Zentralinstitut für Angewandte Mathematik

Interner Bericht

Beiträge zum Wissenschaftlichen Rechnen – Ergebnisse des Gaststudentenprogramms 2001 des John von Neumann-Instituts für Computing

Rüdiger Esser (Hrsg.)

FZJ-ZAM-IB-2001-12

FORSCHUNGSZENTRUM JÜLICH GmbH
Zentralinstitut für Angewandte Mathematik
D-52425 Jülich, Tel. (02461) 61-6402

Interner Bericht

Beiträge zum Wissenschaftlichen Rechnen – Ergebnisse des Gaststudentenprogramms 2001 des John von Neumann-Instituts für Computing

Rüdiger Esser (Hrsg.)

FZJ-ZAM-IB-2001-12

November 2001

(letzte Änderung: 9. November 2001)

Vorwort

Die Ausbildung im Wissenschaftlichen Rechnen ist neben der Bereitstellung von Supercomputer-Leistung und der Durchführung eigener Forschung eine der Hauptaufgaben des John von Neumann-Instituts für Computing (NIC) und hiermit des ZAM als wesentlicher Säule des NIC. Um den akademischen Nachwuchs mit verschiedenen Aspekten des Wissenschaftlichen Rechnens vertraut zu machen, führte das ZAM in diesem Jahr zum zweiten Mal während der Sommersemesterferien ein Gaststudentenprogramm durch. Entsprechend dem fächerübergreifenden Charakter des Wissenschaftlichen Rechnens waren Studenten der Natur- und Ingenieurwissenschaften, der Mathematik und Informatik angesprochen. Die Bewerber mußten das Vordiplom abgelegt haben und von einem Professor empfohlen sein.

Die acht vom NIC ausgewählten Teilnehmer, unter ihnen eine Studentin aus Moskau, kamen für zehn Wochen, vom 6. August bis zum 12. Oktober, ins Forschungszentrum. Sie beteiligten sich hier an den Forschungs- und Entwicklungsarbeiten des ZAM, des Instituts für Festkörperforschung (IFF) und des Zentrallabors für Elektronik (ZEL) und wurden jeweils einem Wissenschaftler zugeordnet, der mit ihnen zusammen eine Aufgabe festlegte und sie bei der Durchführung anleitete.

Die Gaststudenten und ihre Betreuer waren:

Ekaterina Borodina	Mathilde Romberg	ZAM
Thorsten Gellermann	Bernd Mohr	ZAM
Michael Klocke	Artur Baumgärtner	IFF
Stefan Kunis	Gudrun Wagenknecht	ZEL
Katja Lord	Inge Gutheil	ZAM
Andreas Nußbaumer	Godehard Sutmann	ZAM
Gerald Schubert	Bernhard Steffen	ZAM
Sven Trentmann	Herwig Zilken	ZAM

Zu Beginn ihres Aufenthalts erhielten die Gaststudenten eine viertägige Einführung in die Programmierung und Nutzung der Parallelrechner im ZAM. Um den Erfahrungsaustausch untereinander zu fördern, präsentierten die Gaststudenten am Ende ihres Aufenthalts ihre Aufgabenstellung und die erreichten Ergebnisse. Sie verfaßten zudem Beiträge mit den Ergebnissen für diesen Internen Bericht des ZAM.

Wir danken den Teilnehmern für ihre engagierte Mitarbeit - schließlich haben sie geholfen, einige aktuelle Forschungsarbeiten weiterzubringen - und den Betreuern, die tatkräftige Unterstützung dabei geleistet haben.

Ebenso danken wir allen, die im ZAM und der Verwaltung des Forschungszentrums bei Organisation und Durchführung des diesjährigen Gaststudentenprogramms mitgewirkt haben. Besonders hervorzuheben ist die finanzielle Unterstützung durch den Verein der Freunde und Förderer der KFA und die Firma SGI.
Es ist beabsichtigt, das erfolgreiche Programm künftig zu wiederholen, schließlich ist die Förderung des wissenschaftlichen Nachwuchses dem Forschungszentrum ein besonderes Anliegen.

Weitere Informationen über das Gaststudentenprogramm, auch die Ankündigung für das kommende Jahr, findet man unter http://www.fz-juelich.de/zam/gaststudenten.

