eine gute Performance-Steigerung in der Faltungssimulation für Proteine.
Diese Arbeit wird kurz in die biologischen und physikalischen Hintergründe einführen. Sie wird die Hardware vorstellen und wichtige Details zur Programmierung erklären. Darauf folgt die Beschreibung der Implementierung eines Moduls, dass die Energieberechnung des Proteinkraftfeldes ECEPP/3 auf den Co-Prozessoren der Cell Broadband Engine durchführt. Dabei werden nacheinander drei Ansätze untersucht.
Zum Schluß wird das Laufzeitverhalten mit Implementierungen auf anderen Architekturen verglichen und damit die Leistung der Cell Broadband Engine eingestuft.

2 Die Proteinfaltung

Proteine gehören zu den strukturgebenden Grundbausteinen der Zellen und erfüllen als Enzyme, Ionenpumpen, Hormone und in vielen anderen Einzelausprägungen wichtige Funktionen im Körper. Ohne sie wäre er nicht lebensfähig. Daher ist der Name Protein wahrscheinlich auch von dem griechischem Wort protos (das Erste/Wichtigste) abgeleitet.
Die wohl häufigste Proteinstruktur beim Menschen ist die des Kollagens. Gut $\frac{1}{3}$ des Körpers besteht aus Kollagenen, die als Strukturproteine für Haut und Bindegewebe dessen Beschaffenheit maßgeblich bestimmen. Haare bzw. Fell, Nägel/Klauen, Schuppen, Federn, Schnäbel oder Hörner werden aus dem Faserprotein Keratin aufgebaut.
Im Körper selbst übernehmen die Proteine Funktionen als Enzyme, Transportproteine, Rezeptoren, Hormone, Antikörper, etc.
Als Enzyme oder auch Biokatalysatoren ermöglichen und beschleunigen sie Reaktionen unter Körperbedingungen, die normalerweise z. B. erst unter höheren Temperaturen oder wesentlich langsamer verlaufen würden. Die Reaktionspartner (Substrate) werden dazu im sogenannten aktiven Zentrum des Enzyms gebunden. Das aktive Zentrum ist bei den meisten Enzymen substratspezifisch und ein Protein katalysiert meist eine spezielle Reaktion. Für die Reaktion setzt das Enzym die Aktivierungsenergie der Reaktion herunter. Das bedeutet, dass es Energie bereit stellt um die Reaktion einzuleiten.
Ionenkanäle oder -pumpen regulieren passiv oder aktiv (unter Energieverbrauch) die Ionenkonzentration innerhalb einer Zelle. Durch sie können sich Ionen durch Zellwände und in Zellorganellen bewegen. Auch Proteine selbst wandern durch entsprechende Kanäle, um in Zellorganellen zu gelangen. Für diesen Transport müssen sie sich allerdings wieder vollständig ent- und danach zurückfalten.
Als Rezeptoren dienen Proteine als eine Art Nachrichtenempfänger der Zelle. Sie besitzen spezielle Bindungsstellen an denen sich Botenstoffe anlagern, wodurch ein Signal an die Zelle gesendet wird. So besitzen z. B. manche Ionenkanäle Rezeptoren. Auf ein Signal hin öffnet sie ihre Pore. Ein anderes Beispiel sind Rezeptoren an Enzymen. Durch das Andocken eines spezifischen Botenstoffes verändert sich das aktive Zentrum des Enzyms. Dadurch wird die Enzymtätigkeit ausgeschaltet, weil das Substrat nun nicht mehr in das aktive Zentrum passt. So wird z. B. die Enzymaktivität reguliert.
Zudem gibt es Proteine, die mit für die Muskelkontraktion verantwortlich sind oder als Reserven zur Energiegewinnung in lebenswichtigen Organen vorkommen.
Doch woraus sind Proteine aufgebaut, wie werden sie im Körper hergestellt und was macht sie zu den speziellen Funktionsträgern? Dieses Kapitel beschäftigt sich basierend auf [1] und [2] mit dem Aufbau, der Synthese und Faltung der Proteine.

2.1 Proteinaufbau

Ein Protein - auch Eiweiß genannt - ist ein aus Aminosäuren aufgebautes Makromolekül. Im menschlichem Körper kommen Hunderttausende verschiedener Proteine vor, wobei die Struktur der Proteine wesentlich für ihre Funktion ist. Sie bestehen aus einer Kette von Aminosäuren, was man dann (Poly-)Peptid nennt. Polypeptide, die eine eindeutige räumliche Struktur aufweisen, bezeichnet man als Protein. Sie unterscheiden sich an ihrer unterschiedlichen Sequenz aus Aminosäuren. Diese stellt die erste von vier Strukturebenen des Proteinaufbaus dar. Nachfolgend werden die vier Strukturebenen erklärt. Doch zunächst folgen Details zu den Aminosäuren.

2.1.1 Aminosäuren

Die Aminosäuren, aus denen Proteine aufgebaut sind, bestehen selbst wiederum aus einzelnen Atomen. Dabei besitzen sie mindestens eine Carboxylgruppe ($-COOH$) und eine Aminogruppe ($-NH_2$). Sie unterscheiden sich lediglich in ihren Seitenketten - auch Reste genannt. Verbunden werden sie durch eine Peptidbindung zwischen der Aminogruppe eines Bausteins und der Carboxylgruppe eines Anderen [3]. An der Proteinsynthese sind die sogenannten proteinogenen Aminosäuren beteiligt. Diese gehören zu den α-Aminosäuren, deren Aminogruppe am zweiten Kohlenstoffatom gebunden ist.
Das einfachste Beispiel ist Glycin:

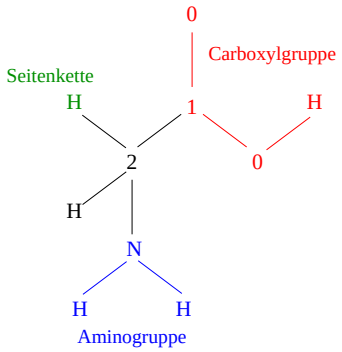

Abbildung 2.1: Glycin - die kleinste α-Aminosäure. Wie jede der Aminosäuren besteht sie aus einer Carboxylgruppe (rot) und einer Aminogruppe (blau), die zwei Kohlenstoffatome voneinander entfernt sind. Am zweiten Kohlenstoffatom, dem C-α, hängt die Seitenkette (grün), die bei Glycin lediglich ein Wasserstoffatom ist.

Insgesamt sind derzeit 22 proteinogene Aminosäuren bekannt. Davon sind 20 über sogenannte Codons oder auch Basentripletts in der Erbsubstanz codiert. Die Abfolge von drei Basen in der Erbsubstanz definiert genau eine Aminosäure. Man nennt diese 20 Aminosäuren deshalb kanonisch.
Die verschiedenen Aminosäuren haben vielfältige Eigenschaften (z. B. sauer, basisch, groß, klein, flexibel, starr, hydrophob, hydrophil, etc.), die sich unterschiedlich auf die Faltung auswirken. So findet man die hydrophoben (wasserhassenden) Aminosäuren vorwiegend im Inneren eines Proteins.

2.1.2 Strukturebenen

Der Aufbau von Proteinen wird hierarchisch in vier Strukturebenen aufgeteilt:
Die **Primärstruktur** beschreibt - wie oben erwähnt - die Aminosäuresequenz innerhalb der Kette. Das heißt, sie beschreibt die Folge der Aminosäuren, die über Peptidbindungen verbunden sind. Diese kovalenten Bindungen „ermöglichen die Rotation der sie verbindenden Atome [...]. Ein Makromolekül könnte deshalb prinzipiell eine nahezu unbegrenzte Anzahl an Formen, oder Konformation, einnehmen [...]“[1]. Die Primärstruktur gibt also keinerlei Aufschluß über die räumliche Anordnung des Proteins.

His - Ser - Gln - Gly - Thr - Phe - Thr - ... - Met - Asn - Thr

Abbildung 2.2: Primärstruktur: Aminosäuresequenz von Thryptophan. Die Primärstruktur eines Proteins ist die Reihe von Aminosäuren, aus denen es aufgebaut ist. Die Namen der Aminosäuren werden hier in der gängigen Dreibuchstabennotation geschrieben. Dabei ist die Leserichtung von der Carboxyl- zur Aminogruppe.

Als **Sekundärstruktur** bezeichnet man häufig auftretende Motive für den lokalen räumlichen Aufbau einer bestimmten Aminosäuresequenz. Dazu gehören die α-Helices, β-Faltblätter und β-Schleifen. Sie stellen einzelne Untereinheiten des Proteins dar.

Abbildung 2.3: Sekundärstrukturarten: α-Helix, β-Faltblatt und β-Schleife. Diese schematische Darstellung zeigt die α-Helix und das β-Faltblatt als Bänder, die β-Schleife im Stäbchenmodell. Quelle: [4]

Die **α-Helix** ist ein oft auftretendes Sekundärstrukturmotiv. Sie stellt eine rechtsgängige Spirale dar, deren Ganghöhe 0,54 nm beträgt. Pro Umdrehung sind das ungefähr 3,6 Seitenketten. Damit hat sie eine äußerst kompakte Struktur, bei der die Aminosäuren nur 0,15 nm entfernt sind. Stabilisiert wird sie durch Wasserstoffbrückenbindungen. Dabei gehen immer die Kohlenwasserstoffgruppe an Position n mit der Gruppe aus Stickstoff und Wasserstoff an Position n+4 eine solche Bindung ein.
Das **β-Faltblatt** verdankt seinen Namen seiner Zickzack-Form, das einem gefalteten Blatt ähnelt. Es besteht aus mindestens zwei sogenannten β-Strängen, die entweder parallel oder antiparallel verknüpft sind. Dies beschreibt die Anordnung der beiden Stränge zueinander. Diese Verknüpfung wird durch Wasserstoffbrückenbindungen gebildet. Allerdings beträgt hierbei der Ab-

stand benachbarter Aminosäuren 0,35 nm.
Die Verbindung zwischen α-Helices und β-Faltblättern stellen **β-Schleifen** dar. Sie führen zum Abknicken der Polypeptidketten und treten bevorzugt an den Oberflächen von Proteinen auf.

Durch die nicht-kovalenten Bindungen (z. B. Van-der-Waals-Kräfte, Wasserstoffbrückenbindungen) zwischen nicht benachtbarten Atomen faltet sich das Protein und bildet damit die **Tertiärstruktur** aus. Diese Wechselwirkungen sind äußerst schwach. Jedoch können eine Vielzahl dieser schwachen Bindungen die Bewegungen der Aminosäurenkette einschränken, sodass diese einen bevorzugten gefalteten Zustand einnimmt.

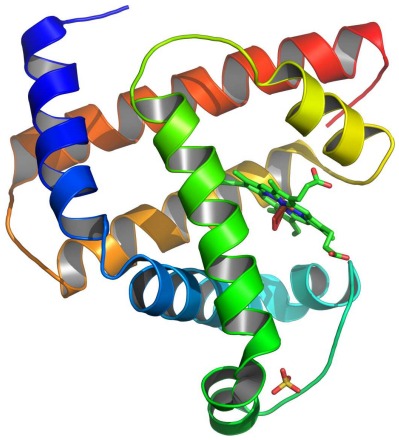

Abbildung 2.4: Tertiärstruktur des Protein Myoglobin. Die Sekundärstruktur von Myoglobin weißt acht α-Helices auf. Die räumliche Anordnung dieser definiert die Tertiärstruktur. Myoglobin war das erste Protein, dessen Tertiärstruktur mittels Kristallstrukturanalyse ermittelt werden konnte. Quelle [5]

Oft bilden mehrere Proteine einen gemeinsamen Komplex. Diese Formation zu einem Makromolekül ist die **Quartärstruktur**, deren einzelnen Teile man auch Monomere nennt. Manche Monomere besitzen auch als eigenständiges Protein ihre Funktion, andere erlangen sie erst in einem solchen Zusammenschluß.

2.2 Proteinbiosynthese

Unter der Proteinbiosynthese versteht man die Herstellung eines Proteins in der Zelle. Der Bauplan für ein Protein ist die Desoxyribonukleinsäure (DNA). Sie wird in einem ersten Schritt der Proteinbiosynthese, der Transkription, gelesen und in Messenger-Ribonukleinsäure (mRNA) umgesetzt. In einem zweiten Schritt, der Translation, wird dann mit Hilfe der Informationen der mRNA das Protein gebaut. Beides soll nachfolgend genauer beschrieben werden.

2.2.1 Transkription

Die Transkription wird mit Bindung des Enzyms RNA-Polymerase an eine DNA-Sequenz eingeleitet. Sie trennt die DNA-Doppelhelix in einem Bereich einiger Basenpaare zu zwei DNA-Einzelsträngen auf. Damit verläuft die Transkription ähnlich wie die DNA-Replikation. An einem der beiden Stränge - dem sogenannte codogenen Strang - lagern sich an den Nukleinbasen der DNA (Adenin, Thymin, Guanin und Cytosin) die komplementären Nukleinbasen der RNA (anstelle von Thymin Uracil, ansonsten identisch zur DNA) an. Die Transkription endet an einer bestimmten Basensequenz, dem Terminator. Die prä-mRNA sowie die Polymerase lösen sich von der DNA.
Für Prokaryoten - Lebewesen ohne Zellkern - wäre die mRNA reif und könnte für die Translation zu den Ribosomen geleitet werden. Bei Eukaryoten - entsprechend Lebewesen mit Zellkern - muss die prä-mRNA noch das Capping und Splicing durchlaufen. Beim Capping wird noch ein Abschluß an die mRNA-Sequenz gehängt, um einen stabileren Zustand für den Transport aus dem Zellkern ins Cytoplasma zu den Ribosomen zu erreichen. Das Splicing entfernt nicht-codierende Abschnitte (Introns) und fügt die mRNA somit neu zusammen. Eine Besonderheit stellt dabei das alternative Splicing dar. Dabei wird erst beim Vorgang selbst von Splicefaktoren entschieden, welche Abschnitte Introns sind. Damit entstehen unterschiedliche mRNAs und später somit auch unterschiedliche Proteine aus einem Gen.

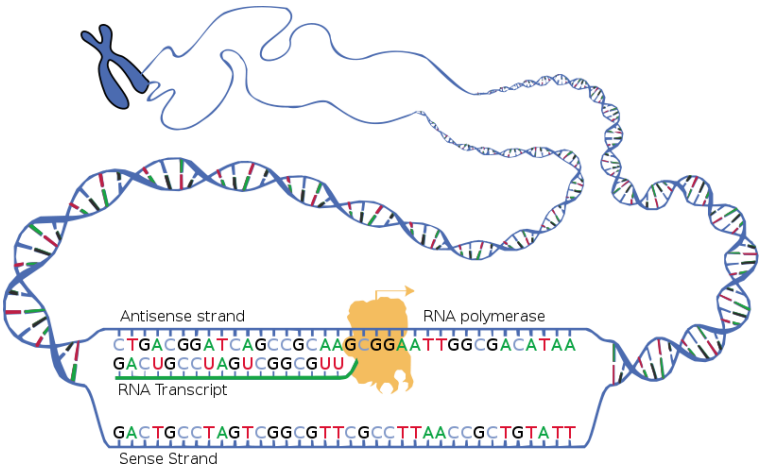

Abbildung 2.5: Modellvorstellung der Transkription. Diese Abbildung zeigt, wie die Aminosäurekette in einem kleinen Bereich durch die RNA-Polymerase aufgetrennt wurde und sich dort am codogenen Strang die komplementären Nukleinbasen der RNA an die der DNA lagern. Dabei bilden Adenin und Thymin sowie Guanin und Cytosin immer ein Paar, wobei Thymin in der RNA durch Uracil zu ersetzen ist. Quelle: [6]

2.2.2 Translation

Nach der Transkription gelangt die reife mRNA im Cytoplasma zu einem Ribosom. Ein Ribosom ist ein Komplex aus Proteinen und RNA. Im Ribosom läuft der zweite Teil der Proteinbiosynthese, die Translation, ab.

Eingeleitet wird dieser Prozess durch einen sogenannten Initiationskomplex, der an das Ribosom gebunden wird. Danach bindet die mRNA an das Ribosom und wird im verarbeitenden Teil des Ribosoms bis zum Start-Basentriplett `AUG` geschoben.
Jeweils drei aufeinanderfolgende Nukleotide der mRNA - ein Basentriplett oder auch Codon genannt - definieren eine bestimmte Aminosäure des Proteins. Das Ribosom katalysiert den Prozess die Aminosäuren miteinander zu verknüpfen. Dazu bringt es die Codons der mRNA mit entsprechenden Anticodons von tRNA zusammen. Die tRNA dient dabei als Adapter für Aminosäuren. An einer Seite bindet die tRNA eine Aminosäure. An der anderen Seite besitzt sie eine Bindungsstelle, die genau zu einer Sequenz auf der mRNA passt. Innerhalb eines Ribosoms werden Codon und Anticodon von mRNA und tRNA zusammengebracht, so dass die zwei Aminosäuren eine Peptidbindung miteinander eingehen können. Danach wandert das Ribosom ein Triplett weiter. Auf der einen Seite entlässt es eine tRNA ohne gebundene Aminosäure. Auf der Anderen bindet es eine neue tRNA mit Aminosäure. Dabei wächst die Kette der Aminosäuren immer weiter bis das Ribosom bei einem Stopp-Codon auf der mRNA angelangt ist. Daran kann keine tRNA binden und der Prozess der Synthese endet damit, dass sich die Aminosäurenkette vom Ribosom löst.

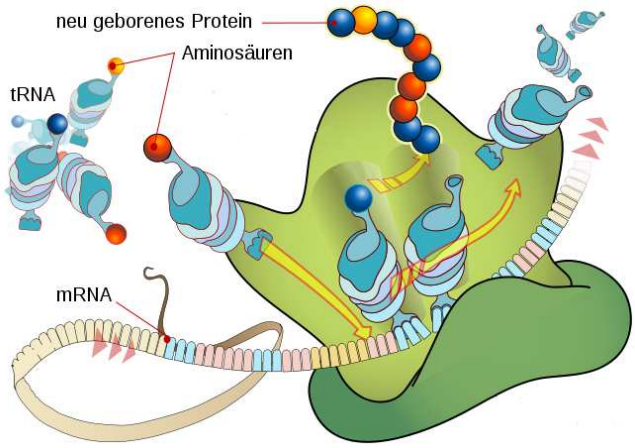

Abbildung 2.6: Modellvorstellung der Translation. Diese Abbildung illustriert, die Translation. Innerhalb des Ribosoms docken tRNAs an Basentripletts der mRNA an. Dadurch werden die daran gebundenen Aminosäuren so nah zusammengebracht, dass sie eine Peptidbindung miteinander eingehen. Aminosäure für Aminosäure wird an die Kette geknüpft und bilden ein neues Protein. Quelle: [7]

2.3 Proteinfaltung

Durch Proteinfaltung erhält eine Aminosäuresequenz eine feste Struktur und damit eine eindeutig definierte Funktion. Eine falsche Faltung hat zur Konsequenz, dass das Protein seine Aufgabe nicht (richtig) erfüllen kann oder gar andere (krankhafte) Funktionen ausübt. Deshalb ist die Erforschung dieses Prozesses so wichtig.
Die Einnahme des gefalteten Zustandes geschieht dabei nicht zufällig, wie ein Gedankenexperiment - das Levinthal-Paradoxon - zeigt.

2.3.1 Das Levinthal-Paradoxon

Das Levinthal-Paradoxon stellt ein kombinatorisches Problem dar. Geht man davon aus, dass jeder Aminosäurerest nur zwei verschiedene Zustände annehmen kann, so gäbe es bei einer Aminosäurekette mit n Gliedern 2^n mögliche Konformationen, die das Protein einnehmen kann. Nimmt man weiter an, dass die Änderung von einer Konformation in eine Andere 10^{-13} Sekunden benötigt, dann würde ein Protein mit 150 Aminosäuren $2^{150} * 10^{-13} s = 1,4 * 10^{32}$ Sekunden oder $4,6 * 10^{24}$ Jahre benötigen (im Vergleich: das Alter der Erde: $4,55 * 10^9$ Jahre). Einfach ausgedrückt: „selbst ein kleines Polypeptid würde eine nahezu unendliche Zeit brauchen, um alle möglichen Konformationen zu testen, um die richtige Struktur (mit der kleinsten Energie) zu finden“[8]. Da sich ein Protein in wenigen Millisekunden bis Sekunden in seinen nativen Zustand faltet, kann dieser nicht über ein Durchprobieren gefunden werden. Vielmehr vermutete schon Levinthal, dass ein jedes Protein einem Faltungsweg in einen energetisch günstigeren Zustand folgt.