Jülich, November 2001 Friedel Hoßfeld, Rüdiger Esser

Inhalt

Automatisches Testen von Software: Eine Testumgebung für UNICORE

Ekaterina Borodina

Moskauer Staatliches Institut für Elektronik und Mathematik
(Technische Universität)

e.borodina@mtu-net.ru

Zusammenfassung: Das System UNICORE ist für die wissenschaftliche Arbeit mit Großrechnern vorgesehen. Diese System stellt ein ”Seamless Interface” für die einfache Benutzung der Ressourcen verschiedener Supercomputer-Zentren zur Verfügung. XML ist ein bequemes Format für die Datenstrukturierung. Dieses Format wird auch bei UNICORE benutzt werden. Die Funktionalität neuer UNICORE-Versionen soll automatisch getestet werden. Die Methoden hierfür durch Analyse und Bearbeitung von XML-Dateien wird vorgestellt. Der Algorithmus für die automatische Modifizierung der XML-Dateien wird erläutert.

Einleitung

Ein Softwaresystem wie UNICORE muß vor der Auslieferung ausgiebig getestet werden. Insbesondere die über ein graphisches Benutzerinterface zur Verfügung gestellen Funktionen müssen für jede neue Version überprüft werden. Diese Überprüfung soll zum großen Teil automatisch erfolgen. Das ist möglich durch das XML-Format der UNICORE Jobs. UNICORE erzeugt seine Angaben immer im internen AJO-Format (Abstract Job Objekt) und parallel dazu für die Interoperabilität auch im XML-Format.

Das XML-Format ist weit verbreitet, und es ist sehr einfach (wie HTML) XML-Dateien durch das Netz zu transportieren. Aber für die vorliegende Arbeit ist der wichtigste Vorteil des XML-Formats, daß es die Datenstruktur beschreiben kann.

Das Testsystem soll verschiedene Ressourcen automatisch prüfen können. Das System soll bequem für die Benutzer sein.

UNICORE

UNICORE (UNiform Interface to COmputing Resources) ist eine Schnittstelle für die einfache Benutzung von Großrechnern; es bietet ein ”Seamless Interface” für verschiedene Rechnersysteme.

Motivation

Wissenschaftler nutzen seit langem Rechner an entfernten Standorten, um komplexe Probleme aus der Computational Science zu lösen. Da Rechner verschiedener Hersteller zum Einsatz kommen, besteht ein erheblicher Lernaufwand, bevor ein neuer Rechner genutzt werden kann. Benutzer, die für ihre Anwendungen Supercomputer benötigen, sind gezwungen, sich mit den system- und rechenzentrumsspezifischen Gegebenheiten vertraut zu machen. Dieses kann sehr zeitaufwendig werden, insbesondere

wenn Rechner in verschiedenen Zentren genutzt werden. UNICORE wurde entworfen, um den Benutzern einen einfachen und sicheren Zugang zu verteilten Rechner-Ressourcen zu schaffen. UNICORE hat zum Ziel, die Stoßkanten zwischen den Systemen und zwischen den Zentren für der Benutzer zu beseitigen. Auch schafft UNICORE eine Software-Infrastruktur für einen einheitlichen und sicheren Zugang zu den Zentren. Dieses Ziel beinhaltet für den einheitlichen Zugang zum einen die Definition einer systemunabhängigen Job-Beschreibung zur Formulierung von Abstrakten Jobs und zum anderen die Schaffung einer einheitlichen UNICORE-Benutzeridentifikation. Für den intuitiven Zugang gehört eine ansprechende Benutzeroberfläche dazu und für die Sicherheit eine tragfähige, auf Standards basierende Sicherheitsarchitektur[1].

Die Drei-Ebenen-Architektur

Die Architektur besteht aus Benutzer-, UNICORE- und Systemebene. Das folgende Bild gibt einen Überblick über die Komponenten und ihr Zusammenspiel.

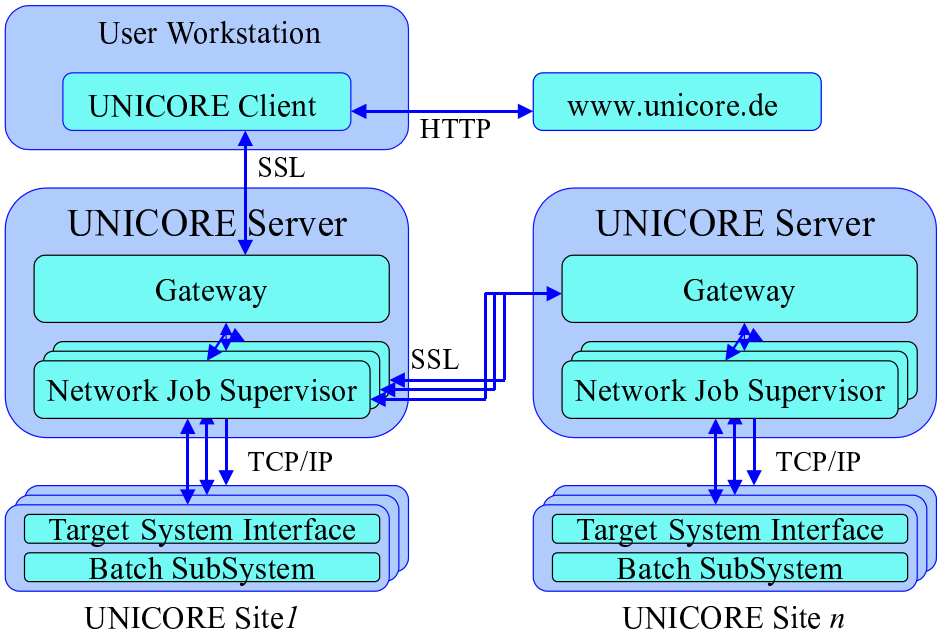


Abbildung 1: UNICORE Architektur.

Auf der Benutzerebene, der Workstation des Benutzers, läuft das grafische Interface zur Erstellung und Kontrolle von UNICORE-Jobs. Von dem zentralen Server www.unicore.de wird die Liste der verfügbaren UNICORE Sites abgefragt. Über das SSL-Protokoll (Secure Socket Layer) wird die UNICORE Server Ebene angesprochen. Diese besteht aus einem Gateway, welches die Benutzer-Authentifizierung durchführt, und Network Job Supervisor (NJS) Servern, die jeweils die Aufträge (Jobs, Statusabfragen) für eine Vsite verarbeiten. Auf der System-Ebene sorgt die Komponente Target System Interface (TSI) für die Schnittstelle zum lokalen Betriebssystem und Batch-Subsystem.

Benutzerfunktionen

Ein Benutzer erstellt einen UNICORE-Job (Ujob) für ein bestimmtes Zielsystem (Vsite) an einem Rechenzentrum (Usite), das UNICORE anbietet . Der Ujob kann aus mehreren Schritten bestehen. Ein Job-Schritt ist entweder eine Task, die in einem Batch-Job für der Zielsystem übersetzt wird, oder ein Teil-Job, der für eine andere Vsite bestimmt ist und selbst wieder aus Job-Schritten besteht. Job-Schritte

können voneinander anhängen. Zur Zeit werden nur sequentielle Abhängigkeiten unterstützt, d.h. eine Task bzw. ein Job-Schritt wird nur dann ausgeführt, wenn alle Vorgänger-Tasks bzw. Vorgänger-Job-Schritte erfolgreich ausgeführt worden sind. Voneinander unabhängige Teile können quasi parallel ausgeführt werden.

Grafische Oberfläche von UNICORE

Für die einfache Benutzerarbeit hat UNICORE eine grafische Oberfläche.

Übersicht

Dem Benutzer wird durch die oben beschriebene Architektur eine einheitliche Benutzer-Schnittstelle zu (Super-)Computern angeboten. Unabhängig vom jeweiligen Zielsystem werden Ressourcen-Anforderungen und Kommando-Optionen über eine graphische Oberfläche angegeben. Der Job Preparation Agent (JPA) erzeugt aus den Angaben den abstrakten Job (das AJO). Später wird dieser vom NJS in die konkreten Befehle und Optionsangaben für das vom Benutzer angewählte Zielsystem übersetzt. Das Zertifikat des Benutzers dient als einheitliche UNICORE-Benutzeridentifikation. Der Prototyp unterstützt derzeit die Erstellung und die Überwachung von Batch-Anwendungen. Die UNICORE-Jobs können aus mehreren Teilen aufgebaut werden, die asynchron auf verschiedenen Rechnern an verschiedenen Rechenzentren ablaufen können. Der Benutzer kann die Teile mit sequentiellen Abhängigkeiten verbinden (siehe Abbildung 2).