2.3.2 Der „neue Weg“ und alte Modellvorstellungen

Diese Vorstellung wurde weiter zum sogenannten „neuen Weg“ entwickelt[9]. Er verbindet die zwei vorher widersprechenden Vorstellungen nach denen die treibende Kraft die Kinetik oder Thermodynamik ist. Nach Levinthal folgt ein Protein bei seiner Faltung einem eindeutig definierten Weg und findet so sehr schnell seinen Endzustand. Einer Idee von Anfinsen nach spielt der ungefaltete Ausgangszustand keine Rolle für den Endzustand. Dem „neuen Weg“ nach hat man sich einen n-dimensionalen Trichter vorzustellen, der die Energielandschaft verschiedener Konformationen beschreibt. Diese Landschaft ist nicht glatt sondern zerklüftet und hält damit einige Fallen bereit. Ein Protein gelangt (normalerweise) aus verschiedenen ungefalteten Zuständen zum gleichen endgültig Gefalteten. Doch gibt es auf dem Abstieg im Trichter viele Wege. Ein Weg der stetig bergab führt, ein anderer, bei dem Teilstücke ohne Energieänderung oder mit erneuten Anhebungen bewältigen werden müssen. Die Trichterbreite beschreibt dabei die „Bewegungsfreiheit“ (Entropie), die mit immer tieferem Abstieg immer geringer wird. Auf dem Boden des Trichters hat ein Protein seinen stabilen Zustand gefunden hat.

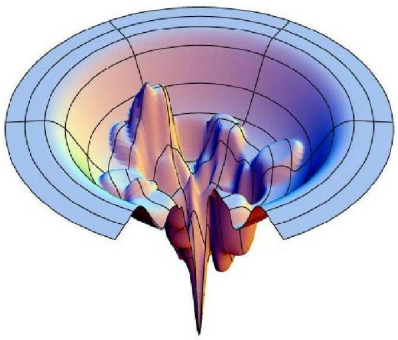

Abbildung 2.7: Illustration des zerklüfteten Faltungstrichters. Dem „neuen Weg“ nach, muss das Protein einen komplexen Abstieg auf seinem Faltungswegs in einem n-dimensionalen Raum zurückzulegen. Es muss tiefe lokale Minima und lange ebene Plateaus überwinden bis es seinen stabilen, gefalteten Zustand gefunden hat. Diese Abbildung deutet dies im dreidimensionalen Raum an. Quelle: [10]

Ein Faltungsweg, der stetig bergab führt, verläuft sehr schnell. Dabei gibt es für das Protein nur zwei (beobachtbare) Zustände - den ungefalteten und gefalteten Zustand. Da man am Anfang der Erforschung der Proteinfaltung nichts anderes beobachten konnte herrschte diese Vorstellung des **Zwei-Zustandsmodells** lange vor.
Bei einem Weg mit längeren Ebenen auf einem Energielevel werden Zwischenzustände (Intermediate) eingenommen. Dies wurde erstmal 1992 von Kim und Baldwin als das **sequenzielle Modell** beschrieben. Diese Modellvorstellung wurde zum sogenannten **Nucleationsmodell** weiterentwickelt. Es beschreibt, dass die Intermediate unterschiedlich schnell eingenommen werden. Dies lässt sich aber ebenso wie die vorherigen Modelle mit dem Trichtermodell erklären. Je nach der Steigung innerhalb der Energielandschaft wird der nächste Zustand entsprechend schnell oder langsam beobachtet. Intermediate, die lange erhalten bleiben, weil der Wechsel zum nächsten Zustand lang dauert nennt man dabei kinetischen Nucleus (dt. Kern).

2.3.3 Denaturierung

Die Umkehrung der Proteinfaltung ist die Denaturierung - die Auflösung der Sekundär- und Tertiärstrukturen. Hervorgerufen wird dieser Vorgang z. B. durch große Hitze oder Kälte, durch Druckänderung oder chemische Einwirkung von Säuren, Basen und organischen Lösungsmitteln. Es gibt sowohl reversible als auch irreversible Denaturierung. In den meisten Fällen ist der Vorgang der Denaturierung nicht von selbst umkehrbar. Das bedeutet: ohne äußere energetische Einflüsse kann der Ausgangszustand nicht mehr erreicht werden und das Protein ist somit unbrauchbar.

3 Simulation der Proteinfaltung mit ECEPP/3

In Kapitel 2 hat man einen Eindruck davon erhalten wie komplex aber zugleich auch schnell die Faltung ist. Es gibt verschiedene Modellvorstellungen, die versuchen den Faltungsprozess zu erklären, doch bleibt er noch weitgehend ungelöst.
Es gehört zur Grundlagenforschung diese Verständnislücke zu schließen. Sie stellt unter anderem die Basis für medizinische und pharmazeutische Forschungen dar, die es sich z. B. zum Ziel gesetzt haben durch Fehlfaltung hervorgerufene Krankkeiten zu verhindern.

Experimentell ist es nicht möglich den Faltungsweg zu beobachten. Es ist lediglich möglich den Ausgangs- und Endzustand sowie evtl. stabile Intermediate zu beobachten. Die Simulation am Rechner - in silicio - soll Aufschlüsse über den Ablauf der Proteinfaltung geben. Es soll z. B. die Frage nach der Häufigkeit oder der Struktur von Intermediaten beantwortet werden.
Die Simulationen basieren auf verschieden Modellen, die mit unterschiedlichen Parametrisierungen arbeiten. Die Ergebnisse liefern Häufigkeitsverteilungen von bestimmten Strukturen, die wiederum experimentell bestätigt werden können. Experimentelle Strukturbestimmung und die Simulation in silicio ergänzen also einander.

3.1 Das Modell

Für die Modellierung von Proteinen gibt es verschiedene Ansätze aus den Bereichen der Quanten-Mechanik und Molekular-Mechanik. Ihnen gemeinsam ist das Ziel: sie versuchen diejenige Konformation zu finden, die die kleinste Energiebilanz aufweist. Denn „ein Protein faltet im Allgemeinen in eine Form, in der die Freie Energie (G) ein Minimum aufweist“[1]. Um diese Konformation zu finden durchläuft das Protein nach der Modellvorstellung des Faltungstrichters (s. 2.3.2) als auch in der Simulation zahlreiche andere Konformationen, die sich als mehr oder minder stabil erweisen.
Bei quantenmechanischen Ansätzen werden die Elektronen eines Systems behandelt. Selbst wenn davon einige ignoriert werden, ist die Anzahl so groß, dass die Berechnung sehr zeitintensiv ist. Kraftfeld-Methoden ignorieren dagegen die elektronischen Bewegungen und berechnen die Energie anhand der Atom-Positionen (frei übersetzt nach [11]).
ECEPP/3 verfolgt das klassische Modell. Es setzt auf dem elektronischen Charakter der Atome auf, der intramolekulare Wechselwirkungen zwischen den Atomen hervorruft. Diese werden durch die verschiedenen kovalenten und nicht-kovalenten Bindungen bestimmt.

3.1.1 Kovalente Bindungen

Kovalente Bindungen oder auch Atombindungen sind chemische Bindungen, die für den festen Zusammenhalt der einzelnen Atome in einem Molekül sorgen. Ein Atom kann je nach Typ ein bis vier kovalente Bindungen ausbilden. Um dies zu verstehen folgt ein kleiner Exkurs zum (bohrschen) Atommodell[1]:
Im Zentrum findet man den Kern, der den Großteil der Atommasse bestimmt. Er enthält eine Anzahl an Protonen und Neutronen, entsprechend der Periodenzahl des Atoms. Daher ist der Kern elektrisch positiv geladen. Um ihn herum stellt man sich konzentrische Elektronenschalen vor, die insgesamt mit genauso vielen Elektronen besetzt sind, wie es Protonen gibt. Dabei trägt die erste Schale maximal 2 Elektronen, die zweite und dritte Schale maximal acht und die Vierte und Fünfte 18 Elektronen. In der Regel sind immer die kernnahen Schalen besetzt, weil die Anziehung dort aufgrund der Nähe zum Kern höher ist. Die stabilste Anordnung für das Atom ist eine volle Besetzung der nächsten Schalen. Dies ist aber nur für die Gruppe der Edelgase so gegeben. Atome in lebenden Organismen haben keine voll besetzte äußere Schale und sind daher um so reaktionsfreudiger desto mehr Elektronen ihnen zu einer vollen äußeren Schale fehlen bzw. zu viel sind. Die Atome sind stets bestrebt, den stabilsten Zustand einzunehmen. Um dies zu erreichen verfolgen sie zwei Methoden:
Die erste ist das Teilen von Elektronen. Teilen sich zwei Atome Elektronen liegen sie nah beieinander, sodass sich ihre Orbitale überschneiden und die Elektronen mal um das eine Atom, mal um das Andere kreisen. Dabei gehen sie eine kovalente Bindung ein. Es ist immer gerade das Atom positiv geladen, um das das zu teilende Elektron nicht kreist. Außerdem unterscheidet man zusätzlich zwischen Einfach- und Doppelbindungen, je nachdem ob sich die zwei Atome ein oder mehrere Elektronen teilen.
Die zweite Methode ist der Elektronenaustausch. Dabei gibt ein Atom einzelne Elektronen, die nur leicht auf der äußeren Schale gebunden sind, an ein anderes Atom ab. Dadurch können beide zu einer voll besetzten äußeren Schale und damit zu einem stabileren Zustand gelangen. In diesem Fall spricht man von einer ionischen Bindung, da beide Atome nun als Ionen - als elektrisch geladene Teilchen - vorkommen.

Im Allgemeinen können Kraftbeiträge vom Strecken von Bindungslängen, dem Öffnen und Schließen von Winkeln sowie Rotationen um einzelne Bindungen eine Rolle spielen. In ECEPP/3 sind die Bindungslängen und -winkel für jede kovalente Bindung fest vorgegeben. Der einzige Freiheitsgrad ist die Rotation um Bindungen, sodass verschiedene Torsionwinkel ausgebildet werden.
Der Torsions- oder auch Diederwinkel beschreibt den Winkel zwischen zwei Ebenen, die zwischen jeweils drei von vier Atomen aufgespannt werden. Spannen die Atome A, B und C die erste Ebene auf und B, C und D die zweite, so beschreibt der Torsionswinkel die räumliche Anordnung der vier Atome zueinander. Die Winkel werden nochmals in Φ-, Ψ und Ω-Winkel eingeteilt. Φ bestimmt dabei den Abstand zweier Carbonyl-Kohlenstoffatome (Kohlenstoffatom mit doppelt gebundenen Sauerstoffatom), Ψ den Abstand zweier Amid-Stickstoffe (Stickstoff-Rest-Verbindung anstatt der Hydroxylgruppe) und Ω zwischen zwei α-Kohlenstoffen (Kern einer Aminosäure).

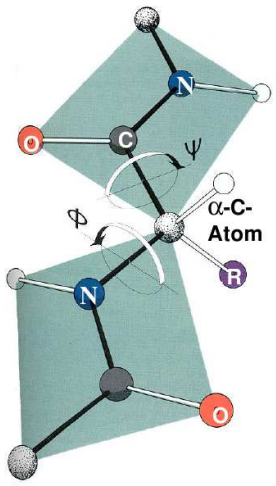

Abbildung 3.1: Torsionswinkel innerhalb eines Molekülausschnittes. Φ ist der Winkel zwischen C, N, C-α und dem nächsten C; Ψ zwischen N, C-α, C, N; Ω zwischen C-α, N, C, C-α. Ausgehend vom C-α-Atom in der Mitte sind die Rotationsachsen der Torsionswinkel Φ und Ψ eingezeichnet. Der Winkel Ω tritt zu einer weiteren Ebene an einem nächsten C-α-Atom in der Kette auf. Quelle: [12]

Die Torsionskräfte entstehen durch die Rotation der kovalenten Bindungen. Eine Einfachbindung zwischen zwei Atomen erlaubt zumeist eine freie Rotation eines Moleküls relativ zum Anderem. Doppelbindungen dagegen sind kürzer und stärker als Einfachbindungungen und fixieren dadurch die Anordnung der Atome. Sie treten nur mit $\Omega \approx 0°$ und $\Omega \approx 180°$ auf.

3.1.2 Nicht-kovalente Bindungen

Nicht-kovalente Bindungen sind keine chemischen Bindungen, sondern elekrostatische Wechselwirkungen. Dazu zählen Coulomb- und van-der-Waals-Wechselwirkungen sowie Kräfte, die von Wasserstoffbrückenbindungen ausgehen.
Das Coulomb-Gesetz besagt, dass sich zwei gleichnamige Ladungen abstoßen und zwei ungleichnamige anziehen. Für die resultierende Energie sind die Ladungen und ihr Abstand ausschlaggebend. Die Energie hängt proportional mit dem Produkt der Ladungen und antiproportional mit dem Abstand zusammen.
Die van-der-Waals-Wechselwirkungen sind sehr schwache Wechselwirkungen zwischen einzelnen Atomen oder auch verschiedenen Molekülen. Sie treten z. B. zwischen Dipolen auf. Ein Dipol ist in diesem Fall ein Atom oder Molekül, das zwei räumlich getrennte elektrische Pole aufweist. Das heißt eine Seite ist elektrisch positiv geladen, die Andere hingegen elektrisch negativ. Bei Atomen herrscht aufgrund der sich bewegenden Elektronen auf den Elektronenbahnen eine stetige Ladungsverschiebung vor. Diese kann zu dipol-ähnlichen Eigenschaften führen. So kann eine Ladungsverschiebung in einem Atom, eine Ladungsverschiebung in einem anderen hervorrufen und dies führt sich kettenartig fort. Auf Grund wechselnder Ladungsverteilungen sind diese Kräfte nur schwach und wirken nur innerhalb sehr kurzer Reichweiten.

Wasserstoffbrückenbindungen treten häufig als Fixierung innerhalb von α-Helices auf. Sie werden zwischen den sogenannten Wasserstoffdonator und -Akzeptor geschlossen. Der Wasserstoffdonator (z. B. Stickstoff oder Sauerstoff) ist kovalent an ein Wasserstoffatom gebunden. Dieser ist negativ geladen, sodass das Wasserstoffatom positiv geladen ist. Kommen sich das Wasserstoffatom und ein negativ geladener Akzeptor nah, so bildet sich eine Wasserstoffbrücke aus. Einzelne Atome gehören der einen (Donator) oder anderen Gruppe (Akzeptor) an als auch Atome die beiden oder keiner Gruppe angehören.

3.2 Das Proteinkraftfeld ECCEP/3

ECEPP/3 (Empirical Conformational Energy for Peptides and Proteins) ist ein klassisches Kraftfeld, dessen Parametrisierung auf empirischen Werten aus Messungen an kleinen Peptiden und quantenmechanischen Berechnungen basiert. Wie vorher schon erwähnt, sind dabei Bindungslängen und -winkel fest und nur die Torsionswinkel frei.
In die Energieberechnung gehen Coulomb- und van-der-Waals-Wechselwirkungen, Energien von Wasserstoffbrückenbindungen und Torsionskräfte ein.

Insgesamt berechnet es sich nach folgender Gleichung:

$$E = \sum_{(i,j)} 332 \frac{q_i q_j}{\epsilon r_{ij}} + \sum_{(i,j)} (\frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}}) + \sum_{(i,j)} (\frac{C_{ij}}{r_{ij}^{12}} - \frac{D_{ij}}{r_{ij}^{10}}) + \sum_{l} U_l (1 \pm cos(\eta_l \xi_l)) \qquad (3.1)$$

Darin beschreibt der erste Term das Coulomb-Potential. Es wird in den meisten Proteinkraftfeldern zur Beschreibung der elektrostatischen Wechselwirkung zwischen Atomen verwendet. Im zweiten Term werden die van-der-Waals-Wechselwirkungen berücksichtigt. Sie werden durch das 12-6-Lennard-Jones-Potential beschrieben. Dabei bilden sie den anziehenden Teil des Lennard-Jones-Potentiales. Allerdings fallen sie mit der sechsten Potenz des Abstandes ab. Auch die Wasserstoffbrückenbindungen (der dritte Term) werden über das Lennard-Jones-Potential beschrieben. Sie sind aber noch kurzreichweitiger als die van-der-Waals-Wechselwirkungen und fallen mit der zehnten Potenz des Abstandes im 12-10-Potential ab.
Dabei beschreiben A, B, C und D jeweils spezielle Parameter, die für die entsprechende Atompaarung für dieses Merkmal experimentell bestimmt werden konnten.

3.2.1 Lennard-Jones-Potential

Das Lennard-Jones-Potential nähert die Wechselwirkung zwischen nicht-kovalent gebundenen Atomen an. Die generelle Form ist die Folgende:

$$V = 4\varepsilon \left\{ \left(\frac{\sigma}{r}\right)^m - \left(\frac{\sigma}{r}\right)^n \right\} \qquad (3.2)$$

Der vordere Teil der Gleichung beschreibt die abstoßenden Kräfte, der hintere die Anziehenden. Die abstoßenden Kräfte beruhen darauf, dass sich die Orbitale der Atome,

die sich zu nahe kommen, überschneiden. Nach der Pauli-Repulsion stoßen sich dann Elektronen mit dem gleichen Spin ab. Die Anziehung hängt von der Bindungsenergie des Atompaares ab. Diese ist umso größer je näher sich die Atome kommen und fällt mit der Entfernung stark ab.
Das 12-6-Lennard-Jones-Potential, wie es für die van-der-Waals-Wechselwirkungen benutzt wird, ist in der nachfolgenden Abbildung skizziert.

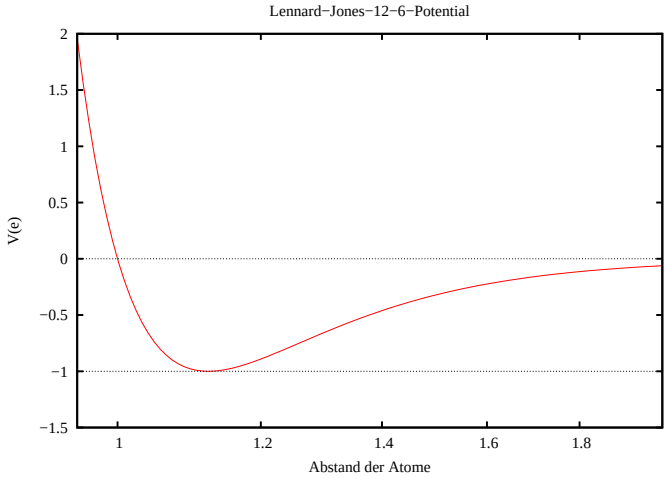

Abbildung 3.3: Das 12-6-Lennard-Jones-Potential. Das 12-6-Lennard-Jones-Potential nähert in ECEPP/3 die van-der-Waals-Wechselwirkungen an. Es wird die Energie gegen den Abstand zweier Atome aufgetragen. ε beschreibt die Tiefe des Potentialminimums und σ die Nullstelle. Der Abstand des Minimums vom Ursprung wird durch die Beziehung $x_{min} = 2^{\frac{1}{6}}\sigma$ beschrieben.

Die Wasserstoffbrückenbindungen werden in ECEPP/3 ebenfalls durch ein Lennard-Jones-Potential beschrieben. Allerdings nicht wie die van-der-Waals-Kräfte mit dem 12-6-Potential, sondern mit dem 12-10-Potential. Außerdem gehen andere Parameter ein, sodass später in der Berechnung zwischen Wasserstoffbrücken- und van-der-Waals-Bindung unterschieden werden muss.
Der letzte Term der Gleichung charakterisiert die Torsionskräfte. Die Rotation bedarf vergleichsweise wenig Energie, sodass man sie als weichen Freiheitsgrad bezeichnet.

Allerdings muss man hinzufügen, dass nicht alle Terme bedingungslos in die Energiebilanz eingehen. So werden alle Wechselwirkungen zwischen zwei Atomen, die zwei Bindungen und weniger voneinander entfernt sind, nicht mit einbezogen. Für die Simulation ist nur die Energiedifferenz zwischen verschiedenen Konformationen relevant. Die Energien für die Wechselwirkungen zwischen diesen nahen Atompaaren sind aber über die Simulation hin konstant. Somit können sie innerhalb der Simulation ignoriert werden.
Die Bindungspartner, die genau drei Bindungen voneinander entfernt sind - sogenannte 1-4-Nachbarn - gehen nur mit dem Faktor $\frac{1}{2}$ im abstoßenden van-der-Waals-Term ein. Dieser Wert ist auf Erfahrung aufgebaut und bringt in das klassische Kraftfeld quantenmechanische Überlegungen ein. Dadurch wird begünstigt, dass innerhalb der Simulation nahgelegene Atome sich aneinander vorbei bewegen können.

Hinzu kommt, dass für jede Paarung entweder der van-der-Waals-Term oder der Wasserstoffbrückenbindungs-Term relevant ist. Dies hängt davon ab, ob diese Paarung aufgrund ihrer Atomtypen eine Wasserstoffbrücke ausbilden kann.

3.3 Simple Molecular Mechanics for Proteins (SMMP)

SMMP[13][14][15] ist ein open source Paket verschiedener Algorithmen zur Simulation von Proteinen. Dazu stellt es mehrere Kraftfelder zur Verfügung, die das reale Potential, das man nicht kennt, annähern sollen.
Schon vom Levinthal-Paradoxon (s. 2.3.1) her weiß man, dass es selbst unter vereinfachenden Annahmen zu lange dauern würde, wenn das Protein jede mögliche Konformation durchprobieren würde. Die Energieberechnung zum Auffinden des Minimums wäre daher nicht schneller. Vielmehr versucht man mittels Monte-Carlo-Simulation von einer Konformation ausgehend zufällige Änderungen vorzunehmen und einen absteigenden Weg durch die Energielandschaft zu wählen. Doch allein das Kriterium des abwärts Gehens, garantiert nicht, dass man das globale Minimum innerhalb der Landschaft erreicht. Aufgrund der rauhen Energielandschaft landet man leicht in einem lokalen Minimum und findet nicht mehr heraus. Deshalb arbeitet SMMP z. B. mit dem Metropolis-Algorithmus, der manchmal auch ein Wegstück „bergauf“ wählt.

3.3.1 Monte-Carlo-Simulation

Die Monte-Carlo-Simulation ist ein stochastisches Verfahren, das auf oft wiederholten Zufallsexperimenten basiert. Dabei löst die Monte-Carlo-Simulation nicht nur stochastische Probleme, sondern auch Deterministische, wie z. B. mehrdimensionale Integrale:

$$\int_{x\in\Omega} P(x)A(x)d^n x$$

Die Integration verläuft in diesem Fall über alle möglichen Konformationen des Proteins ($x \in \Omega$). Da dies aber eine unendlich große Anzahl ist, versucht man den Raum Ω mit zufälligen Konformationen abzudecken. $A(x)$ ist die Observable. Die Häufigkeit mit der eine bestimmte Konformation auftritt wird über $P(x)$, der kanonischen Verteilung, ermittelt. Die kanonische Verteilung beschreibt die Wahrscheinlichkeit eines Zustandes in einem thermodynamischen System. Diese Wahrscheinlichkeit beschreibt sich wie folgt:

$$P(x) = \frac{1}{Z}e^{-\beta E} \text{ mit } \beta = \frac{1}{k_B T}$$

Dabei ist k_B die Bolzmannkonstante, T die Temperatur des System und Z die Zustandssumme, die selbst so beschrieben wird:

$$Z_k(N,V,T) = \sum_{x\in\Omega} e^{-\frac{E_x}{k_B T}}$$

In unseren Simulationen wird z. B. ein kanonisches Ensemble behandelt - ein System mit konstanter Anzahl an Teilchen N, Volumen V und Temperatur T.
Die Konformationen ergeben sich aus einer zufälligen Änderung an einem zufällig gewählten Freiheitsgrad. Der Freiheitsgrad ist ein beliebiger Torsionswinkel. Dadurch verändern sich die kartesischen Koordinaten aller Atome, die hinter diesem Winkel gelegen sind. Dies resultiert in anderen relativen Abständen der Atome untereinander und die Energie des System muss neu berechnet werden.
Eine solche neue zufällige Konformation wird in der Monte-Carlo-Simulation als Schritt auf dem Weg zum Minimum angenommen, wenn die Energie geringer als die der alten Konformation ist. Damit ist ein stetiges Abfallen der Energie garantiert. Jedoch besteht aufgrund der rauhen Energielandschaft die große Gefahr in einem lokalen Minimum zu landen. Um dies zu verhindern wird der Metropolis-Algorithmus verwendet.

3.3.2 Metropolis-Algorithmus

Dieser Algorithmus dient dazu den gesamten Konformationsraum einschließlich des globalen Minimums zu erkunden. Der Algorithmus erlaubt es mit einer gewissen Wahrscheinlichkeit auch einen Iterationsschritt zu akzeptieren, der eine höhere Energie aufweist.
Dabei wird beibehalten, dass ein Schritt akzeptiert wird, wenn die Energie der neuen Konformation geringer ist. Für den anderen Fall wird die Energiedifferenz zwischen alter und neuer Konformation betrachtet. Die Akzeptanzwahrscheinlichkeit p_a beträgt

$$p_a = \min(1, e^{-\frac{\Delta E}{k_B T}}).$$

Zur Entscheidung wird dieser Wert mit einer Zufallszahl zwischen 0 und 1 verglichen. Ist p_a kleiner der Zufallszahl wird der Schritt angenommen, ansonsten abgelehnt.

3.3.3 Parallel Tempering

Der Metropolis-Algorithmus garantiert, dass bei einer unendlich langen Simulation das globale Minimum gefunden wird. Jedoch wird der Faltungsweg eine lange Zeit in einem kleineren Gebiet des Faltungstrichters verlaufen, da nur kleine Änderungen an der Konformation geschehen. Das Parallel Tempering, auch Replica Exchange genannt, deckt innerhalb kürzerer Rechenzeiten ein weiträumigeres Gebiet ab, indem es größere Sprünge erlaubt[16].
Die Idee, die dahinter steckt, ist M Kopien eines Proteins bei verschiedenen Temperaturen zu simulieren. Dabei ist die Kopie bei der höchsten Temperatur dazu in der Lage den größten Raum abzutasten, während Kopien bei geringeren Temperaturen innerhalb kleinerer Bereiche suchen.
Das Problem bei kleineren Temperaturen ist, dass der Weg in einem lokalen Minimum enden könnte. Parallel Tempering sieht unter bestimmten Bedingungen einen Tausch der Konformationen bzw. Temperaturen zweier Kopien vor. Durch eine Erhöhung der Temperatur kann eine Konformation zusätzliche Energie erhalten, um evtl. ein solches

lokales Minimum zu überwinden.
Die M Kopien sind eigenständige kanonische Ensembles, die alle bei einer anderen Temperatur T_i simuliert werden. Dabei sind die Temperaturen in der Regel aufsteigend entlang der Prozessoren sortiert. Ein Tausch erfolgt meist zwischen zwei benachtbarten Kopien, wenn:

$$P_{ij} = e^{-\Delta\beta\Delta E}.$$

Für die Konformationsbestimmung der nativen Struktur sieht es so aus, als wenn Rechenzeit vergeudet würde, da man M-mal das gleiche Protein simuliert. Letzendlich hat sich aber herausgestellt, dass eine Parallel Tempering Simulation mehr als $\frac{1}{M}$ mal so effizient ist als eine Standard Monte Carlo Simulation bei einer Temperatur um das Minimum zu finden. Diese gesteigerte Effizienz stammt daher, dass Konformationen von niedriger Temperatur erlaubt wurde Regionen zu erreichen, die sie nie hätten erreichen können, wenn sie bei einer normalen Simulation bei einer Temperatur und M-mal so viel Zeit ausgeführt worden wären (frei übersetzt nach [16]).
Für die Entschlüsselung des Faltungsweges ist aber der Weg das Ziel. Parallel Tempering führt eine Reihe von möglichen Faltungsvorgängen durch. Dadurch wird die gesamte Energielandschaft erkundet und es können Pfade erkannt werden. Diese können evtl. Aufschluss darüber geben nach welchen Kriterien sich ein Protein faltet.
Hinzu kommt, dass eine solche Aufgabe gut für Parallelrechner und Cluster geeignet ist. Denn es ist ein einfaches paralleles Rechnen möglich und der Kommunikationsaufwand sehr gering. Daher soll er nun auf der Cell Broadband Engine implementiert werden.

4 Die Cell Broadband Engine

Die Proteinfaltung ist ein rechenintensives Themenfeld der Biophysik. Der Zeitaufwand für die Energieberechnung einer Konformation steigt in ECEPP/3 quadratisch mit der Anzahl an Atomen. Um dem Aufwand bei der Suche nach der optimalen Konformation gerecht zu werden, kommt man nicht umhin auf parallele Rechnerarchitekturen zu setzen. Der Trend geht zu Mehrkern-Prozessoren, weil sich dadurch leichter mehr Transistoren auf einem Chip unterbringen lassen. Wollte man stattdessen nur die Taktrate erhöhen, würde der Stromverbrauch immens steigen. Mit einer effizienten Ausnutzung von Parallelrechnern kann man aber auch bei geringeren Taktraten gleiche Leistungen erreichen.
Dieses Kapitel soll die Hardware der Cell Broadband Engine (CellBE) und die daraus für die Programmierung resultierenden Konsequenzen vorstellen.

4.1 Die Hardware

Die Cell Broadband Engine wurde entwickelt um an Effizienz mit FPGAs mithalten zu können und an die Vielseitigkeit von Heim-PCs heran zu reichen. Eigens dafür schlossen sich die Firmen Sony, Toshiba und IBM zu einer Allianz, bekannt unter dem Namen STI, zusammen. Die neuartige Architektur, die über 4 Jahre hinweg entwickelt wurde, zeichnet sich dabei durch hohe Rechenleistung bei vergleichbar kleiner Chipgröße, geringem Stromverbrauch und damit verbundener Abwärme aus.
Dieser Mikroprozessor verbindet heterogene Systemkomponenten. Da wäre zum einem ein PowerPC Processing Element - kurz PPE - als Initiator und zum anderen 8 Synergistic Processing Elements - kurz SPE - als schnelle Recheneinheiten. Verbunden werden sie über den Element Interconnect Bus (EIB) mit großer Daten-Bandbreite.
Doch neun Kerne allein machen die Cell Broadband Engine nicht ohne weiteres zu dem Rechenkünstler, die sie ist. Mit zeitgleichem Multithreading sowie Pipelining schafft man ein hohes Maß an Parallelität. Top-Performance erreicht man aber erst mit der Ausnutzung der Vektorrecheneinheit auf den SPEs - auch wenn dies mit zusätzlichem Programmieraufwand und komplexerem Code einhergeht.
Seinen ersten Einsatz fand der Cell-Prozessor in der Sony Playstation 3. Anwendungen in der Medienverarbeitung können die Vektoreinheit effizient ausnutzen. So findet er ebenso in Toshibas Home Entertainment Produkten seinen Platz. Spitzenplätze erreicht er auch beim wissenschaftlichen Rechnen in IBM's BladeServern. Das Cell-basiertes Hybridsystem Roadrunner at Los Alamos National Laborator setzte sich im Juni 2008 als erster Peta-Flop-Rechner auf Platz eins der Top500[17]. Die ersten vier Plätze der Green500 werden ebenfalls von der Cell Broadband Engine basierten Systemen belegt (Stand: Juni 2009)[18].

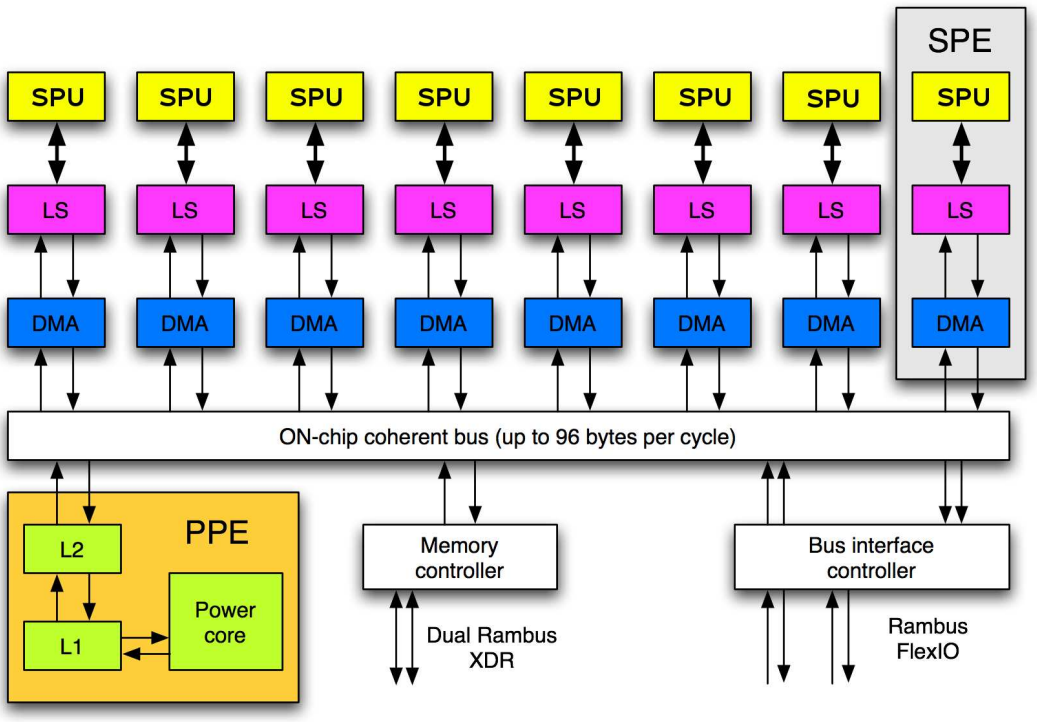

Abbildung 4.1: Schematische Darstellung des Cell-Prozessors. Diese Abbildung stellt die Struktur der Cell Broadband Engine dar. An den Element Interconnect Bus (EIB - Mitte) sind die acht Synergistic Processing Elements (SPE - oben), das I/O-Interface (rechts unten), der Memory Inferface Controller (MIC - Mitte unten) sowie das PowerPC Processing Element (links unten) angeschlossen. Der Prozessor des PPE wird durch 2 Cache-Stufen unterstützt. Der Memory Flow Controller (MFC) des SPE ist von der SPU getrennt und kann gleichzeitig auf den Local Store (LS) zugreifen. Quelle: [19]

4.1.1 PowerPC Processing Element (PPE)

Das PPE ist ein 64-Bit Power-PC - ein Performance Chip (PC) mit Leistungsoptimierung durch erweiterte RISC-Architektur. Bei dieser Erweiterung handelt es sich um ein zusätzliches Instruction Set für Vektorrechnung. Dieses kann die Ausführung auf den SPEs vorbereiten. Weiter verfügt es über zwei 32 KB große Level-1-Caches – jeweils einen für Daten und einen für Befehle. Diese werden durch einen 512 KB großen Level-2-Cache unterstützt. Die Taktrate des Prozessors ist 3,2 GHz.
Das PPE ist die Programmsteuereinheit, dessen Aufgabe die Kontrolle der SPEs ist. Es kann die Ausführung auf den SPEs starten, unterbrechen und stoppen. Dabei muss es vor allem ihre Einsätze synchronisieren. Es übernimmt also die Verwaltungsarbeit und übergibt die rechenintensiven Teilaufgaben an die SPEs. Mit der Fähigkeit zwei Threads gleichzeitig ausführen zu können, kann es neue Threads für die SPEs erstellen und parallel sequentielle Programmteile abarbeiten.

4.1.2 Synergistic Processing Element (SPE)

Die SPEs sind acht identische on-Chip Co-Prozessoren. Sie bestehen jeweils aus einer eigenen Synergistic Processing Unit (SPU), einem eigenen Local Store (LS) sowie einem Memory Flow Controller (MFC). Die SPU verfügt über eine Arithmetical Logical Unit (ALU), die eine Operation auf jedes Element eines Vektors anwenden kann. Zu den mächtigen Operationen des SPU Instruction Sets gehört beispielsweise ein Multiply-Add-Befehl. In einem Prozessortakt können damit in den 128 128-Bit Registern zwei Vektoren der Länge 4 multipliziert und das Ergebnis auf einen dritten Vektor addiert werden - das macht 8 Floating-Point-Operationen pro Takt bzw. 25,6 GFlop/s. Kann man diese Funktionen richtig ausnutzen, erzielt man hohe Rechenleistung.
Bei einem 256 KB großen LS, der nicht durch einen Cache unterstützt wird, müssen die Daten schnell ausgetauscht werden können. Dies übernimmt der MFC – Hardware die den Datentransfer mittels Direct Memory Access (DMA) zwischen Hauptspeicher (HS) und LS bzw. zwischen LS und LS verschiedener SPEs auslöst. Da mit dem Cache auch die Caching-Strategien fehlen, muss der Programmierer selbst dafür Sorge tragen, dass z. B. nicht mehrere SPEs auf eine Hauptspeicheradresse schreiben. Allerdings liegen auch alle Optimierungsmöglichkeiten beim Programmierer.

4.1.3 Element Interconnect Bus (EIB)

Der EIB ist ein 4-fach unidirektionaler Ringbus. Jeweils zwei Ringe versenden mit und zwei gegen den Uhrzeigersinn. Somit ist die maximale Anzahl an Stationen, für eine Nachricht bei 12 Teilnehmern am Bus, sechs. Jeder Teilnehmer kann mit 16 Byte pro Takt lesen und schreiben. Wichtig für den Durchsatz ist seine Taktrate. Diese liegt bei der Hälfte des Prozessors. Bei 4 Ringen, die jeweils 3 Operationen von 16 Byte alle zwei CPU-Takte übertragen, kommt man auf eine Übertragungsgeschwindigkeit von 96 Byte pro Takt. Die 12 Teilnehmer sind zum einem die PPE und die SPEs, hinzu kommen der Memory Inferface Controller (MIC) und zwei I/O Interfaces. Der MIC ist zuständig für den Zugriff auf den HS, während die I/O Interfaces zum Datenaustausch zwischen den zwei Prozessoren eines Blades dienen.

4.2 Die Programmierung

Aufgrund der Hardwarebeschaffenheit der Cell Broadband Engine rückt für die Programmierung das Master-Worker-Konzept in den Vordergrund. Dabei ist ein Prozess als Master ausgezeichnet, der die Aufgaben verwaltet und den Workern die dazu zugehörigen Daten bereitstellt. Die Worker arbeiten ihre Aufgabe ab, liefern das Ergebnis zurück und bekommen eine neue Aufgabe. Die SPEs erfüllen mit ihrer starken Rechenleistung die Aufgabe der Worker, das PPE übernimmt den Teil des Masters.
Die Parallelisierung der Cell Broadband Engine lässt sich grundsätzlich in zwei Modelle einteilen. Beim datenparallelen Modell gibt es ein SPE-Programm, dass auf allen SPEs mit verschiedenen Datensätzen ausgeführt wird. Bei diesem Modell kann man bei einer gleichmäßigen Datenaufteilung mit einer guten Lastbalancierung rechnen.

Das zweite - das taskparallele - Modell entspricht einem Pipelining über die Prozessoren. Ein Datensatz wird von SPE zu SPE geschickt und erfährt auf jedem einen anderen Bearbeitungsschritt. Um dabei eine gute Ausnutzung der Prozessoren zu erzielen sollten die Ausführzeiten der einzelnen Aufgaben nicht zu sehr voneinander abweichen. Ansonsten entstehen zu große Wartezeiten und die Funktion mit der längsten Ausführung bestimmt die Gesamtausführung. Sind die Zeiten aber annähernd gleich, bekommt man ab dem ersten fertigen Datensatz in gleichmäßigen Abständen Ergebnisse geliefert. Eine Mischung aus beiden Modellen ist ebenso möglich.

Doch wie oder womit programmiert man auf der Cell Broadband Engine im Detail? Die maschinennächste Programmierung ist die Assembler-Programmierung. Aber es gibt auch andere Methoden.
Das Software Development Kit (SDK) stellt zusätzlich zum normalen C/C++-Sprachumfang Intrinsics zur Verfügung. Diese erlauben das Starten von Routinen auf den SPEs von dem PPE, regeln Datentransfer, nutzen die SIMD-Vektoreinheit auf den SPEs, etc.
Eine Abstraktionsstufe höher steht die Programmierung mit den Schnittstellen OpenMP und MPI für shared-Memory bzw. distributed-Memory-Systeme. OpenMP[20] würde dabei ein Ersatz für die Intrinsics zum Starten von SPE-Routinen darstellen, da beides parallele Threads auf den SPEs ermöglicht. MPI dagegen arbeitet auf Prozess-Ebene. Damit kann man bei einem Cell-Cluster über die Knoten parallelisieren.
Noch eine Stufe darüber steht das Framework CellSuperScalar (CellSS). Es bietet ähnlich OpenMP die Möglichkeit mit Pragmas innerhalb eines sequentiellen Codes Bereiche zu markieren, die parallel gerechnet werden. Der Compiler macht dann daraus über die SPEs parallelisierten Binary-Code, der auch die SIMD-Einheit ausnutzt. Der Vorteil von CellSS ist es, dass der Code ebenso für andere Architekturen compilier- und ausführbar ist.
Doch so einfach ist es nicht ein Programm auf die Cell Broadband Engine zu portieren. Zusätzlich zu dem generellen Gerüst, um Threads auf den SPEs zu starten, und ein paar SIMD-Funktionen bedarf es mehr. Es gilt eine Parallelisierung über mehrere SPEs vorzunehmen. Die Daten dafür müssen verteilt werden. Außerdem muss dabei besondere Acht auf Datenstrukturen gelegt werden. Für die Berechnung mit SIMD-Funktionen müssen meist entsprechende Anpassungen am Algorithmus vorgenommen werden. All dies soll nun im Detail erklärt werden.