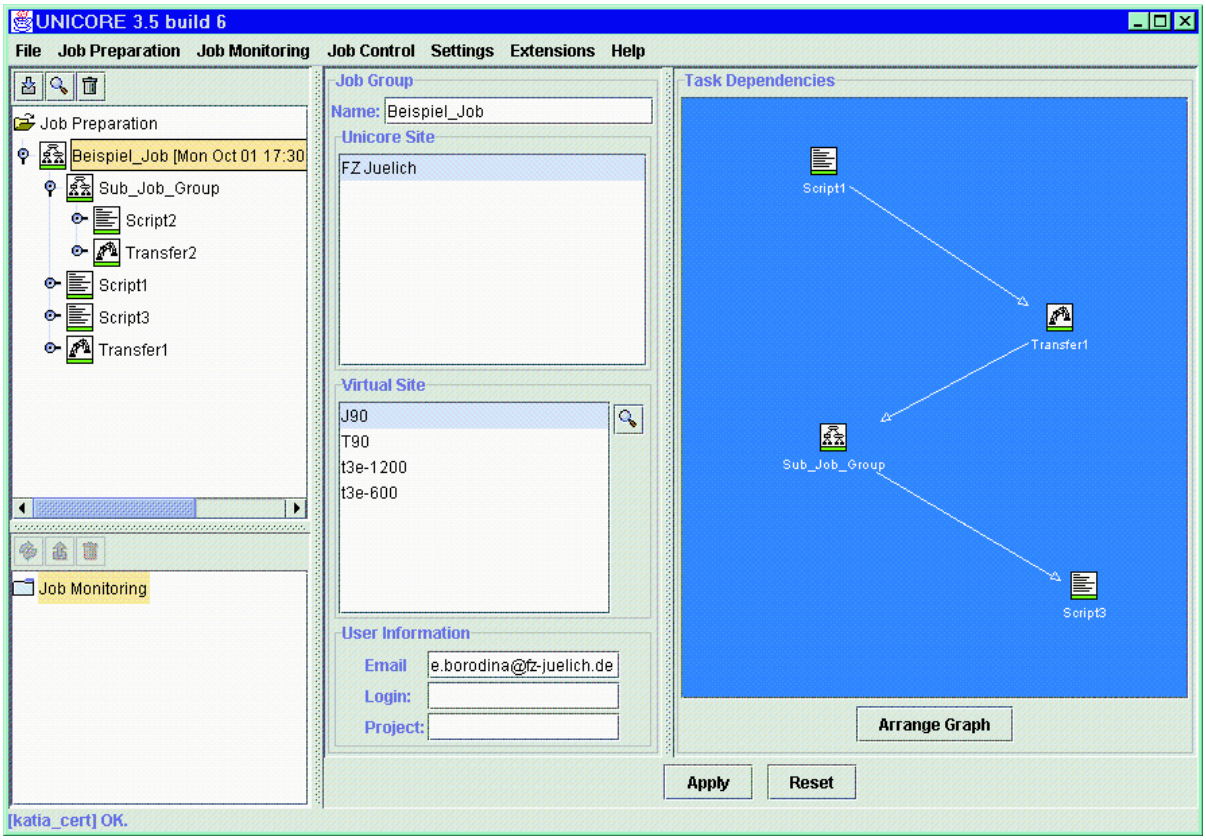


Abbildung 2: Grafische Oberfläche von UNICORE. Beispiel.

Über den Job Preparation Agent (JPA), das grafische Benutzerinterface zur Erstellung und Submission von UNICORE-Jobs, können derzeit Jobs aus den folgenden Elementen zusammengesetzt werden:

- Script Task
- Transfer Task
- Job-Gruppe, die Teil-Jobs für andere Zielsysteme enthält.
- Import Task
- Export Task

Ein neuer UNICORE-Job wird im JPA über die Auswahl ”New Job” im File-Menü erstellt. Mit dieser Funktion erzeugt der Benutzer den äußeren Rahmen für den UNICORE-Job, der allgemeine, für den gesamten Job relevante Information enthält:

- das Zielsystem, das aus einer Liste der in UNICORE verfügbaren Zielsysteme ausgewählt wird,
- den Jobnamen und
- die Email-Adresse des Benutzers, an die Nachrichten vom System geschickt werden sollen.