4.2.1 PPE: Aufteilung über die SPEs

Im Folgenden werde ich das grundlegende Vorgehen auf Seiten des PPE vorstellen, das zum Anstoßen der SPE-Threads benötigt wird. Allerdings werde ich dabei nicht explizit auf die einzelnen Parameter in den Aufrufen eingehen, da dies an dieser Stelle zu weit führen würde. Für Details diesbezüglich verweise ich auf Programmierhandbücher[21][22].

Der wichtigste Bestandteil für einen SPE-Thread ist sein Context. Dieser ist die logische Repräsentation des SPE. Er hält Informationen über das Executable, den Programmcounter, Umgebungsvariablen, etc. und wird auf dem PPE erzeugt und dem Aufruf dem

SPE mitgegeben. Mit dem Befehl `spe_context_create` wird dieser angelegt. In einem nächsten Schritt soll er mit dem SPE-Executable verbunden werden. Verfolgt man allein das datenparallele Modell, dann kann sofort mit `spe_program_load` ein Handle, das ein Pointer auf das Executables ist, übergeben werden. Verfolgt man allerdings das taskparallele Modell oder eine Mischung aus beiden, dann müssen den SPEs verschiedene Executable zugewiesen werden. Dazu weist man dem Handle explizit eines zu. Dies geschieht über `spe_image_open`. Erst danach verbindet man das Handle mit dem Context. Für ein Programm, das zwei verschiedene SPE-Executable anstoßen soll, sieht das wie folgt aus:

```
// Namen der SPU-Executables
char spe_names[2][20] = {"spu1/spu_main1","spu2/spu_main2"};
// Schleife ueber die SPEs um:
for(num=0; num<2; num++){
 // den Context zu erstellen
 if((data[num].spe_ctx = spe_context_create (0, NULL)) == NULL){
  perror("failed creating context"); exit(1);
 }
 // das Image des Executables zu laden und
 if (!(program[num] = spe_image_open(&spe_names[num][0]))){
  perror("fail opening image"); exit(1);
 }
 // mit dem Context zu verbinden
 if(spe_program_load(data[num].spe_ctx, program[num])){
  perror("failed loading program\n"); exit(1);
 }
}
```

Als nächstes ist es wichtig, das SPE-Programm in einem eigenständigen Thread zu starten. Sonst wartet das PPE mit seiner weiteren Ausführung auf die Beendigung des SPE. Dazu definiert man sich folgende Funktion, die einen neuen Thread startet und diesen auf dem SPE ausführt:

```
//erzeugt Thread auf dem SPE und fuehrt dort den Context aus
void * spu_pthread(void *arg){
 spu_data_t *datp = (spu_data_t *)arg;
 uint32_t entry = SPE_DEFAULT_ENTRY;
 if(spe_context_run(datp->spe_ctx,&entry,0,datp->argp,NULL,NULL)<0){
  perror ("Failed running context"); exit (1);
 }
 pthread_exit(NULL);
}
```

Bei dem Aufruf kann ein Argument an das SPE-Programm übergeben werden. Deswegen werden die Daten für den SPE-Thread für ein SPE gebündelt in einer Struktur gehalten. Diese enthält den Context, den pthread sowie den Argumentpointer.

```
1 //Datenstruktur fuer SPE-Threads
typedef struct spu_data{
3  spe_context_ptr_t spe_ctx;
   pthread_t pthread;
5  void *argp;
} spu_data_t;
7 //Instanzen
spu_data_t data[ANZAHL_SPES];
```

Hat man für beide SPEs die Datenstruktur gefüllt und den pthread gestartet, laufen die Programme vollkommen unabhängig voneinander. Das PPE kann in dieser Zeit andere Arbeit leisten. Sei es für das Programm nötige Rechenarbeit oder die Vorbereitung neuer SPE-Threads. Dabei wartet es auf die Beendigung der SPE-Threads. Dies geschieht über `pthread_join`. Am Ende eines Programmes folgt dann nur noch das Schließen des Images: `spe_image_close` und das Zerstören des Contexts: `spe_context_destroy`, um den Speicher freizugeben.

4.2.2 Übertragung von Datenstrukturen

Es bedarf des DMA-Transfers, um die SPEs mit genügend Daten zu versorgen, damit diese ständig Berechnung durchführen können. Dieser ist darauf ausgelegt große Datenmengen vom HS in den LS zu transferieren. Für kleine Benachrichtigungen zur Synchronisation zwischen PPE und SPE gibt es die Kommunikation über Mailboxes.
Der DMA-Transfer kann sowohl von dem PPE als auch dem SPE eingeleitet werden, wobei die Namen der Befehle durch unterschiedliche Präfixe gekennzeichnet sind. Von dem PPE eingeleitet beginnen die DMA-Kommandos mit `spe_mfcio_` – von dem SPE mit `mfc_`. Für einen solchen Transfer wird vorher ein Tag vereinbart, der bei jedem Transfer mitgegeben wird. Insgesamt sieht ein Programm mit DMA-Transfer wie folgt aus:

```
//Tag reservieren
2 if((tag_id=mfc_tag_reserve())==MFC_TAG_INVALID){
   printf("SPE: ERROR - can't reserve a tag ID"); return 1;
4 }
//Daten aus dem Hauptspeicher in den LS holen
6 mfc_get(HS_Daten, LS_Daten, Groesse, tag_id, 0, 0);
do_something();
8 //Daten aus dem LS in den Hauptspeicher zurueckschreiben
mfc_put(HS_Daten, LS_Daten, Groesse, tag_id, 0, 0);
10 //Tag freigeben
mfc_tag_release(tag_id)
```

Zusätzlich kann der DMA-Transfer von LS zu LS erfolgen. Jedoch wurde dies in dieser Arbeit nicht verwendet, sodass an dieser Stelle nicht weiter darauf eingegangen wird.
Die maximale Transfergröße einer Übertragung liegt bei 16 kByte und Peak-Performance erreicht man bei Größen, die ein Vielfaches von 128 Byte betragen. Doch abgesehen von eher unüblichen Transfergrößen für den DMA-Transfer wie 1, 2, 4, 8 Byte muss die Datengröße ein Vielfaches von 16 Byte sein. Dies setzt voraus, dass jede Datenstruktur, sei es ein Array oder ein Objekt einer Klasse, ein Vielfaches von 16 Byte ist. Jede Datenstruktur so mit Padding aufgefüllt werden, dass ein solches Alignment gegeben ist.
Zu transferierende Daten dürfen nicht beliebig im Speicher abgelegt sein. Sowohl die

Hauptspeicheradresse als auch die Local-Store-Adresse zwischen denen die Daten übertragen werden sollen, müssen auf einer 16-Byte-Grenze liegen. Das heißt, dass die Adresse ohne Rest durch 16 teilbar ist.
Dies hat der Programmierer im Code selbst zu berücksichtigen. Dafür sind folgende Methoden vorgesehen:

- **der Attributzusatz** `__attribute__((aligned(16)))`
  sorgt dafür, das die Startadresse der zugehörigen Variable oder des zugehörigen Objektes auf einer 16-Byte-Grenze beginnt. Acht zu geben ist z. B. bei Arrays. Dieses Attribut stellt nur sicher, dass der Beginn des Arrays auf einer solchen Grenze gegeben ist. Will man aber vielleicht erst ab einem bestimmt Array-Element Daten übertragen, so ist nicht garantiert, dass auch dieses Element entsprechend auf einer 16-Byte-Grenze startet. Davon kann man nur ausgehen, wenn ein Array-Element 16 Byte oder ein Vielfaches davon lang wäre oder die Länge multipliziert mit dem Index ein Vielfaches von 16 ergibt.

- `malloc_align(size, alignment)`
  fordert dynamisch erzeugten Speicherplatz an, der auf einer bestimmten Byte-Grenze beginnt. Entsprechend des zweiten Parameters beginnt der angeforderte Speicherplatz an einer Adresse, die ohne Rest durch $2^{alignment}$ teilbar ist. Mit `malloc_align` angeforderter Speicher muss auch wieder mit `free_align` freigegeben werden. Außerdem sollte man auch hier davor gewarnt werden unachtsam mit Pointerarithmetik zu arbeiten.

Zudem ist für die Programmierung mit C++ zu beachten, dass die Default-Konstruktoren das normale `malloc` verwenden. Will man aber auch Klassenobjekte auf die SPEs übertragen, so muss das `new` und `delete` entsprechend überladen werden, damit die Objekte jeweils auf 16-Byte-Grenzen beginnen. Dafür ist es vorteilhaft eine Basis-Klasse anzulegen, in der dies implementiert ist und von der entsprechend abgeleitet werden kann.

4.2.3 SIMD-Funktionen nutzen

Für die effektive Ausnutzung der SPE-Rechenleistung muss man die Vektoreinheit mit ihren besonderen Funktionen nutzen. Die ALU des SPE verfügt über Vektorregister und über entsprechende Befehle, die einen ganzen Vektor von vier Floating-Point-Zahlen verarbeiten können. Damit erfüllt das SPE die Eigenschaft für eine SIMD (Single-Instruction-Multiple-Data)-Architektur nach der Flynnschen Rechnerklassifikation[23].
Eine Operation kann also gleichzeitig auf mehrere Daten angewendet werden. So können z. B. mit einer SIMD-Funktion zwei vierkomponentige Float-Vektoren aufaddiert werden. Das Ergebnis wird ebenfalls auf einem Vektor gespeichert.

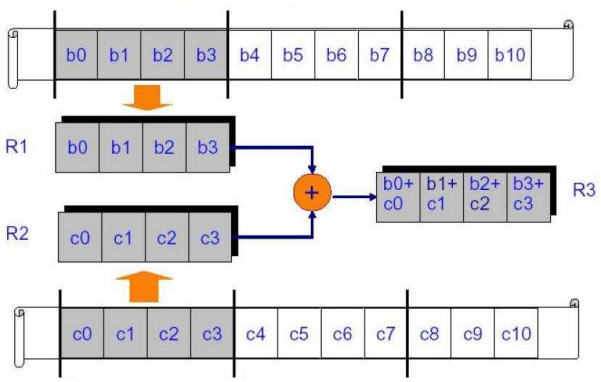

Abbildung 4.2: Single-Instruction-Multiple-Data (SIMD) Berechnung. Eine SIMD-Operation führt eine Operation auf zwei Vierer-Vektoren aus. Das Ergebnis wird ebenfalls wieder in einen solchen Vektor geschrieben. In diesem Fall wird eine Addition durchgeführt. Es sind aber auch boolsche Operationen oder Verschiebeoperationen möglich. Quelle: [24]

Mit dem Vektorbefehl Multiply-Add werden nicht nur vier, sondern sogar acht, Floating-Point-Operationen in einem ausgeführt. Das bedeutet: ist die Operations-Pipeline einmal gefüllt, erhält man damit acht Flop pro Prozessortakt, sofern man einfach genau rechnet. Benötigt man doppelte Genauigkeit halbiert sich die Leistung entsprechend auf vier. Genau dieser Faktor ist maßgeblich für die Beschleunigung des Programmes.
Das Füllen der Operationspipeline dauert einige Taktzyklen. Diese Latenzzeit beträgt für einfach genaue SIMD-Operationen sechs Takte, für doppelt genaue weitere sieben Takte. Dies zeigt auf, dass einzelne SIMD-Operationen länger brauchen als vier einzelne Berechnungen. Gewinn erzielt man erst bei vielen aufeinanderfolgenden Operationen.
Doch nicht jedes Problem lässt sich vektorisieren. Für eine Vektorisierung muss der Algorithmus des sequentiellen Programmes entsprechend angepasst werden. Einschränkend wirkt sich dabei aus, dass auf allen Elementen eines Vektors nur die gleiche Operation ausgeführt werden kann.
Gut geeignet für die Cell Broadband Engine sind Anwendungen, die mit Koordinaten arbeiten. Denn die mathematische Darstellung ist dabei schon vektoriell. So könnte man die drei kartesischen Koordinaten in einem Vektor ablegen. Für die Berechnungen mit den Vektoren stehen dann die SIMD-Funktionen zur Verfügung. Noch effektiver wäre es, wenn man die vierte Vektorkomponente ebenfalls nutzen könnte.
Doch Problemstellungen sind selten im mathematischen Sinne vektoriell zu lösen. Und auch wenn einige Berechnungen des Algorithmus vektoriell erfolgen können, ist es meist ein sehr geringer Teil des Programmes, der die Leistung der SPEs ausnutzt. Deshalb müssen die Algorithmen speziell darauf ausgelegt werden Vektorrechnung zu benutzen, damit keine Rechenleistung verschenkt wird.
Es lassen sich folgende grundlegende Bedingungen formulieren, die für eine Vektorisierung erfüllt sein müssen:

- Daten, die mit einer SIMD-Funktion bearbeitet werden sollen, müssen
  - völlig unabhängig voneinander sein

  - zu Additionen, Subtraktion, usw. gruppiert werden können
    (dies ist einfacher, wenn die Reihenfolge der Berechnungen vertauschbar ist)
  - idealerweise sollen die Berechnungen in einfacher Genauigkeit ein Vielfaches von vier betragen, oder bei doppelter Genauigkeit in Paaren vorkommen, um einen kompletten `vector` zu füllen und so die komplette SIMD-Leistung auszuschöpfen

- auf der gleichen Komponente innerhalb des Datentyps `vector` stehen, um miteinander verarbeitet zu werden; außerdem sollte die Position der Operanden nur selten verschoben werden müssen

Eine relativ einfache Parallelisierung kann man bei voneinander unabhängigen Schleifendurchläufen durchführen. Dazu werden die Daten von vier Durchläufen gesammelt und parallel ausgewertet. Große Einbußen im Laufzeitverhalten muss man hinnehmen, wenn man Verzweigungen benutzt. Die Fehleinschätzung des Zweiges - also das Nachladen der Instruktionen des zu durchlaufenden Zweiges - kostet 18-19 Taktzyklen. SIMD-Operationen müssen also entsprechend gut vorbereitet sein, um die mögliche Leistung auch wirklich zu erzielen.

4.3 Zusammenfassung

Die Cell Broadband Engine bietet eine gute Chance zur Laufzeitoptimierung. Für die Portierung auf die Cell Broadband Engine muss man einige Vorüberlegungen anstellen, inwieweit das Programm dafür geeignet ist.
Das wichtigste Kriterium ist die Frage nach der Vektorisierbarkeit. Dafür sollte eine hohe Datenunabhängigkeit gegeben sein, um flexibel die Operationsreihenfolge ändern zu können. Ohne eine effiziente Ausnutzung der Vektoreinheit wird man mit der Leistung nicht zufrieden sein. Denn die Programmierarbeit ist mit einem Mehraufwand verbunden, der sich auch auszahlen soll.
Der nächste Blick sollte auf die Datenmenge und die Übertragung gerichtet werden. Der LS verfügt lediglich über 256 KB, in die außer den Programmdaten auch das Programm selbst geladen werden muss. In diesem Zusammenhang spielt die Übertragungsgeschwindigkeit zwischen HS und LS, sowie die Latenzzeit eine wichtige Rolle. Es sollten immer große Datenmengen mit einem einzigen DMA-Transfer übertragen werden, da sich sonst die Latenzzeit in der Ausführzeit zu stark bemerkbar macht. Außerdem sollten die Berechnungszeiten zwischen einzelnen Übertragungen länger als die Übertragung selbst sein. Die Daten für diesen Zeitraum müssen dazu gesammelt in den LS passen.
Für das Erzielen einer schneller Übertragung sind einige Dinge bei der Erstellung der Datenstrukturen zu beachten. Die Daten müssen dazu zusammenhängend im HS auf einer 16-Byte-Grenze abgelegt werden. Außerdem müssen sie, um auf verschieden viele SPEs übertragen werden zu können, ein Vielfaches von 16 Byte groß sein. Dies muss außerdem vorliegen, wenn die Datengröße 16 KByte überschreitet und der Transfer gesplittet werden muss.

5 Die Implementierung

Die Implementierung des Proteinkraftfeldes ECEPP/3 beinhaltet die Erstellung eines SPE-Moduls, das die Energieberechnung der aus der Monte-Carlo-Simulation stammenden Konformationen übernimmt. Ziel ist es, die Zeit dieses rechenintensiven Teils des Programmes zu minimieren, um die Faltung größerer Proteine zu simulieren zu können. In den ersten beiden Kapiteln wurde dargestellt welche Bedeutung Proteinen und deren Faltung beizumessen ist und auf welchen Grundlagen das Kraftfeld ECEPP/3 basiert. Außerdem wurde die Cell Broadband Engine hardwaretechnisch vorgestellt und zusätzlich ein paar absolut notwendige Programmierbesonderheiten gezeigt, die im Verlauf dieses Kapitels ihre Anwendung finden werden.
Für die Umsetzung des Moduls musste man sich mit zwei wesentlichen Punkten befassen: die Datenstrukturen, deren Parallelisierbarkeit und Übertragung auf die SPE; und die Anpassung des Algorithmus. Dazu werden erste Vorüberlegungen sowie drei Entwicklungsstufen dargestellt.

5.1 Analyse

In der seriellen Implementierung verbraucht die Energieberechnung über 99% der Rechenzeit[25]. Daher gilt es sie effizient zu parallelisieren. Mit der Cell Broadband Engine besteht die Möglichkeit gleich mehrerer Parallelisierungen. Auf einer SPE kann die Berechnung theoretisch 4-fach parallelisiert werden, indem SIMD-Funktionen genutzt werden. Ein Cell-Blade verfügt über 16 SPEs, die gleichzeitig genutzt werden können. Außerdem besteht in einem Cluster von Cell-Blades die Möglichkeit der MPI-Parallelisierung über alle Blades.
Für die Proteinfaltung mit dem Parallel Tempering Monte-Carlo-Algorithmus bedeutet dies, dass man auf jedem Blade die Simulation bei einer Temperatur vornehmen könnte. Zusätzlich kann die Energieberechnung über bis zu 16 SPEs parallelisiert werden, wobei jedes Element für sich nochmals vektorisiert rechnet.

Die zu implementierende Berechnung besteht aus mehreren einfachen Operationen, die alle als SIMD-Funktionen auf der Cell Broadband Engine implementiert sind. Die daraus resultierenden Ergebnisse werden in einer Summe zusammengefasst. Die Reihenfolge der Aufsummierung spielt dabei keine Rolle. Die Abstandsberechnung zweier Atome ist der zeitlich dominierende Faktor der Berechnung. Diese ist für kartesische Koordinaten vektoriell. Diese Problemstellung sollte also für die Implementierung auf der Cell Broadband Engine geeignet sein.
Der Grund für die lange Rechenzeit bei großen Molekülen ist die Anzahl der nötigen Berechnungen für einen Simulationsschritt. Die Komplexität des Algorithmus liegt bei

$0(n^2)$. Die Anzahl der durchzuführenden Berechnungen vervierfacht sich demnach, wenn sich die Größe des Proteins (d. h. die Anzahl seiner Atome) verdoppelt.
Da der Rechenaufwand von der Anzahl der Daten abhängt, bietet sich hierbei das datenparallele Modell für die SPE-Ausnutzung an. Deshalb sollte das Problem bei einer gleichmäßigen Aufteilung gut über die SPEs skalieren.

5.1.1 Zielsetzung

Das Ziel dieser Arbeit ist es, die Energieberechnung für das Proteinkraftfeld ECEPP/3 möglichst effizient auf der Cell Broadband Engine zu implementieren. Dazu muss sie vektorisiert und über alle 16 SPEs eines Blades parallelisierbar sein.
Für die Implementierung ist ein geeigneter Algorithmus sowie passende Datenstrukturen im Hinblick auf die SIMD-Konformität zu konzipieren. Zu beachten ist dafür, dass der Algorithmus ein schnelles Abarbeiten von Daten ermöglicht. Die Daten müssen den Alignment-Anforderungen entsprechen und in möglichst großen Datenblöcken übertragen werden. Zudem müssen die Daten in den LS passen. Dafür sollte es von der Datenstruktur und vom Algorithmus her möglich sein die Daten über die SPEs aufzuteilen.
Der Quellcode soll gut lesbar und verständlich sein. Außerdem soll es möglich sein, Funktionalitäten wieder zu verwenden, um das Projekt weiterführen zu können.

5.1.2 Schnittstelle

Das SPE-Modul, das die Energieberechnung durchführen soll, soll aus einem bestehenden Fortranprogramm, das die Monte-Carlo-Simulation übernimmt, aufgerufen werden. Um die Implementierung des Moduls unabhängig vom Fortranprogramm zu ermöglichen, werden die Eingabedaten der Simulation aus Dateien eingelesen und in die entsprechenden Datenstrukturen, die als Schnittstelle zwischen PPE- und SPE-Programm zur Verfügung stehen, überführt. Später übernimmt das Fortran-Programm selbst die Bereitstellung der Daten.
Die Datenstrukturen müssen für die Übertragung zwischen PPE und SPE und umgekehrt in beiden Programmen bekannt sein und den Alignment-Anforderungen entsprechen. Außerdem sollte die zu übertragende Datenmenge so reduziert wie möglich sein aber zugleich von der PPE auch soweit für die Berechnung aufbereitet sein, damit die SPE nur zu rechnen braucht. Verzweigungen sind sehr zeitintensiv, sodass das PPE den SPEs möglichst viele Entscheidungen abnehmen sollte.

5.1.3 Programmiersprache C++

Die Wahl der Programmiersprache für dieses Modul ist auf C++ gefallen, um ein maximales Maß an Effizienz bei möglichst guter Lesbarkeit zu erzielen. Da C++ zur nativen Cell-Programmierung gehört, kann man bei sorgfältiger Programmierung die Systemresourcen optimal ausnutzen. Der Programmierer hat dabei die Optimierung selbst in der Hand. Bei den Intrinsics des SDK handelt es sich um Inline-Assembler-Code, sodass

sie eine verständliche und zugleich maschinennahe Nutzung möglich machen.
Eine Abbildung der realen Welt ist auch gut möglich. So kann z. B. ein Atom oder ein Torsionswinkel in einer eigenen Klasse modelliert werden. Dadurch wird ein intuitiverer Zugriff auf die Daten geschaffen.
Eine Kapselung der Daten resultiert auch in einer besseren Lesbarkeit des Quellcodes. Zudem kann das Programm durch das Klassenprinzip in funktionale Einheiten gegliedert werden. Dadurch kann auch ein Teil der Komplexität in den Klassen „versteckt“ und Wiederholungen vermieden werden. Außerdem können die Klassen wiederverwendet werden.
Wenig Wiederholung ist gerade für größere Programme wichtig, da sie möglichst mit allen zugehörigen Daten in den LS der SPEs passen sollen. Selbst wenn unzählige Instanzen einer Klasse erzeugt wurden, sind die Methoden nur einmal referenziert und nehmen daher auch nur einmal Speicherplatz ein.

5.1.4 Einfache oder doppelte Rechengenauigkeit

Die Cell Broadband Engine stellte in ihrer ersten und zweiten Generation (QS20+21) nur einfach-genaue Floating-Point-Berechnungen zur Verfügung. Seit QS22 sind genauso doppelt-genaue Floating-Point-Berechnungen möglich. Nutzt man diese, ist die zu erzielende maximale Rechenleistung aber entsprechend nur halb so groß.
Für die Simulation an sich sind die absoluten Ergebnisse der Gesamtenergie nicht von tragender Bedeutung. Wichtig ist, dass die Energiedifferenz zwischen zwei Schritten korrekt ist. Daher haben wir uns für einfach-genaue Floating-Point-Arithmetik entschieden. Denn es kann so eine höhere Leistung erzielt werden. Die Gefahr bei einfacher Genauigkeit besteht darin, dass aufgrund von Auslöschung das Ergebnis so stark beeinflusst wird, dass die Energiedifferenz zwischen zwei Konformationen nicht richtig ist. Dieses Problem ist im Laufe der Entwicklung im Auge zu behalten.

5.1.5 Daten

Für eine Simulation sind folgende Daten nötig:

- Beschreibung des Kraftfeldes
  Zur Beschreibung der Lennard-Jones-Potentiales dienen die Matrizen A, B, C und D aus der Gleichung (3.1). Sie haben die Form $m \times m$, wobei m die Anzahl der Atome darstellt. Die Matrizen werden auf $n \times n$ Matrizen abgebildet, wobei $n = 18$ ist. Denn für das Potential sind lediglich die Atomtypen der wechselwirkenden Paare von Bedeutung und so kann für große Problemgrößen Speicherplatz gespart werden.
  Für die Beschreibung des Torsions-Potentiales gibt es drei Parameterfelder der Länge n, die für jeden Torsionstypen die Tiefe und das Vorzeichen sowie die Anzahl der Minima des Potential beinhaltet.

- Beschreibung der Atome
  Die Atome werden mit ihrer Bezeichnung, ihren Atomtypen, ihrer Ladung und ihren kartesischen Koordinaten beschrieben. Dabei ist der Typ relevant für die Wechselwirkungen mit anderen Atomen. Über ihn werden die Parameter innerhalb der oben genannten Matrizen indiziert und so die Berechnungen der Lennard-Jones-Potentiale durchgeführt. Die Ladung wird für die Coulomb-Wechselwirkung benötigt.
  In alle drei Terme geht der Abstand der Atompaare ein. Er lässt sich aus den kartesichen Koordinaten, der räumlichen Anordnung der Atome, berechnen. Mit jedem Simulationsschritt, in dem ein zufällig gewählter Torsionswinkel zufällig geändert wird, verschieben sich alle Atome, die sich in der Sequenz hinter diesem Winkel befinden. Zur Berechnung der neuen Konformation müssen alle Koordinaten neu übertragen werden.

- Beschreibung der Torsionswinkel
  Ein Torsionswinkel besteht aus den Angaben seiner Bezeichnung, seinem Typen und seines Wertes. Der Wert muss für die Berechnung des Kosinus im Bogenmaß vorliegen oder dahin umgerechnet werden. Wie gerade schon erwähnt, wird mit jedem Simulationsschritt ein Winkel zufällig geändert. Daher muss auch diese Neuerung vor jeder Berechnung auf die SPE übertragen werden.

- Beschreibung der nahen Nachbarn
  In die Berechnung fließen diejenigen Energien nicht ein, die aus Wechelwirkungen zwischen direkten Nachbarn und nächsten Nachbarn stammen. Wechselwirkungen zwischen Atomen die genau drei Bindungen voneinander entfernt sind, gehen mit dem Faktor $\frac{1}{2}$ im van-der-Waals-Term ein. Die Gründe dafür wurden bereits in 3.2.1 genannt.
  Nun muss dieses Wissen über die Nachbarschaftsverhältnisse der Atome geeignet gespeichert werden. Die Speicherung soll möglichst wenig Speicherplatz benötigen und sich effizient auf den Algorithmus auswirken.
  Die Lösung sind Nachbarschaftslisten, die jeweils Atompaarungen ihrer Ausprägung enthalten. Z. B. enthält eine Liste alle Atomepaare, die 1-4-Nachbarn sind. Diese Listen müssen dann von den SPEs abgearbeitet werden und können einfacher vektorisiert werden.

- Unterscheidung zwischen van-der-Waals- und Wasserstoffbrückenbindungs-Term
  Außer den Nachbarschaftsverhältnissen muss unterschieden werden, ob eine Atompaarung eine Wasserstoffbrücke ausbilden kann oder nicht. Darüber entscheiden die Atomtypen. Je nachdem geht der van-der-Waals- oder Wasserstoffbrückenbindungs-Term in die Bilanz ein.
  Dafür eignet sich ebenfalls das Prinzip der Listen, sodass die Klassen weiter spezifiziert werden, z. B. zu 1-4-Nachbarn die keine Wasserstoffbrücken ausbilden.

5.2 Umsetzung

Zur Umsetzung des Moduls gehört die Erstellung der Datenstrukturen sowie des Algorithmus. Die Entwicklung eines Algorithmus wird in einem Prozess über mehrere Stufen beschrieben. Dieser Prozess wurde maßgeblich vom Laufzeitverhalten, der Skalierung über die SPEs und die Größe der berechenbaren Proteinstrukturen getrieben. In diesem Zusammenhang wurden weitere Erkenntnisse gesammelt, die für spätere Vorüberlegungen in anderen Projekten eine Rolle spielen können.

5.2.1 Datenstrukturen

Bei der Entwicklung der Datenstrukturen spielen der Speicherbedarf, das Alignment und die Verwendung von Vektordatentypen eine wichtige Rolle. Am Beispiel des Torsionswinkel sollen die ersten beide Punkte erklärt werden.
Ein Torsionswinkel besteht aus seiner Bezeichnung, seinem Typ und seinem Wert. Diese werden als char, int und float gespeichert. Ein int- und ein float-Wert nehmen jeweils vier Byte ein, für den Namen benötigt man drei Character.
Die Attribute des Objektes ordnet man am besten vom Größtem zum Kleinsten an. Denn dann wird am wenigsten Padding vom Compiler hinzugefügt, um einen Datentypen auf einer bestimmten Grenze beginnen zu lassen. In diesem Fall steht zuerst der Typ und Winkel, danach die Bezeichnung. Dies sind insgesamt 11 direkt aufeinanderfolgende Byte. Der Compiler würde nun ein Byte Padding hinzufügen, damit der nächste Datentyp auf einer 4-Byte-Grenze anfängt. Um aber bei einem Array von Torsionswinkeln eine fehlerfreie Übertragung für jedes Element einzeln garantieren zu können, müssen mindestens zwei weitere Byte explizit hinzugefügt werden. Der Compiler würde dies durch drei weitere (implizite) Byte auf eine 16-Byte-Grenze ergänzen. Damit benötigt jeder abgespeicherte Winkel 16 Byte.
Für die wichtigste Struktur, das Atom, wird ein Vektordatentyp verwendet. Das Atom besteht aus einem Namen, einem Typ, einer Ladung sowie einer Lage im R^3. Die zugehörigen Daten sind char, int und float. Die wichtigste Methode, die für ein Atom zu definieren ist, ist die Abstandsberechnung zwischen zwei Atomen. Da diese Methode für jede Energieberechnung einmal aufgerufen wird, ist die Laufzeit dieser Methode maßgeblich für die Gesamtlaufzeit. Zu berechnen ist $r^2 = (x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2$. Dies kann größtenteils mit SIMD-Funktionen (folgt in Kapitel 5.2.2) berechnet werden. Deshalb werden die drei Koordinaten in einem float-Vektor abgespeichert. Um den vierten Eintrag nicht frei zu lassen, wird die Ladung dort abgelegt. Darauf folgt der Typ und Name. Der float-Vektor ist 16 Byte lang. Damit ergeben sich insgesamt 23 Byte. Diese müssen für ein 16 Byte Alignment bis auf 32 Byte aufgefüllt werden.

5.2.1.1 Übertragung

Es ist für die Ausführung besser, wenn die Übertragung von den SPEs selbst eingeleitet wird. Dann können die SPEs gleichzeitig den DMA-Transfer für sich einleiten, während das PPE höchstens in seinen zwei Threads diese Aufgabe übernehmen könnte. Außerdem

ist die Latenzzeit für den SPE-seitigen DMA-Transfer kürzer als für den PPE-seitigen. Aber woher wissen die SPEs von welchen Hauptspeicheradressen sie sich wieviele Daten holen sollen? Dafür gibt es eine Kontrollstruktur, die diese Informationen bereit hält. Handelt es sich um einzelne Daten kann dies auch über Mailbox-Kommunikation erfolgen. Allerdings benötigt eine 64-Bit-Adresse schon zwei solcher Empfangsbefehle. Daher ist die Kontrollstruktur die erste Wahl. Die Hauptspeicheradresse dieser Struktur ist im Context eines jeden SPE gespeichert. Die Adresse des Contextes ist der Übergabeparameter an die SPE-Hauptfunktion.
Zur Initialisierung der Berechnung werden die Matrizen A, B, C und D zur Beschreibung des Kraftfeldes benötigt. Die Listen definieren für welche Atompaare welche Wechselwirkungen berechnet werden müssen. Diese müssen zu Beginn einmalig auf die SPEs gebracht werden. Für jede einzelne Konformation müssen die Daten der Atome und Torsionswinkel geändert werden. Das bedeutet, dass dieser Hauptspeicherbereich immer neu auf die SPEs übertragen werden muss.

5.2.2 Vektorisierung

Die Vektorisierung des Algorithmus ist die schwierigste Aufgabe. Nicht jede Problemstellung lässt eine Vektorisierung zu und mehrere Nebenfaktoren beeinträchtigen die Leistung. Um annähernd einen Speedup von vier gegenüber dem nicht vektorisierten Programm zu erreichen, müssen der Algorithmus und die Datenstrukturen auf einen schnellen Programmablauf ausgelegt sein. Eine Möglichkeit ist von Fall zu Fall neu zu entwickeln und am besten auch gegenüber anderen Varianten zu testen. Denn man denkt selten an alle Faktoren, die die Zeit herabsetzen können.
Am einfachsten ist die Umsetzung für Datenstrukturen, die vektoriell dargestellt werden können. Ein gutes Beispiel ist die Abstandsberechnung zweier Atome, die in der Klasse Atom bereitgestellt werden soll. Dafür muss $r^2 = (x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2$ berechnet werden. Schreibt man die Koordinaten eines Atoms als Vektor, so ist ein Teil der Berechnung durch eine Differenz und eine Potenz darzustellen.
Dies sieht dann wie folgt aus:

$$(\overrightarrow{v_1} - \overrightarrow{v_2})^2 = \begin{pmatrix} x_1 - x_2 \\ y_1 - y_2 \\ z_1 - z_2 \end{pmatrix}^2$$

Danach müssen lediglich die einzelnen Komponenten des Vektors aufaddiert werden. Damit werden acht Operationen durch zwei SIMD- und zwei Einzeloperationen ersetzt. Für Rechnungen, in denen mehrere gleiche Operationen verwendet werden und deren Operationsreihenfolge beliebig ist, könnten die Operationen so zusammengefasst werden, dass mehrere auf einmal berechnet werden. Dies ist jedoch eher selten.
Öfter treten Schleifen auf, deren Berechnungen unabhängig voneinander sind. Diese können gut vektorisiert werden, indem man die Daten für vier bzw. zwei (bei einfach bzw. doppelter Genauigkeit) Iterationen in einem Vektor speichert und alle vier bzw. zwei Iterationen eine SIMD-Funktion darauf ausführt.

Daraus ergibt sich nachfolgende Struktur:

```
vec_float a, b;
 for(i=0;i<n;i+=4){
   for(j=0;j<4;j++){
      a.s[j]=...;
      b.s[j]=...;
   }
   spe_func(a.v,b.v);
 }
```

Wichtig ist hierbei das Inkrement um vier in der äußeren Schleife. Die dabei „übersprungenen“ Iterationen werden in der inneren Schleife vorbereitet und in einem Schritt in der SIMD-Funktion verarbeitet.

5.2.3 Algorithmus

Das zu implementierende Problem lässt sich trotz allen Einschränkungen, die im Vorhinein gemacht wurden, noch auf verschiedene Arten bearbeiten. Wie der Entwicklungsweg zur erstmal endgültigen Fassung von statten ging soll nun erklärt werden.

5.2.3.1 Ausgangspunkt

Noch einmal die zu implementierende Formel:

$$E = \sum_{(i,j)} 332\frac{q_i q_j}{\epsilon r_{ij}} + \sum_{(i,j)}(\frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}}) + \sum_{(i,j)}(\frac{C_{ij}}{r_{ij}^{12}} - \frac{D_{ij}}{r_{ij}^{10}}) + \sum_{l} U_l(1 \pm cos(\eta_l \xi_l)) \qquad (5.1)$$

Darin gibt es drei Doppelsummen, die die Wechselwirkungen Coloumb, van-der-Waals und Wasserstoffbrückenbindungen von jedem Atom zu jedem Atom beschreiben. Die letzte Summe läuft einfach über alle Torsionswinkel. Die Wechselwirkungen naher Nachbarn (bis zwei Bindungen entfernt) werden nicht berechnet. Der Wasserstoffbrückenbindungsterm geht in die Bilanz ein, wenn eine solche Brücke zwischen den zwei Atomen ausgebildet werden kann (s.3.1.2). Ansonsten geht der van-der-Waals-Term ein, der bei 1-4-Nachbarn mit dem Faktor $\frac{1}{2}$ im abstoßenden Teil des Lennard-Jones-Potentials zu berücksichtigen ist.
Der Ablauf eines sequentiellen Programmes sieht demnach folgendermaßen aus:

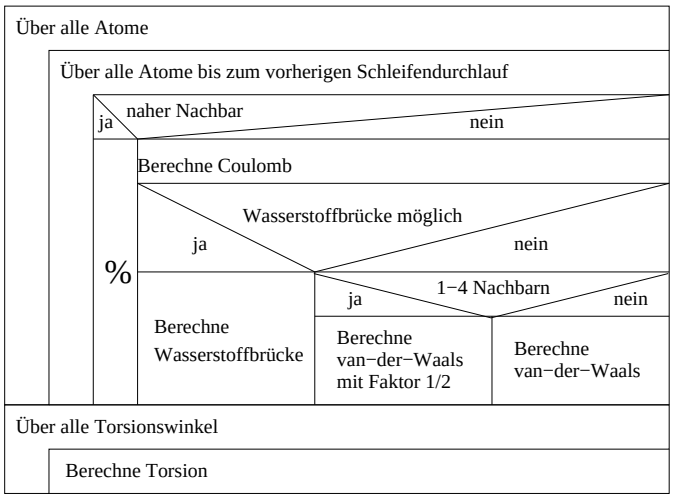

Abbildung 5.1: Nassi-Shneiderman-Diagramm für einen sequentiellen Programmablauf. Dieser Ablauf ist die Folgerung aus der oben beschriebenen Formel und den genannten Einschränkungen, wie man ihn auf einer herkömmlichen Architektur implementieren würde. Es werden alle Atompaare durchlaufen und anhand der Kriterien der entsprechende Summand berechnet.

Insgesamt lassen sich damit folgende Aussagen über die Problemstellung machen:

- die Komplexität des Coulomb-, van-der-Waals- oder Wasserstoffbrückenterms beträgt $O(n^2)$
- die Komplexität des Torsionterms beträgt $O(n)$
- innerhalb der Berechnung sind drei Entscheidungen zu treffen:
  - sind die Atome eines Atompaares direkte oder nächste Nachbarn, brauchen die zugehörigen Wechselwirkungen nicht berechnet zu werden
  - kann zwischen den Atomen eines Atompaares eine Wasserstoffbrücke ausgebildet werden, so geht der Wasserstoffbrückenterm in die Bilanz ein, ansonsten der van-der-Waals-Term
  - sind die Atome eines Atompaares 1-4-Nachbarn, so geht der abstoßenden Teil des van-der-Waals-Terms nur mit dem Faktor $\frac{1}{2}$ ein

Insgesamt ergeben sich dabei drei unabhängige Klassen von Atompaaren, die in der Berechnung gesondert betrachtet werden müssen: a) die weit entfernten und b) 1-4 Nachbarn, die keine Wasserstoffbrücke ausbilden sowie c) alle, die sie ausbilden können.
Dabei gibt es weniger 1-4-Nachbarn als weit-entfernte Nachbarn und auch die Anzahl der Atompaare die Wasserstoffbrücken ausbilden sind wesentlich geringer als die, die es können. Dies wird für den weiteren Verlauf der Implementierung relevant werden.

5.2.3.2 Abarbeiten von Listen

Das Konzept der Listen ist simpel. Die Listen halten die Informationen über die Atompaare bereit, die einer Klasse (z. B. 1-4-Nachbarn und keine Wasserstoffbrückenbindung) angehören. Eine solche Liste kann, von vorn nach hinten in Vierer-Gruppen abgearbeitet, vektorisiert werden. Dabei gibt es keine Sonderfälle, die die Befehlsfolge unterbrechen können. Lediglich am Ende muss bei der Berechnung der Summe darauf geachtet werden, dass der Vektor nicht voll besetzt ist, wenn die Liste keine durch vier teilbare Anzahl an Elementen enthält.
Der Ablauf für diese Problemstellung sieht dann wie folgt aus:

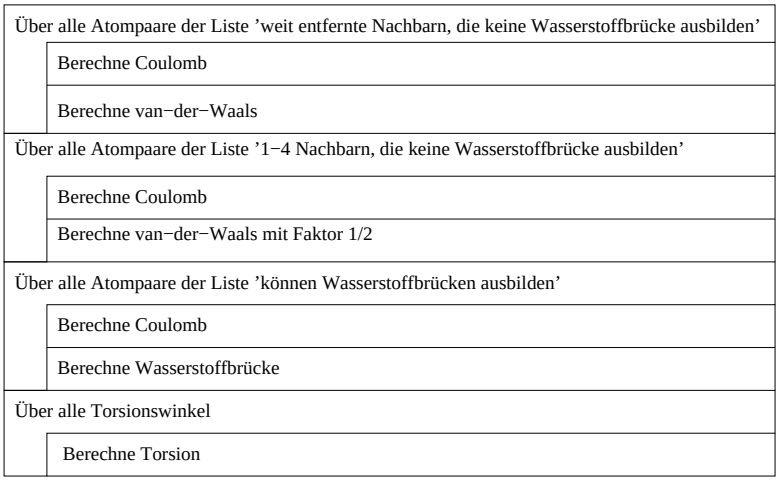

Abbildung 5.2: Nassi-Shneiderman-Diagramm für Listen-Programm. Mit den in 5.2.1 beschriebenen Listen ist der Programmablauf stark auf die Vektorisierung ausgelegt. Der Ablauf enthält im Vergleich zum sequentiellen Ablauf keinerlei Verzweigungen. Jedoch deuten die einfachen Schleifen im Gegensatz zu der doppelten Schleife im sequentiellen Ablauf das Problem der großen Anzahl an Listeneinträgen an.