Für das Element Script-Task sind die benötigten Ressourcen zu spezifizieren. Zu den Ressourcen gehören:

- der Speicherplatzbedarf,
- die Anzahl der Knoten,
- die Anzahl der Prozessoren pro Knoten,
- die CPU-Zeit,
- die Plattenplatzbedarf.

XML - Format für UNICORE-Jobs

XML (Extensible Markup Language) gibt gute Möglichkeiten für die Datenstrukturierung. UNICORE kann alle seine Angaben über einen Job in XML-Format speichern. Diese Möglichkeit wurde speziell für das Testen der Software hinzugefügt.

XML Sprache

XML ist eine Metasprache zur Definition von Markup-Sprachen. Das ist die vereinfachte Form von SGML. So wie HTML mit SGML definiert ist, so kann man mit XML eigene Markup-Sprachen oder auch eigene Erweiterungen von HTML bzw. XHTML mit eigenen Tags für bestimmte Elemente mit bestimmten logischen Bedeutungen definieren. Die mit XML definierten Markup-Sprachen werden als XML-Anwendungen bezeichnet. Die Syntax, Struktur und Bedeutung der Tags wird für jede XML-Anwendung definiert.

- XML kann mit bereits existieren Web-Protokollen (z.B. HTTP) eingesetzt werden und stellt keine zusätzliche Anforderungen [2].

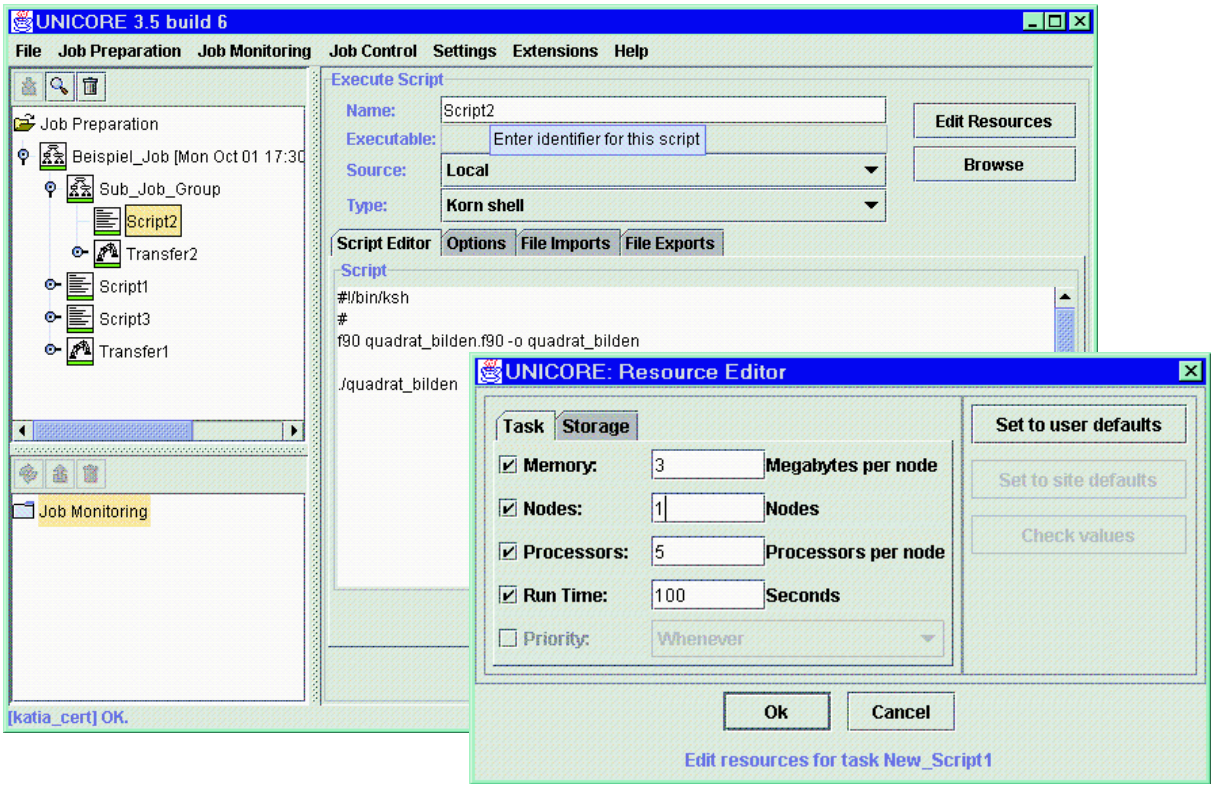


Abbildung 3: Grafische Oberfläche von UNICORE. Beispiel des Scripts.

- XML unterstützt eine Vielzahl von Anwendungen. HTML (Hyper Text Markup Language) ist oft nicht genug! Jede neue Version macht Probleme, weil ein neuer Browser notwendig ist. Im XML ist die Erstellung von neuen Kommandos möglich.
- Es ist einfach, Programme zur Verarbeitung von XML-Dokumenten zu schreiben. Das ist im Prinzip ähnlich wie HTML.
- Die Anzahl der optionalen Funktionen in XML wurde so klein wie möglich gehalten, damit die Verarbeitung von XML-Dokumenten wenig Zeit benötigt.
- XML-Dokumente sind selbst für den Laien relativ verständlich.
- Der Entwurf von XML ist formal und knapp.
- XML-Dokumente sind einfach anzulegen, wie HTML.

XML Dokument

- Beschreibt sämtliche Regeln, denen die Markup-Syntax folgen soll.
- Listet alle externe Ressourcen (external Entities) auf, die Teil des Dokumentes sind.
- Deklariert alle internen Ressourcen (internal Entities), die innerhalb des Dokumentes verwendet werden.
- Listet die Typen der Nicht-XML-Ressourcen auf.
- Listet alle Nicht-XML-Ressourcen (Binärdateien) auf.

In der Regel sollten die XML-Dokumente folgende Anforderungen erfüllen:

- In der erste Zeile des Dokuments befinden sich die Markup-Sprache, die Versions-Nummer und zusätzliche Information.
- Die Daten befinden sich zwischen Tags. Dabei muß, im Gegensatz zu HTML, jedes geöffnete Tag ein zugehöriges geschlossenes Tag haben.
- Innerhalb eines Tages kann man verschiedene Attribute definieren. Die Werte von Attributen sollten zwischen doppelten Hochkommata stehen.
- Die Reihenfolge der geschachtelten Tags in XML ist streng kontrolliert. Deswegen ist die Folge von geöffneten und geschlossenen Tags wichtig.
- Die Information, die sich zwischen den Anfangs- und End-Tags befindet, wird als Daten betrachtet und deshalb werden alle Symbole für die Formatierung, z.B. Blanks, Zeilenumbrüche nicht ignoriert, im Gegensatz zu HTML.

Falls ein XML-Dokument den oben genannten Regeln entspricht, heißt es formal richtig und die Parser für XML-Dokumente können es korrekt einlesen.

Das ist ein Beispiel für ein XML-Programm:

```
< ? xml version=Ş1.0Ť?>
 <contacts>
   <contact>
      <name>
         <first>John</first>
         <last>Belcher</last>
      </name>
      <address>
         <street>Pennington 13322</street>
         <city>Washington</city>
      </address>
      <tel>888 77 66</tel>
      <email>jb@southside.com</email>
   </contact>
</contacts>
```

Man sieht die Dokumentstruktur: jeder ”contact” ist innerhalb ”contacts” geschrieben und besteht aus ”name” (”name” besteht aus ”first” and ”last”), ”address”, ”tel” und so weiter. Ein XML-Dokument besteht aus einer Folge von Tags und Werten. Jedes Tag hat einen formal Anfang (z.B. ”<first>”), einen Wert (”John”) und ein Ende (”</first>”).

Testmöglichkeiten

Es gibt verschiedene Möglichkeiten, neue Softwareversionen zu testen. Als Beispiel sind drei Schemas gezeigt (Abbildung 4).