Das erste Testsystem besteht aus 33 Atomen. Daraus ergeben sich $\frac{33*(33-1)}{2} = 528$ zu berechnende Atompaare. Davon gehen für das hier simulierte Peptid 87 nahe Nachbar-Atompaare ab, sodass noch 441 übrig bleiben. Außerdem gibt es 15 Torsionswinkel. Für eine Iteration der Berechnung einer solch kleinen Konformation mit knapp 900 Summanden, benötigt man rund $4,85 * 10^{-5}$ Sekunden auf einem SPE. Inklusive der Übertragung der Atom- und Torsionsdatensätze sind es $5,57 * 10^{-5}$ Sekunden.
Wie ich in meiner Seminararbeit über die Vektorisierung und Parallelisierung einer Wellengleichung[26] zeigen konnte, kann man auf dem SPE mit der Vektorisierung einen Speedup von vier erzielen. Ein entsprechender Vergleichswert für diese Problemstellung auf den SPEs existiert nicht. Einziger Vergleichswert sind $9,45 * 10^{-5}$ Sekunden für das Originalprogramm auf dem PPE. Dies ist nur geringfügig langsamer als die vektorisierte SPE-Version. Jedoch ist die SPE-Berechnung ohne Vektorisierung langsamer als eine Berechnung auf dem PPE. Somit lässt dieser Vergleichswert auf einen ähnlichen Speedup für die Vektorisierung schließen.
Eine Parallelisierung über die SPEs ist bei diesem Modell einfach vorzunehmen, indem

jedes SPE einen Teil der Listen erhält. Die Ergebnisse müssen am Ende nur einzeln summiert werden. Aufgrund der guten Lastbalancierung und keinerlei Kommunikationsbedarf der SPEs untereinander sollte das Konzept über alle 16 SPEs skalieren.
Doch so effektiv wie die Listen die Berechnung auch machen, sie haben einen sehr hohen Speicherbedarf. Die vier Listen teilen sich alle zu berechnenden $\frac{n(n-1)}{2}$ Atompaare abzüglich der nahen Nachbarn. Dabei steigt die Anzahl der 1-4-Nachbarn und der Paare die Wasserstoffbrücken ausbilden linear an. Die Anzahl der weit-entfernten Nachbarn ohne Wasserstoffbrücken steigt dagegen quadratisch.

5.2.3.3 Abarbeiten von inversen Listen

Das Konzept der Listen hat sich generell als effizient erwiesen. Doch ist der Speicherbedarf zu optimieren. Das größte Problem ist es, die große quadratisch ansteigende Liste der weit-entfernten Nachbarn ohne Wasserstoffbrückenbindungen zu speichern. Aber die drei Klassen, die in der Berechnung gesondert betrachtet werden müssen, können auch über die jeweils kürzesten Listen gebildet werden.
Das sind:

- nahe Nachbarn
- 1-4 Nachbarn
- Wasserstoffbrücken

Der daraus sich ergebende Ablauf ähnelt sehr dem sequentiellen Programm:

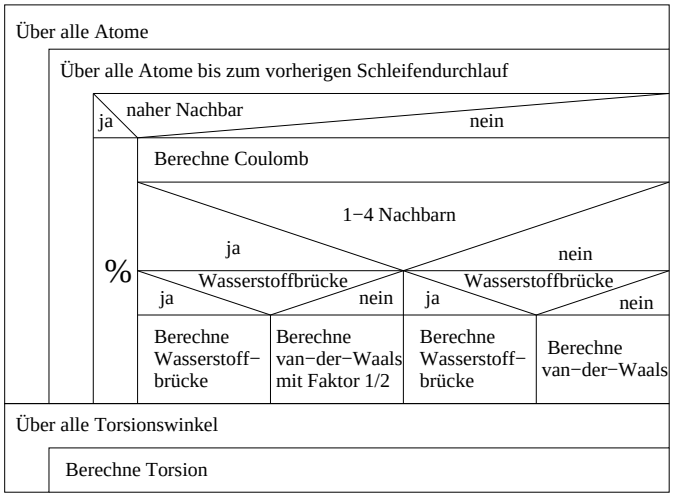

Abbildung 5.3: Nassi-Shneiderman-Diagramm für Programm mit inversen Listen. Um die Eingrenzung der Klassen vorzunehmen, muss man ein Ausschlussverfahren innerhalb der Listen vornehmen. Der Ablauf des Programmes ähnelt damit sehr dem sequentiellen Programm. Im Vergleich zur ersten Version hat man wieder Verzweigungen. Man hat aber auch wieder eine doppelte Schleife. Sie deutet an, dass der Speicheraufwand für die Listen gesunken ist.

Das Problem ist, dass innerhalb dieses Ablaufes eine Vektorisierung nur schwer vorzunehmen ist. Bei dem Coulomb- und Torsionsterm gibt es kein Problem immer vier aufeinanderfolgende Rechnungen zu einer SIMD-Rechnung zusammenzufassen. Doch ist dies bei den Wasserstoffbrücken- und den zwei unterschiedlichen van-der-Waals-Termen durch ihr unregelmäßiges Auftreten nicht der Fall. Dadurch kann nicht nach vier Atompaaren eine SIMD-Berechnung durchgeführt werden, da diese Paare nicht zu einer Klasse gehören. Es muss immer abgefragt werden, ob vier Datensätze zu einer Klasse gesammelt worden sind, damit nun eine SIMD-Operation ausgeführt werden kann. Doch würde dies eine weiter Abfrage innerhalb eines jeden Zweiges bedeuten, sodass vier Entscheidungen getroffen werden müssen.
Und auch wenn vier Verzweigungen nach nicht viel klingt, so beweisen die Ergebnisse, dass dies Zeit kostet. Das gleiche System von 33 Atomen benötigt für die Berechnung einer Iteration $5,23 * 10^{-4}$ Sekunden und mit Übertragung $5,27 * 10^{-4}$ Sekunden. Damit ist diese Implementierung um den Faktor 10 langsamer geworden. Dafür wird zur Speicherung der Listen weniger Platz im LS verbraucht, sodass größere Proteine simuliert werden könnten.
Der Grund für die Einbußen im Laufzeitverhalten liegt darin, dass eine falsche Vorhersage des zu durchlaufenden Zweiges 18-19 Taktzyklen dauert[27]. Denn es müssen die Befehle des anderen Zweiges ins Befehlsregister geladen werden. Dies macht den Gewinn durch die SIMD-Funktionen pro Durchlauf wieder zunichte.

5.2.3.4 Berechne zu viel und zieh es wieder ab

Die SIMD-Funktionen bringen also nur eine große Leistung hervor, wenn die Befehlssequenz nicht durch Verzweigungen unterbrochen wird. Doch wie vermeidet man die Verzweigungen und umgeht gleichzeitig große Listen? Was nimmt man dafür in Kauf?

Die Rechenleistung zum Erzielen von entsprechender Performance ist vorhanden, wie man an der ersten Implementierung gesehen hat. Zu vermeiden sind lediglich Verzweigungen, sodass das Programm eine Berechnung nach der Anderen ausführt. Das nachfolgende Konzept setzt darauf auf. Zusätzlich macht es sich die Tatsache zu Nutze, dass für die meisten Atompaare die Coulomb- und van-der-Waals-Wechselwirkung in die Bilanz eingehen.
Die Coulomb- und van-der-Waals-Wechselwirkungen werden zunächst für alle möglichen Atompaare berechnet, egal in was für einer Klasse sie sind. Danach werden die Listen durchlaufen. Die Coulomb- und van-der-Waals-Wechselwirkungen der nahen Nachbarn, die gar nicht in die Bilanz eingehen, werden erneut berechnet und wieder abgezogen. Gleiches gilt für alle Atompaare die eigentlich Wasserstoffstoffbrücken bilden: die van-der-Waals-Wechselwirkung wird abgezogen, stattdessen der Wasserstoffbrückenterm dazuaddiert. Für 1-4-Nachbarn, die keine Wasserstoffbrücken ausbilden, muss $\frac{1}{2}$ mal der abstoßende Teil des van-der-Waals-Terms abgezogen werden, um ihn zu korrigieren.

5.2.3.5 Problem bei der Implementierung

Während des Testens ergaben sich falsche Ergebnisse. Das Problem ist die Rechengenauigkeit. Wie bereits in 5.1.4 erklärt, hatten wir uns für die einfachgenaue Floating-Point-Rechnung entschieden um eine höhere Performance zu erzielen. Jedoch nimmt man dafür in Kauf, dass die Zahlendarstellung mit weniger Stellen in der Mantisse auskommt. Dies ist gerade dann ein Problem, wenn man mit sehr großen und kleinen Zahlen rechnet. Es kommt zur Auslöschung, wenn auf eine sehr große Zahl eine sehr Kleine addiert wird.
Für die Addition müssen beide Zahlen so verschoben werden, dass sie den gleichen Exponenten haben. Liegen die beiden Zahlen weit genug auseinander verändert sich die Größere der beiden Zahlen nicht, da die relevanten Ziffern der kleinen Zahl nicht mehr im darstellbaren Bereich liegen.
Allerdings kommen gerade in der Coulomb- und van-der-Waals-Wechselwirkung große Summanden vor, wenn die Atome nahe Nachbarn sind. Diese Summanden werden zwar später wieder von der Summe abgezogen, doch kann es zwischenzeitlich zu Auslöschung kommen. Um dem Fehler zu beheben wurde zuerst die Aufsummierung, die sowieso nicht mit SIMD-Funktionen realisiert ist, in doppelt-genauer Floating-Arithmetik vorgenommen.
Doch half die doppelte Genauigkeit nur bedingt. Für Konformationen mit sehr eng beieinanderliegenden nahen Nachbarn nahmen die Summanden so große Werte an, sodass dann die Summe aus dem darstellbaren Bereich herausreichte.
Um den Fehler entsprechend abzufangen kommt man nicht umher, eine Verzweigung einzubauen, die erkennt wann es sich um einen nahen Nachbarn handelt. Allerdings wird diese Atompaarung für die SIMD-Rechnung nicht übersprungen. Denn dies würde wieder bedeuten, dass man nicht regelmäßig SIMD-Berechnungen durchführen kann.Stattdessen wird der Betrag korrekt berechnet, aber beim Aufsummieren nicht dazuaddiert. Man spart also nicht die Berechnung selbst, sondern nur das nachträgliche Abziehen. Aber man bleibt SIMD-konform und muss daher nur eine Verzweigung einbauen.
Durch diese Verzweigung entstand 50% Verlust, weswegen sich noch die Frage stellte, ob man diesen nicht minimieren kann.
Man weiß, dass eine Fehleinschätzung 18-19 Taktzyklen kostet. Mit einer Vorhersage, welcher Zweig öfter angenommen wird kann man die Anzahl der Fehleinschätzung auf die Anzahl der nahen Nachbarn reduzieren. Eine Vorhersage, dass der erste Zweig öfter durchlaufen wird als der Zweite sieht so aus:

```
if(__builtin_expect((naher Nachbar),0))
  ueberspringe();
else
  addiere();
```

Jedoch kostet eine Vorhersage immer sechs Taktzyklen, sodass man dadurch nur gewinnen kann, wenn sonst quasi immer eine Fehleinschätzung gemacht wurde. In unserem Fall kostet die Vorhersage mehr als durch das Vermeiden der Fehleinschätzung gewonnen werden konnten.

Zur Berechnung müssen wie in der vorherigen Methode nur die linear ansteigenden Listen gespeichert werden. Die erste „Überschlagsrechnung“ für alle Atompaare kann ohne Unterbrechung gerechnet werden und damit die volle SIMD-Leistung ausschöpfen. Gleiches gilt für die Korrekturrechnung. Auch dabei können die Listen wieder unterbrechungsfrei abgearbeitet werden.
Der Ablauf ergibt sich wie im folgenden Struktogramm dargestellt:

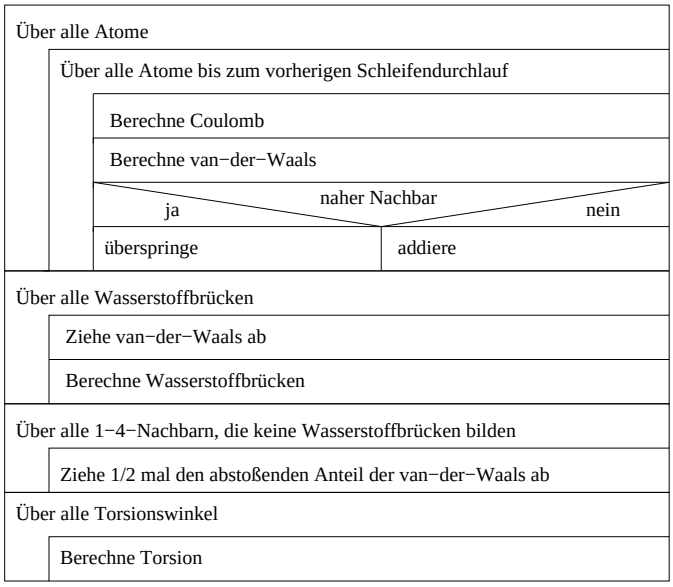

Abbildung 5.4: Nassi-Shneiderman-Diagramm für Programm, das Korrektur rechnet. Dieser Programmablauf verbindet die Vorzüge der beiden ersten Versionen: kurze Listen und keine Verzweigungen, dadurch hohe Leistung. Dabei berechnet es Wechselwirkungen, die nicht in die Bilanz eingehen, und zieht sie später wieder ab.

Obwohl das Programm mehr Rechnungen als nötig durchgeführt hat und diese teilweise wiederholt, um das zu viel berechnete wieder abzuziehen, und Ersatzterme berechnet, erzielt man mit dieser Implementierung eine gute Rechenleistung.
Das Startsystem von 33 Atomen benötigt nun mit $7,37 * 10^{-5}$ Sekunden nur noch ca. $\frac{1}{7}$ der Zeit der zweiten Version. Gegenüber der ersten Implementierung hat man Speicherplatz im LS gewonnen. Somit können größere Proteine simuliert werden. Für die Beispiel-Datensätze waren 503 Atome die Größte Anzahl, die auf einer einzelnen SPE durchgeführt werden konnten.
Der Algorithmus verhält sich entsprechend $O(n^2)$: bei einer Verdoppelung des Problems vervierfacht sich die Rechenzeit.

i	Problemgröße n_i	Rechenzeit t_i pro Iteration	$\left(\frac{n_i}{n_{i-1}}\right)^2$	$\frac{t_i}{t_{i-1}}$
1	33	$7,37 * 10^{-5}$	-	-
2	123	$9,52 * 10^{-4}$	$\left(\frac{123}{33}\right)^2 = 13,7$	$\frac{9,52*10^{-4}}{7,37*10^{-5}} = 12,92$
3	503	$1,58 * 10^{-2}$	$\left(\frac{503}{123}\right)^2 = 16,7$	$\frac{1,58*10^{-2}}{9,52*10^{-4}} = 16,60$

5.2.3.6 Parallelisierung über die SPEs

Der letzte Schritt der Implementierung ist die Aufteilung über die SPEs. Dabei sollen möglichst viele Daten und Aufgaben verteilt werden, um den LS effizient auszunutzen und dabei eine gute Lastbalancierung zu erzielen.
Die „Überschlagsrechnung“ kann man sich in Form einer Dreiecksmatrix vorstellen. Dabei ist jeder Eintrag eine Atompaarung. Eine einfache aber effiziente Parallelisierung erreicht man indem man die Zeilen dieser Matrix fortlaufend über die SPEs verteilt. Das heißt, dass bei 4 SPEs SPE 1 die Zeile 1, 5, 9, SPE 2 Zeile 2, 6, 10 erhält.

		a	b	c	d	e
SPE 1	a					
SPE 2	b	ba				
SPE 3	c	ca	cb			
SPE 4	d	da	db	dc		
SPE 1	e	ea	eb	ec	ed	

Abbildung 5.5: Modell der „Überschlagsrechnung“ für vier SPEs. Um jede Wechselwirkungen zwischen jedem Atom mit jedem anderen Atom abzubilden, arbeitet man die Einträge dieser Wechselwirkungsmatrix ab. Dabei muss lediglich das untere Dreieck durchlaufen werden, da im oberen die gleichen Wechselwirkungen in die entgegengesetzt Richtung eingetragen sind und auf der Diagonalen die Wechselwirkungen von jedem Atom zu sich selbst. Diese müssen aber nicht berechnet werden.

Dabei hat das erste SPE drei Atompaare weniger zu berechnen als das vierte SPE. Da maximal über 16 SPEs verteilt wird, ist der Unterschied maximal 15. Dafür müssen für diese Aufteilung der Berechnung auf jedem SPE alle Atomdaten vorhanden sein.
Bei der Korrekturrechnung über die Listen kann eine blockweise Aufteilung der Listen vorgenommen werden, sodass alle SPEs eine gleich lange Liste abzuarbeiten haben. Dazu muss auf jedem SPE auch nur der Block der Liste bekannt sein, der zu berechnen ist. Die Liste der nahen Nachbarn hingegen ist relevant für die „Überschlagsrechnung“, sodass diese nicht aufgeteilt werden darf.
Das Laufzeitverhalten über die SPEs wurde für verschiedene Proteine mit 33, 123, 153, 503 und 1153 Atomen für die reine Energieberechnung und mit Übertragung der Daten auf die SPEs gemessen.

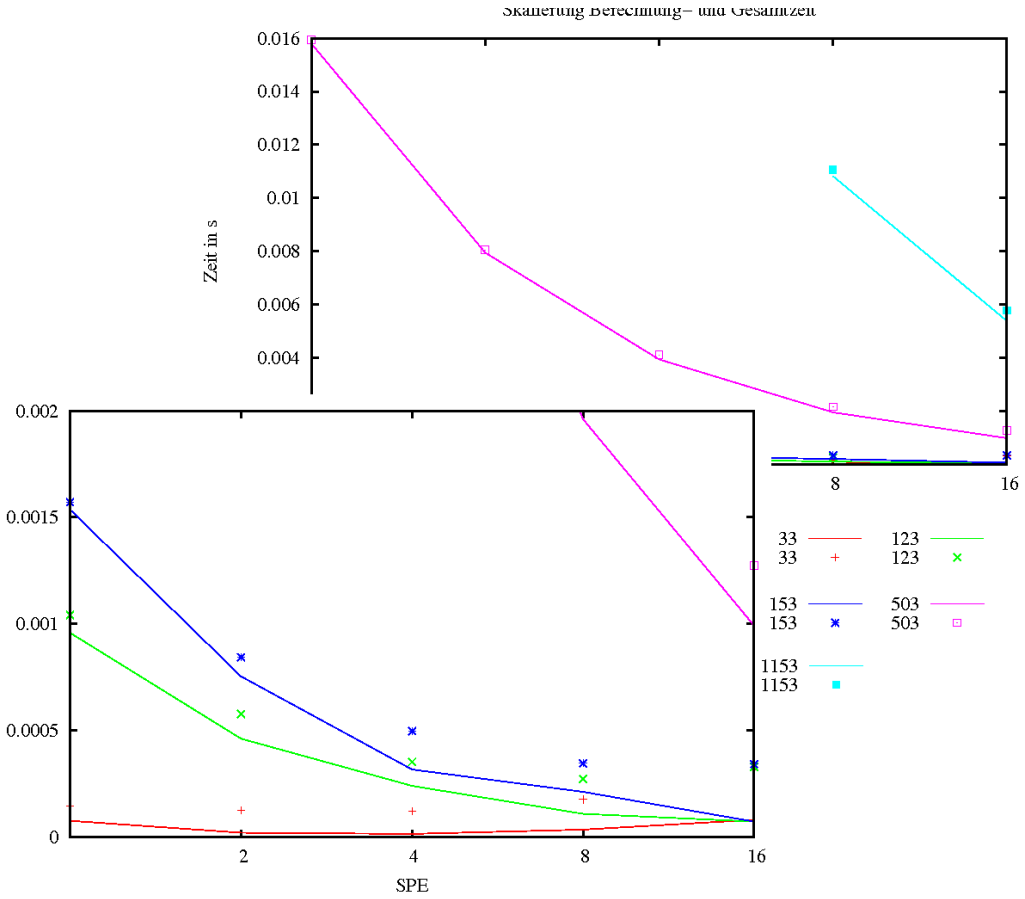

Abbildung 5.6: Laufzeitverhalten über 1-16 SPEs. Dieses Diagramm zeigt das Laufzeitverhalten bei der Aufteilung über alle SPEs. Dafür wurden Messungen der reinen Energieberechnung (Linien) und der Energieberechnung mit Übertragung (Punkte) von fünf Proteinen mit 33, 123, 153, 503 und 1153 Atomen berechnet. Die größte Problemgröße konnte nur auf acht und 16 SPEs berechnet werden, da für weniger SPEs der LS nicht ausreicht. Einschränkendes Problem ist die Liste der nahen Nachbarn, die nicht aufgeteilt werden konnte, sondern auf jedem SPE komplett vorliegt. Die kleineren Problemgrößen sind im Ausschnitt unten links dargestellt. An ihnen erkennt man den Unterschied zwischen der Berechnung mit und ohne Übertragung.