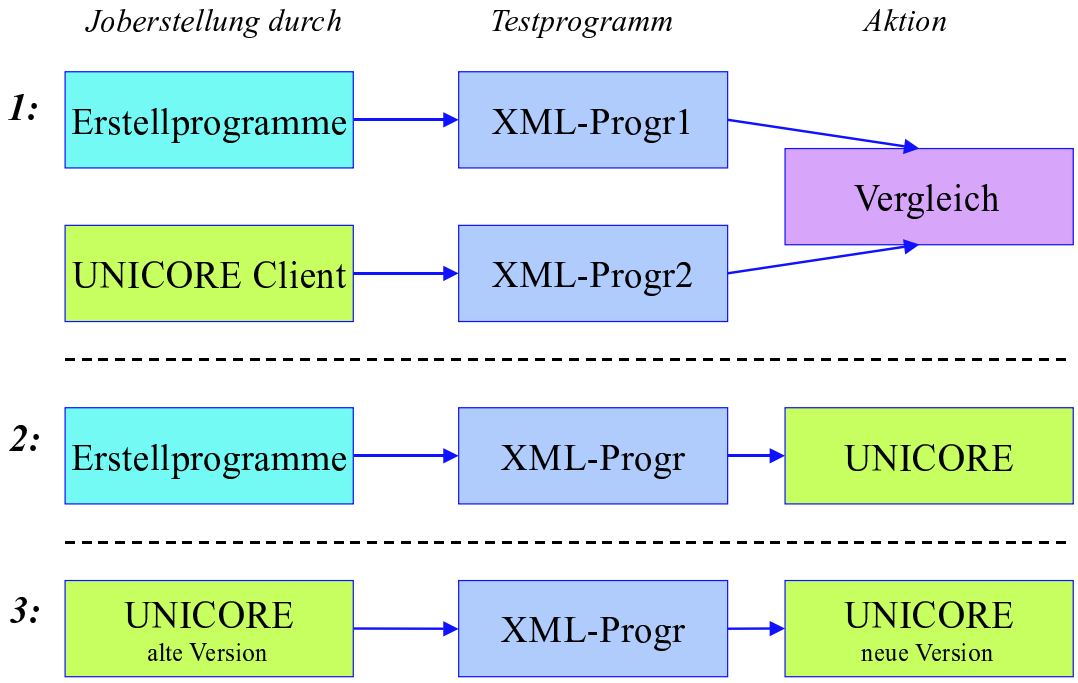


Abbildung 4: Testmöglichkeiten.

Das erste Schema zeigt die UNICORE-Jobs im XML-Format (XML-Progr1&2). Der erste Job wurde direkt von UNICORE erzeugt, und der zweite Job wurde mit Hilfe eines Erstellungsprogramms (Job-Generator) erstellt. Das bietet die Möglichkeit, beide Jobs zu vergleichen, um eventuelle Fehler im UNICORE-Job der neuen UNICORE-Version zu finden.

Das zweite Schema gibt die Möglichkeit, den UNICORE-Job in XML-Format (XML-Progr) ohne die grafische Oberfläche zu erstellen und dann dieses Programm in UNICORE zu submittieren und auszuführen.

Das dritte Schema beschreibt die Erstellung eines XML-Programms direkt aus der Vorgänger-Version des UNICORE und die Ausführung dieses XML-Programms in der neuen Version.

Schema 1 hat den Nachteil, daß es in diesem Fall zu schwierig ist, einen konkreten Vergleich zu machen. Hier kann man den Unterschied zwischen wichtigen und unwichtigen Fehlern nicht leicht finden. Schemas 2 & 3 testen die UNICORE Funktionalität in der neuen Version. Der von UNICORE erzeugte Output kann dann analysiert werden. Aber das zweite Schema bietet auch die Möglichkeit, Werte zu modifizieren oder neue Werte für zu testende Funktionen zu machen unabhängig von der grafischen Oberfläche von UNICORE.

Testalgorithmus

Unsere Testalgorithmus ist eine Kombination aus Schema 2 und 3 (Abbildung 4).

Die Abbildung 5 zeigt den Softwaretestalgorithmus. Dieses Test-XML-Programm soll modifizieren werden. Das Modifier-Programm nimmt dieses Programm als Input und bietet dem Benutzer die Möglichkeit, verschiedene Parameter zu modifizieren. Zu einem späteren Zeitpunkt wird dieses Programm von UNICORE ausgeführt. Und als letzter Schritt wird die Analyse des UNICORE Output durchgeführt.