Bei entsprechender Problemgröße skaliert das Programm über alle SPEs. Bestes Indiz ist der Speedup. Dieser gibt Aufschluß über das Skalierungsverhalten. Ideal ist linearer Speedup. Das heißt, dass sich bei einer Verdopplung der SPEs die Zeit für die Berechnung halbiert. Dies ist aber nur der Idealfall, der bei idealer Lastbalancierung und keiner Kommunikation eintritt. Denn dann gibt es keinen Overhead durch Warten auf andere Prozessoren.

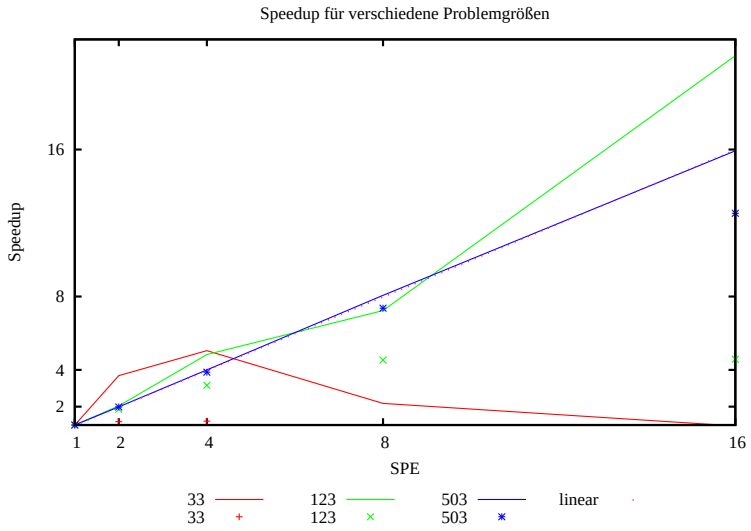

Abbildung 5.7: Speedup über 1-16 SPEs für 33, 123 und 503 Atome. Für 1153 kann der Speedup nicht bestimmt werden, da kein Wert für ein SPE vorliegt. Der lineare Speedup ist gepunktet dargestellt. Dem gegenüber werden die verschiedenen Werte für die entsprechende Atomanzahlen ohne (Linien) und mit Übertragung (Punkte) gestellt.

Man sieht deutlich, dass dabei die Problemgröße eine wichtige Rolle spielt. Für das kleinste Problem mit 33 Atomen skaliert es nur für vier SPEs. Danach verliert man Rechenleistung, durch Fehleinschätzung von Schleifen auf mehreren SPEs.
An den Zeiten mit Übertragung sieht man, dass die Latenzzeit des DMA-Transfers für kleine Problemgrößen das Skalierungsverhalten deutlich einschränkt. Für größere Proteine spielt das aber mit zunehmender Größe keine tragende Rolle mehr, da die Berechnungzeit deutlich höher als die Übertragungszeit ist.

5.3 Zusammenfassung

Betrachtet man zum Abschluss des Moduls den Entwicklungsweg, so ist es kein einstufiger Prozess gewesen. Vielmehr bestand die Entwicklung aus einer Iteration von Problemanalyse, Vorgehenskonzeption, Umsetzung und Testen. Der Grund dafür liegt darin, dass nicht alle auf die Ausführzeit einwirkenden Faktoren auch gleichzeitig berücksichtigt wurden. Außerdem muss man sie gut gegeneinander abwägen können. Die Optimierung des Codes liegt (bei der nativen Programmierung) beim Programmierer.
Die wesentliche Aussage für die SPE-Programmierung ist es gradlinig verlaufenden, vektorisierten Code zu schaffen. Die Ergebnisse dieses Moduls zeigen, dass Verzweigungen innerhalb einer Schleife rund 50% SIMD-Leistung kosten. Sie zeigen, dass nicht nur ein Wissen über die Rechenvorschriften, sondern auch über die Daten wichtig ist. Ohne Kenntnis über die Häufigkeitsverteilung der verschiedenen Klassen wäre man nicht auf den Ansatz der letzten Implementierung gekommen.
Auf der Cell Broadband Engine ist es nicht mehr so, dass das Einsparen von Rechen-

schritten auch weniger Rechenzeit bedeutet. Denn es kostet nicht nur die Berechnung Taktzyklen, sondern eben auch eine Verzweigung (4 Zyklen) und die Fehleinschätzung des zu durchlaufenden Pfades (18-19 Zyklen). Im Vergleich kostet eine einfach-genaue SIMD-Funktion 6 Taktzyklen. Das heißt, dass anstelle einer falsch eingeschätzten Verzweigung (22-23 Takte) 16-17 SIMD-Funktionen (einmal 6 Zyklen zum Füllen der Pipeline, danach jeden Takt ein Ergebnis) durchgeführt werden können. Das sind im Normalfall 64-68 Floating-Point-Operationen, wenn ein Multiply-Add oder -Subtract angewendet werden kann sogar 128-136 Floating-Point-Operationen.
Der Speicherplatz auf den SPEs ist der zweite wichtige Faktor. Der LS muss das Programm, sowie alle Daten enthalten. Die Speicherverwaltung übernimmt der Programmierer. Der LS stellt dabei den Cache der SPEs dar, in den die Daten aus dem HS mittels DMA-Transfer geholt werden. Die Übertragungszeit kann dabei für ein Programm ein beschränkender Faktor sein. Um dem entgegen zu wirken gibt es auf der Cell Broadband Engine das Prinzip des double bufferings. Denn ein SPE kann rechnen, während Daten in den LS kopiert werden. Zwischen zwei Iterationen wird nur der Puffer des Datensatzes getauscht, der berechnet werden soll und der neu geholt werden muss. Damit kann der Anteil der Übertragungszeit an der Gesamtzeit auf ein Minimum reduziert werden und der Rest bleibt hinter der Ausführung versteckt (s. 7.2).
Das Prinzip des double bufferings steht noch als Optimierung des Moduls aus. Dadurch kann nicht nur die Gesamtzeit weiter reduziert werden, sondern auch die Größe des Proteins beliebig vergrößert werden. Denn es müssen nicht alle Daten eines Proteins gemeinsam in den LS, sondern können nachgeladen werden. Dafür muss aber außer dem Tausch der Puffer zwischen den Recheneinheiten für das double buffering auch der Algorithmus und die Organisation der Listenstrukturen geändert werden, um eine korrekte Parallelisierung vornehmen zu können. Es sind dann immer nur Teile der Daten der Atome im LS, was eine zeilenweise Aufteilung der „Überschlagsrechnung“ nicht möglich macht.

Die letzte Version erzielt erfreulicherweise für das Protein mit 33 Atomen wieder Zeiten in der Größenordnung 10^{-5} der ersten Version. Außerdem skaliert diese Version über alle 16 SPEs bei einer entsprechenden Problemgröße. Somit kann man den Algorithmus für die Cell Broadband Engine als effizient einstufen. Wie er im Vergleich zum Original-Algorithmus auf der PPE und anderen Architekturen abschneidet wird in Kapitel 6 behandelt.

6 Vergleich zu anderen Architekturen

Die Rechnerarchitektur beschreibt den internen Aufbau der einzelnen Komponenten und deren Zusammenspiel. Zu den grundlegenden Komponenten eines Rechners gehören die CPU, der Speicher und das Bussystem. Doch nicht nur Größen wie Taktgeschwindigkeit oder Übertragungsbandbreite sind entscheidend. Auch designspezifische Fragen wie z. B. die Art des Befehlssatzes, die Einstufung nach der Flynnschen Rechnerklassifikation oder die Fähigkeit zu Multi-Threading sind ausschlaggebend für die Leistung eines Computers. Dabei wirkt sich manche Eigenschaft positiv auf eine Art von Anwendungen aus, andere Eigenschaften wieder auf ganz andere Anwendungen.
Wie die Implementierung der Energieberechnung gezeigt hat, musste der Algorithmus für die Cell Broadband Engine speziell angepasst werden, um ihre Vorzüge auszunutzen und damit mehr Leistung zu erzielen. Zum Vergleich steht das originale Fortran-Programm aus SMMP. Der Programmcode ist auf keine Architektur hin optimiert. Eine solche Optimierung kann nur eventuell von Seiten des Compilers vorgenommen werden.
Nachfolgend werden die Vergleichs-Architekturen vorgestellt und diskutiert wie die Cell Broadband Engine im Vergleich zu den anderen Architekturen abschneidet.

6.1 Die Vergleichs-Architekturen

Zum Vergleich wurden folgende Systeme des Jülich Supercomputing Centre herangezogen:

JuiceNext - Juelich Initiative Cell Cluster next generation Der JuiceNext ist ein Cluster aus Cell-Prozessoren. Es besteht aus 2,5 BladeCenter H, mit insgesamt 35 QS22 Blades. Jedes Blade verfügt über zwei PowerXCell 8i Prozessoren mit jeweils 8 SPEs. Über die zwei I/O Interfaces sind die Prozessoren miteinander verbunden, sodass auf einem Blade eine Parallelisierung über bis zu 16 SPEs möglich ist. Die einzelnen Blades sind über Infiniband 4x verbunden.

Jump - Juelich MultiProcessor Der aktuelle Jump basiert seit dem Upgrade im Mai 2008 auf einem IBM POWER6-System. Er setzt sich aus 14 symmetrischen Multiprozessorsystem-Knoten (SMP) zusammen. Jeder Knoten besitzt 32 simultan multithreadfähige (SMT) Prozessoren. Die MPI-Kommunikation verläuft ebenfalls über Infiniband.

Jugene - Juelicher BlueGene/P Jugene ist Deutschlands und Europas schnellster Supercomputer. Er belegt derzeit Platz drei der aktuellen Top500-Liste (Stand: Juni 2009). Seine Architektur ist IBMs BlueGene/P. 72 Racks mit jeweils 32 Nodecards, die aus 32 Knoten mit vierfach SMT-Prozessor bestehen, ergeben 294.912 Prozessoren. Das Netzwerk der Knoten ist ein dreidimensionaler Torus.

Juropa - Juelich Research on Petaflop Architectures Juropa ist der neueste Supercomputer in Jülich. Er ist aufgeteilt in zwei Teile: Juropa-JSC und HPC-FF (High Performance Computing For Fusion). Zusammen verfügt er über 3288 Knoten. Jeder Knoten besteht aus zwei Intel Xeon X5570 (Nehalem-EP) Quad-Core Prozessoren. Insgesamt sind das über 26304 Prozessoren. Der JSC-Teil besteht dabei aus dem Sun Blade System 6048, der HPC-Teil aus Bull NovaScale R422-E2 Technologie. Momentan ist er in der Top500 auf Platz 10 (Stand: Juni 2009).

6.1.1 Kenngrößen

Die wichtigsten Kenngrößen der Systeme sind in der nachfolgenden Tabelle aufgeführt:

	JuiceNext	Jump	Jugene	Juropa
Architektur	QS22	Power6	BlueGene/P	IntelXeon X5570
Taktfrequenz	3.2 GHz	4.7 GHz	850 MHz	2.93 GHz
Knoten	35	14	73.728	3288
Rechenkerne				
pro Knoten	18	32	4	8
insgesamt	630	448	294.912	26.304
Hauptspeicher				
pro Knoten	8 GB	128 GB	2 GB	24 GB
insgesamt	280 GB	1,8 TB	144 TB	79 TB
Stromverbrauch				
insgesamt	4,7 kW	55 kW	2,52 MW	1,5 MW
pro Knoten	134,29 W	3.928,57 W	34,18 W	456,2 W
pro Rechenkern	7,46 W	122,76 W	8,54 W	57,03 W
Leistung				
Peak	7 TF	8,4 TF	1 PF	308 TF
Linpack	-	5,4 TF	825,5 TF	274,8 TF

6.2 Vergleich

Der Vergleich der Cell-Implementierungen untereinander und das Skalierungsverhalten machen noch keine Aussage über das Potential der Cell Broadband Engine. Es stellt sich die Frage, ob der Cell im Vergleich zu anderen Architekturen konkurrenzfähig ist. Ist die Leistung allein überzeugend genug, Aufwand in eine Neuimplementierung zu stecken oder gibt es andere ausschlaggebende Faktoren? Oder lohnt es überhaupt nicht?

6.2.1 Vergleichbarkeit

Ein direkter Vergleich des Original-Programmes mit dem neuem Modul ist mit etwas Vorsicht zu genießen. Davon mal abgesehen, dass zum Vergleich zwei vollkommen unterschiedliche Algorithmen antreten, die außer dem Ergebnis nichts gemeinsam haben, so ist das Modul noch nicht in sein eigentliches Umfeld eingebettet.
Das Fortranprogramm berechnet vor der eigentlichen Energieberechnung zuerst die Koordinaten der neuen Konfiguration. Die Konfiguratonsberechnung nimmt nur wenig Zeit im Vergleich zu Berechnung ein, jedoch ist sie in der Zeitmessung mit enthalten.
In der Cell-Implementierung ist dieser Anteil nicht mit enthalten. Dafür sollte dabei nicht nur die reine Berechnung sondern auch die zugehörige Übertragung der Daten auf die SPEs mit dazu gerechnet werden, da sie zu jeder Iteration dazu gehören. Weil aber mit double buffering versucht werden kann, den Transferanteil hinter der Berechnung zu verstecken, werden für den folgenden Vergleich beide Zeiten ausgewiesen. Wie das double buffering funktioniert wird im Ausblick (s. 7.2) beschrieben.

6.2.2 Alanin 15

Ein erster Vergleich der Rechenzeit auf Basis des aus 15 Aminosäuren bestehenden Polypeptids Alanin 15 (153 Atome):

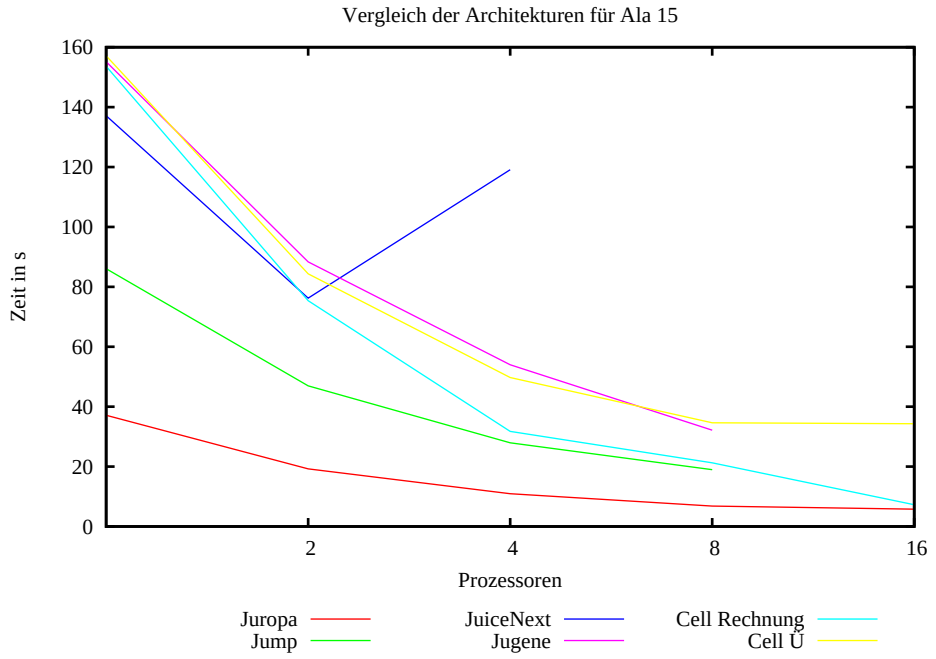

Abbildung 6.1: Vergleich der Architekturen für Alanin 15. Bei diesem Diagramm handelt es sich um den zeitlichen Vergleich zwischen den folgenden fünf Architekturen: Juropa (rot), Jump (grün), JuiceNext (nur PPE - blau), Jugene (magenta) und JuiceNext (nur SPE - reine Berechnung türkis, mit Übertragung gelb). Die Zeit (in Sekunden) wird gegenüber der Anzahl von 1 bis 16 Prozessoren aufgetragen. Das simulierte Peptid besteht aus 15 Alaninen (Ala_{15}).

Deutlicher „Sieger“ ist Juropa mit knapp 37 Sekunden auf einem Rechenkern. Mit mehr als der doppelten Zeit von 86 Sekunden ist Jump der Zweitschnellste. JuiceNext erreicht sowohl mit dem Original-Programm, das über die PPEs parallelisiert wurde, als auch mit der SPE-Parallelisierung etwas bessere Zeiten als Jugene. Damit braucht die Cell Broadband Engine ungefähr vier mal so lang wie Juropa.
Bis auf den Lauf auf der PPE des JuiceNext skalieren die Programme auf allen Architekturen bis zu acht Prozessoren für diese Problemgröße. Bei der Parallelisierung über 16 Prozessoren stagniert oder verschlechtert sich die Skalierung. Lediglich die reine Berechnung auf den SPEs folgt dem linearen Trend, doch die Übertragung macht diesen Vorteil in der Gesamtstatistik zunichte.
Auf den ersten Blick erstaunlich, dass der drittschnellste Rechner der Welt in diesem Vergleich so schlecht abschneidet. Seine Leistung zieht er jedoch aus seiner großen Anzahl an Prozessoren. Ein einzelner Prozessor kann aber nicht mit den höher getakteten Prozessoren Juropas mithalten.

6.2.3 Arginin 50

Der zweite Vergleich basiert auf dem größten Beispiel, das aus dem vorangegangen Kapitel bekannt ist. Es handelt sich dabei um eine Kette von 50 Arginin-Aminosäuren.

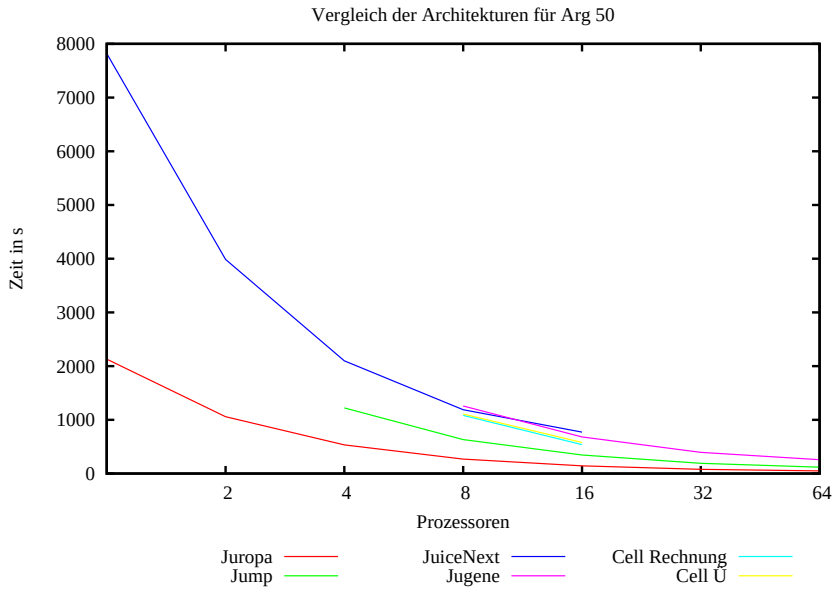

Abbildung 6.2: Vergleich der Architekturen für Arginin 50. Wiederum der zeitliche Vergleich zwischen den fünf Architekturen. Das Protein besteht aus 50 Argininen. Nicht für jede Architektur konnte bei jeder Prozessoranzahl ein Wert ermittelt werden. Nur auf Juropa konnte das gesamte Spektrum von 1 bis 64 Prozessoren gemessen werden. Für die Cell-Implementierung waren nur Messungen auf 8 und 16 SPEs möglich, da bei kleineren Aufteilung die Listen nicht komplett in den LS passten. Für Jump und Jugene stehen keine Werte mit 1 und 2 Prozessoren (Jump), 1 bis 4 Prozessoren (Jugene) zur Verfügung, weil die Rechnung mehr als 1800 s gedauert hätte, was auf diesen beiden Systemen das Limit für kleine interative Jobs ist.

Die Rangfolge in Punkto Schnelligkeit hat sich mit dem größeren Problem natürlich nicht verändert. Doch sieht man nun, dass das Skalierungsverhalten für das Fortran-Programm bei entsprechender Problemgröße auch weiter als bis zu acht Prozessoren reicht. Auch hierbei bleibt der Abstand zwischen Juropa und der Cell Broadband Engine beim Vierfachem. Eine Zeit auf zwei bzw. vier Prozessoren des Juropa entspricht in etwa der Zeit von acht bzw. 16 SPEs der Cell Broadband Engine.
Dass für die Cell-Implementierung nur Zeiten für acht und 16 SPEs vorliegen, liegt daran, dass die kompletten Atom- und Torsiondaten plus die komplette Liste der nahe Nachbarn und die Teillisten der 1-4-Nachbarn und Wasserstoffbrückenbindungen so groß sind, dass erst bei einer Aufteilung auf acht SPEs alle Daten in den LS passen.

6.2.4 Warum dann Cell?

Im direkten Vergleich hat die Cell Broadband Engine weniger gut abgeschnitten. Die Implementierung ist gut doppelt so langsam wie auf Jump und viermal so langsam wie auf Juropa. Aber sie ist vergleichbar mit Jugene, dem weltweit drittschnellsten Supercomputer. Wenn man sich die Tabelle der Kenngrößen (s. 6.1.1) betrachtet, so sieht man, dass sie noch in einer zweiten Größe vergleichbar sind: in ihrem Stromverbrauch pro Prozessor. Nähern wir uns der Frage also aus dieser Richtung: wieviel Leistung erzielt man mit der Cell Broadband Engine wenn man den Stromverbrauch betrachtet.

Dazu eine einfache Gegenüberstellung:
Für das letzte Beispielprotein aus 1153 Atomen brauchte Juropa mit dem Fortran-Programm auf vier Prozessoren ungefähr genauso lang, wie der Jump auf acht oder der Jugene/die Cell Broadband Engine auf 16 Prozessorkernen/SPEs. Rechnet man die Anzahl der Prozessoren einmal um auf den Stromverbrauch ergibt sich folgende Statistik:

1. 134,29 W Cell Broadband Engine (1 Blade/Knoten)
2. 136,64 W Jugene (16 Kerne)
3. 228,12 W Juropa (4 Kerne)
4. 982,03 W Jump (8 Kerne)

Die beiden Letzten in der zeitlichen Statistik schneiden nun am besten ab. Juropa erzielt die vierfache Leistung pro Kern, braucht aber auch fast das doppelte an Strom bei gleicher Leistung. Der Jump bildet das absolute Schlußlicht als größter Stromverbraucher.
Für die Einstufung der Green500, die den energiesparendsten Supercomputer sucht, werden die Flops pro Watt gewertet. Dies ist hier jedoch nicht angebracht, da der Algorithmus der Cell-Implementierung mehr Floating-Point-Operationen ausführt als das Original-Programm. Doch wurde dieser Umweg ja nur aus Leistungsgründen gemacht und sollte deshalb nicht noch zusätzlich positiv gewertet werden.

6.3 Zusammenfassung

Um die Leistung eines Computers zu steigern erhöht man die Anzahl der Prozessoren und achtet auf eine gute Vernetzung. Doch heißt eine Steigerung der Prozessoren auch eine Steigerung der Transistoren und damit auch des Stromverbrauchs. Um der Rechenleistung, deren Bedarf in den nächsten Jahren noch stark steigen wird, gerecht zu werden muss auch der Faktor Energiebedarf einkalkuliert werden.
In dieser Hinsicht macht die Cell Broadband Engine einen Schritt in die richtige Richtung. Nicht umsonst basiert der aktuell führende Top500-Rechner auf einem Hybrid-System mit Cell-Prozessoren als Rechenbeschleuniger. Das Ergebnis lässt sich bzgl. Energieeffizient zeigen, denn platziert es sich in der Green500 auf Platz drei.
Alles im Allem wurde eine Leistung erzielt, die sich nicht von den derzeitigen Supercomputern abschütteln lässt. Dabei muss man einräumen, dass die Ergebnisse nur mit einfach-genauer Floating-Point-Arithmetik erzielt wurden. Die Anwendungen für die Cell Broadband Engine sollten nicht nur deswegen sorgfälltig ausgewählt werden. Doch nimmt man dies in Kauf und steckt entsprechenden Aufwand in die Neuimplementierung, kann man gute Leistungen erwarten.

7 Ausblick

Wie Kapitel 6 gezeigt hat, schneidet diese Implementierung auf der Cell Broadband Engine gegenüber den Supercomputer nicht schlecht ab. Außerdem steht bei der jetzigen Cell-Implementierung noch Optimierungsarbeit an. Denn bis jetzt wurde erstmal nur das beste Vorgehen für die Berechnung gefunden.
Wie man in Abbildung 5.6, dem Skalierungsverhalten über die SPEs, sieht, nimmt die Übertragung der Daten zwischen den Berechnungen gerade bei den kleinen Problemen viel Zeit in Anspruch. Im Abschnitt 5.3 wurde bereits angerissen, dass dieser Faktor mit double buffering auf ein Minimum reduziert werden kann. Zudem begünstigt eine zusätzliche Veränderung der Parallelisierung, dass die Problemgröße kein begrenzender Faktor für diese Implementierung ist. Doch der erste Verbesserungsschritt wäre ein reduzieren der übertragenen Datenmenge.
In diesem letzten Kapitel sollen die Gründe für eine Verbesserung und erste Ansätze für die Umsetzung geklärt werden. Am Ende jeder der drei folgenden Verbesserungsschritte wäre das Programm lauffähig, sodass der Zeitgewinn gemessen werden kann.

7.1 Reduzieren der Datenmenge

Zu Beginn jeder Energieberechnung müssen die Daten der Atome und Torsionswinkel auf die SPEs übertragen werden. Ein Atom wird durch seine Bezeichnung, seinen Typ, seine Ladung und seine kartesischen Koordinaten beschrieben. Ein Torsionswinkel durch seine Bezeichnung, seinen Typ und seinen Wert. Bislang wird jeweils die komplette (gepaddete) Datenstruktur auf die SPEs transferiert. Dabei ändern sich zwischen zwei Energieberechnung eines Proteins die Atome und Winkel nicht komplett. Lediglich die Koordinaten der Atome und die Werte der Winkel werden durch die Monte-Carlo-Simulation verändert. Nur wenn die sich veränderten Werte zusammenhängend abgespeichert werden, können sie mit einem DMA-Transfer auf die SPE geholt werden. Dafür müssen sie separat modelliert werden. Für die Klasse Atom bedeutet dies eine Klasse Koordinaten. Die Funktion der Abstandsberechnung in der Klasse Atom muss entsprechen angepasst werden, um auf die Koordinaten korrekt zuzugreifen. Für etwas anderes werden die Koordinaten innerhalb der Energieberechnung nicht benötigt, deshalb sollte auch der Zugriff darauf nur über die Klasse Atom möglich sein. Zudem bleibt damit die Benutzung der Funktion wie gehabt intuitiv und es muss im Programm nichts geändert werden.
Für die Torsionswinkel gilt das Gleiche. Für dessen Werte müsste eine neue Klasse erstellt werden und der Zugriff über die Klasse Torsion geschehen. Vorsicht ist dabei beim DMA-Transfer bzgl. der Transfergröße geboten. Die Koordinaten des Atoms werden als Vektordatentyp, der 16 Byte groß ist, übertragen. Der Wert des Torsionswinkels nimmt

aber nur 4 Byte ein. Bei der Übertragung muss daher die Größe entsprechend aufgefüllt werden und auch der Empfangspuffer entsprechend groß angelegt werden, damit keine anderen Daten versehentlich überschrieben werden. Für die Übertragung müssen nur die Hauptspeicheradressen der beiden neuen Klassen auf den SPEs über den Context bekannt gemacht werden.

7.2 double buffering

Das Prinzip des double bufferings kommt aus der Computergrafik. Für die Ausgabe von Videosequenzen oder Spielegrafiken müssen zum einem die aktuellen Bilddaten aus dem Speicher gelesen werden und gleichzeitig die Daten für das nächste Bild berechnet und gezeichnet werden. Ohne double buffering würde der Speicher aus dem gelesen wird mit den neu gezeichneten Daten überschrieben. Dadurch, dass beides parallel stattfindet, kann es zu Bildflackern kommen. Deshalb wurde der Speicher in einen Front- und Backpuffer geteilt. Die Daten im Frontpuffer werden zur Ausgabe gelesen, neu berechnete Daten werden in den Backpuffer geschrieben. Zwischen zwei Ausgaben müssen dann nur die Adressen der Puffer getauscht werden. Das Prinzip bleibt das Gleiche, nur dass es darum geht Daten in den Speicher zu holen und nicht auszugeben.
Da die SPU und der MFC getrennt voneinander arbeiten können, kann man die Übertragungszeit mit Berechnungen verstecken, indem sie parallel verlaufen. Dazu müssen auf dem SPE zwei Puffer für zu empfangende Daten reserviert sein. Zwischen zwei Berechnungen tauscht man dann die Adressen aus, wohin neue Daten geschrieben werden sollen und mit welchen gerechnet wird. Dies wären die Koordinaten der Atome und die Werte der Torsionswinkel.
Der Nachteil des Prinzips ist, dass der doppelte Speicherplatz benötigt wird. Da der LS nur 256 KB groß ist, grenzt dies erstmal die Größe der zu berechnenden Proteine ein. Das Problem ist, dass alle Daten für eine Berechnung und nun auch für die Nächste im LS liegen müssen. Dieser Missstand soll im letzten Schritt behoben werden.

7.3 Blockweise Parallelisierung

Eine andere Parallelisierung soll helfen dieses Problem zu beheben. Denn für die derzeitige Parallelisierung der „Überschlagsrechnung“ sind alle Daten der Atome nötig. Mit einer blockweisen Parallelisierung der Wechselwirkungsmatrix wäre es möglich mit nur einem Teil der Daten, die immer wieder nachgeladen werden, die Energieberechnung für beliebig große Proteine zu vollziehen. Durch das double buffering macht sich auch eine Einteilung in kleinere Übertragungen nicht in der Ausführzeit bemerkbar.
Nacheinander werden den SPEs Blöcke der Wechselwirkungsmatrix zugewiesen, wobei Blöcke über der Diagonalen übersprungen werden. Dazu werden die zugehörigen Atomdaten der entsprechenden Zeilen und Spalten des Blockes geliefert.
Zu berücksichtigen sind bei diesem Verfahren die Listen. Zu jedem Block muss es auch jeweils Listen geben, die nur die Einträge für diesen Block enthalten, da nur dann die zugehörigen Atomdaten auf dem SPE verfügbar sind.

Für die Berechnung des Torsionsanteils werden die Torsionswinkel nun ebenfalls block- und nicht zeilenweise über die SPEs verteilt.
Außer, dass diese Parallelisierung es möglich macht, beliebig große Proteine zu berechnen, sollte auch die Lastbalancierung bei der „Überschlagsrechnung“ verbessert werden. Bis auf die Blöcke, die über die Diagonale ragen, haben alle SPEs die gleiche Anzahl an Berechnung durchzuführen.
Die neue Verteilung der Listen ist nicht mehr so, dass jedes SPE gleich viele Listenelemente erhält. Allerdings sollte dies kein großer Nachteil sein. Die Wasserstoffbrückenbindungen sind zufällig über die Matrixeinträge verteilt, sodass diese Liste weitgehend gleichmäßig verteilt sein wird. Nahe Nachbarn finden sich zur Hälfte immer in der Nähe der Diagonalen, sodass ein möglicher Overhead dadurch abgefangen wird, da dort weniger Berechnungen für die „Überschlagsrechnung“ zu machen sind.
Es ist aber zu testen, was die beste Blockgröße ist. Sie sollte so groß sein, dass die komplette Übertragung des nächsten Blockes durch die aktuelle Berechnung versteckt wird und knapp unter der Größe, das der DMA-Transfer aufgrund der 16kB Grenze aufgeteilt werden muss. Denn dann hätte man eine zweite Latenzzeit.

7.4 Einsatz

Wie bereits erwähnt soll das originale Fortran-Programm verwendet werden, um die Monte-Carlo-Simulation und das Parallel Tempering auf dem PPE durchzuführen und die Energieberechnung mit diesem Modul auf die SPEs auszulagern. Mit den geplanten Verbesserungen sollte die Zeit auf die reine Berechnungszeit reduziert werden können. Eventuell verbessert sie sich aufgrund der neuen Parallelisierung sogar noch. Auf jeden Fall macht sie es möglich beliebig große Proteine zu berechnen. Damit steht die Cell-Implementierung dem Original-Programm im Funktionsumfang in nichts nach.
Angedacht sind dann längere Monte-Carlo-Simulationen für größere Proteine als zuvor. Die Ergebnisse der Simulationen sollen genauere Auskunft über den Faltungswegs selbst und die Intermediate, die zwischenzeitlich eingenommen werden, geben.
Mit der Implementierung des Proteinkraftfeldes ECEPP/3 auf der Cell Broadband Engine war außer der Erstellung dieses Moduls und der Möglichkeit der Nutzung dieser Rechenresource auch ein besseres Wissen über die Cell Broadband Engine zu bekommen wichtig. Es wurden Fallstricke und Probleme sowie erste Lösungsansätze aufgezeigt. Das Laufzeitverhalten wurde bezüglich Leistung und Energieeffizienz mit den aktuellen Supercomputern des JSC verglichen. Das zwischenzeitliche Ergebnis ist im Vergleich zu einen lang optimierten Programm als akzeptabel einzustufen und die nächsten Verbesserungsschritte wurden gegeben. Insgesamt wurden damit alle Anforderungen im Rahmen dieser Arbeit erfüllt.

A Literaturverzeichnis

[1] Alberts, B.; Bray, D.; Hopkin, K.; Johnson, A.; Lewis, J.; Raff, M.; Roberts, K.; and Walter, P.: Lehrbuch der Molekularen Zellbiologie (German Edition). Wiley-VCH, 2005.

[2] Bielka, H. and Börner, T.: Molekulare Biologie der Zelle. Gustav Fischer Verlag, 1995.

[3] Jakubke, H.D.: Peptide. Chemie und Biologie. Heidelberg: Spektrum, 1996.

[4] Brokenshire, D.A.: A Protein Primer: A Musical Introduction to Protein Structure. http://www.ibm.com/developerworks/power/library/pa-celltips1/.

[5] Protein Data Bank: Myoglobin 3D structure. http://de.wikipedia.org/w/index.php?title=Datei:Myoglobin.png.

[6] National Human Genome Research Institute: DNA, transcription. http://www.genome.gov/Images/EdKit/bio2c_large.gif.

[7] Schema, dass die Translation von mRNA und die Synthese von Proteinen am Ribosom zeigt. http://commons.wikimedia.org/wiki/File:Ribosome_mRNA_translation_en.svg.

[8] Berry, A. and Radford, S.E.: From Protein Folding to New Enzymes (Biochemical Society Symposia). Portland Press, London, 1 ed., 2001.

[9] Branden, C. and Tooze, J.: Introduction to Protein Structure: Second Edition. Garland Science, 2 ed., 1999.

[10] Dill, K.A. and Chan, H.S.: From Levinthal to pathways to funnels. Nature Structural Biology, 4, 1997.

[11] Leach, A.: Molecular Modelling: Principles and Applications (2nd Edition). Prentice Hall, 2 ed., 2001.

[12] Kaltner, D.H.: http://www2.vetmed.uni-muenchen.de/tiph_c/Vorlesungen/BIOCHEMneuFeb.pdf.

[13] Eisenmenger, F.; Hansmann, U.; Hayryan, S.; and Hu, C.: SMMP - A Modern Package for Protein Simulations. Computer Physics Communications, 138, 2001.

[14] Eisenmenger, F.; Hansmann, U.; Hayryan, S.; and Hu, C.: An enhanced version of SMMP open-source software package for simulation of proteins. Computer Physics Communications, 174, 2006.

[15] Meinke, J.H.; Mohanty, S.; Eisenmenger, F.; and Hansmann, U.H.E.: SMMP v. 3.0 - Simulating proteins and protein interactions in Python and Fortran. Computer Physics Communications, 178, 2008.

[16] Earl, D.J. and Deem, M.W.: Parallel tempering: Theory, applications, and new perspectives. Physical Chemistry Chemical Physics, 2005. http://www.rsc.org/ej/CP/2005/b509983h.pdf.

[17] Liste der schnellsten Supercomputer der Welt. http://www.top500.org/.

[18] Liste der energieeffizientesten Supercomputer der Welt. http://www.green500.org/.

[19] IBM: Schema processore Cell. http://de.wikipedia.org/w/index.php?title=Datei:Schema_Cell.png.

[20] OBrien, K.; OBrien, K.; Sura, Z.; Chen, T.; and Zhang, T.: Supporting OpenMP on Cell. International Journal of Parallel Programming, 36, 2008.

[21] IBM Redbooks: Programming the Cell Broadband Engine Architecture: Examples and Best Practices. Vervante, 2008.

[22] SPE Runtime Management Library - CBEA JSRE Series, 2006. https://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/1DFEF31B3211112587257242007883F3.

[23] Bauke, H. and Mertens, S.: Cluster Computing: Praktische Einfhrung in das Hochleistungsrechnen auf Linux-Clustern (X.systems.press) (German Edition). Springer, 1 ed., 2005.

[24] Homberg, W.: Introduction to Architecture and Programming of the Cell Processor. Tech. report, Jülich Supercomputing Centre des Forschungszentrums Jülich, 2008. http://www.fz-juelich.de/jsc/datapool/cell/GRS_Arch_Prog_Cell.pdf.

[25] Meinke, J.H. and Hansmann, U.H.E.: Single-molecule Protein Simulations on Thousands of Processors.

[26] Schubert, L.: Die Cell Broadband Engine und eine Wellengleichung - ein Lehrbeispiel fur Parallelisierung und SIMD -. Tech. report, Jülich Supercomputing Centre des Forschungszentrums Jülich, 2008.

[27] Clark, M.A.: Maximizing the power of the Cell Broadband Engine processor: 25 tips to optimal application performance. http://whozoo.org/mac/Music/Primer/Primer.htm.

B Glossar

Cache Ein Cache ist ein Zwischenspeicher, der Daten zwischen einem höherliegenden Speicher in der Speicherhierarchie und der CPU redundant hält, um die Zugriffszeit auf diese Daten zu reduzieren

Deadlock Ein Deadlock entsteht beim gegenseitigen Warten aufeinander bei Prozessen/Threads, die erst dann das Signal zur Freigabe schicken, wenn sie selbst eines erhalten haben

Eukaryont Ein Eukaryont ist ein Lebewesen, dass im Gegensatz zu Prokaryonten Zellkern und Zellmembran hat. Zudem besitzen sie mehrere Chromosomen.

Distributed-Memory System In einem Distributed-Memory System hat jede Systemkomponente ihren eigenen Speicher

FPGA FPGA steht für Field Programmable Gate Array und steht für Schaltkreise, deren spezielle Funktion nachträglich programmiert werden kann ohne an der Hardware etwas zu ändern

Lastbalancierung Unter Lastbalancierung zwischen verschiedenen Prozessoren/Kernen versteht man eine gleichmäßige Verteilung der Rechenlast

Linpack Linpack bezeichnet ein Programm zum Messen der Leistung von Supercomputern. Es bedient sich dabei der gleichnamigen Programmbibliothek zum Lösen von linearen Gleichungssystemen. Das Ergebnis wird in Floating-Point-Operationen per Second (FLOPS) angegeben. Dieser Wert gilt als Vergleichswert für das Ranking in der Top500.

OpenMP OpenMP ist eine Programmierschnittstelle ähnlich MPI für Shared-Memory-Programmierung. Die Parallelisierung finden hier auf Threadebene statt

Shared-Memory System In einem Shared-Memory System greifen die verschiedenen Systemkomponenten auf den gleichen Speicher zu. Das Memory Managment System übernimmt die Koordination der Speicherzugriffe. Bei Cache-basierten Memory-Systemen werden Strategien benötigt, die sicherstellen, dass auf das jeweils aktuelle Speicherdatum zugegriffen wird. Im Falle der Cell Broadband Engine ist auf den SPEs kein Cache vorhanden, sodass der Programmierer beim DMA-Transfer selbst darauf Acht geben muss

unidirektionaler Ringbus Ein Datenbus, der ringförmig mit allen Teilnehmern verbunden ist. Unidirektion bedeutet dabei, dass die Daten nur in eine Richtung gesendet werden können