Die Erstellung von XML-Programm ist direkt mit UNICORE möglich. Diese Variante ist geeignet, wenn man die Unterschiede zwischen verschiedenen Versionen von UNICORE testen möchte. Die Erstellung

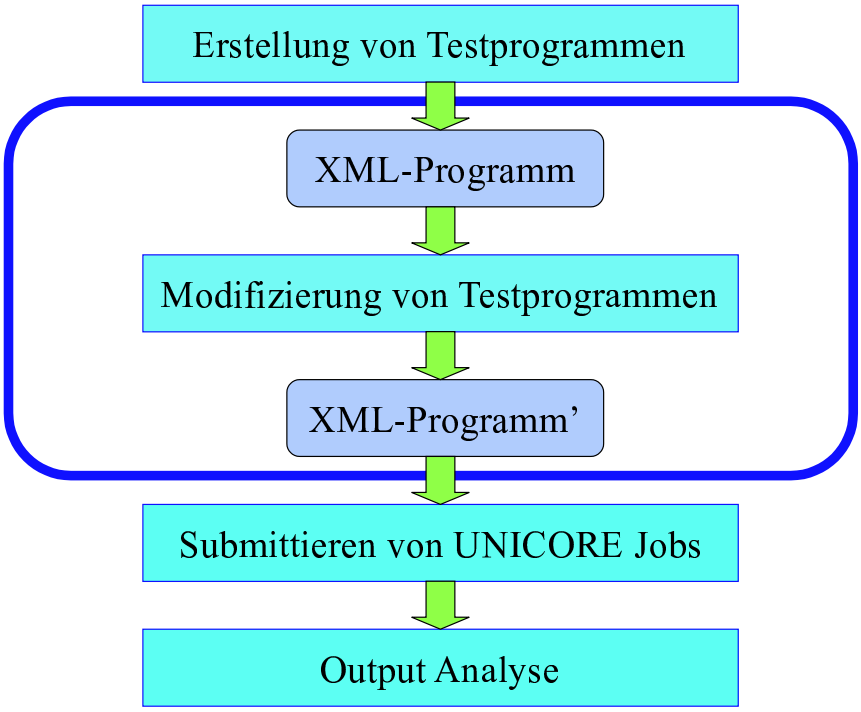


Abbildung 5: Testalgorithmus.

der XML-Programme durch ein spezielles Programm (Job Generator) ist auch möglich.

Der Output wird von UNICORE erzeugt. Nach Ausführung eines Jobs listet UNICORE alle Informationen über die Job-Ausführung auf. Ein Beispiel ist in Abbildung 6 präsentiert. Man kann sehen, welche Jobs erfolgreich ausgeführt wurden, in welchen Zielsystemen, was nicht funktioniert, und so weiter.

Meine Arbeit hat sich auf den Teil, der in Abbildung 5 eingerahmt ist, konzentriert. Dieser Teil muß automatisiert werden. Für dieses Ziel muß ein besonderes Programm geschrieben werden. Im Folgenden wird dieses Programm ”Modifizierprogramm” genannt.

Algorithmus des Modifizierprogramms

Das Algorithmus besteht aus folgenden Schritten:

- Analysieren des XML-Programms. In Fall eines komplexen Jobs bedeutet das die Erstellung von Datenstrukturen, mit deren Hilfe das Programm bearbeitet wird.
- In Fall des Standard-Jobs (UNICORE-Jobs sind in der Regel nicht so kompliziert und dann ist es nicht so wichtig zu wissen, was innerhalb der Jobs steht) verändert das Programm die spezifizierten Ressourcen und weist ihnen neue Werte zu.
- In Fall eines komplexen Jobs (große Anzahl von Sub_Job_Groups oder tief geschachtelte Jobs) ist der Dialog mit dem Test-Benutzer notwendig (durch ein “friendly Interface”). Z.B. schreibt das Programm zunächst die Taskstruktur auf und dann erfolgt der Dialog mit dem Benutzer durch ein Menü-Interface.
- Veränderung des XML-Testprogramms mit den neuen Werten.

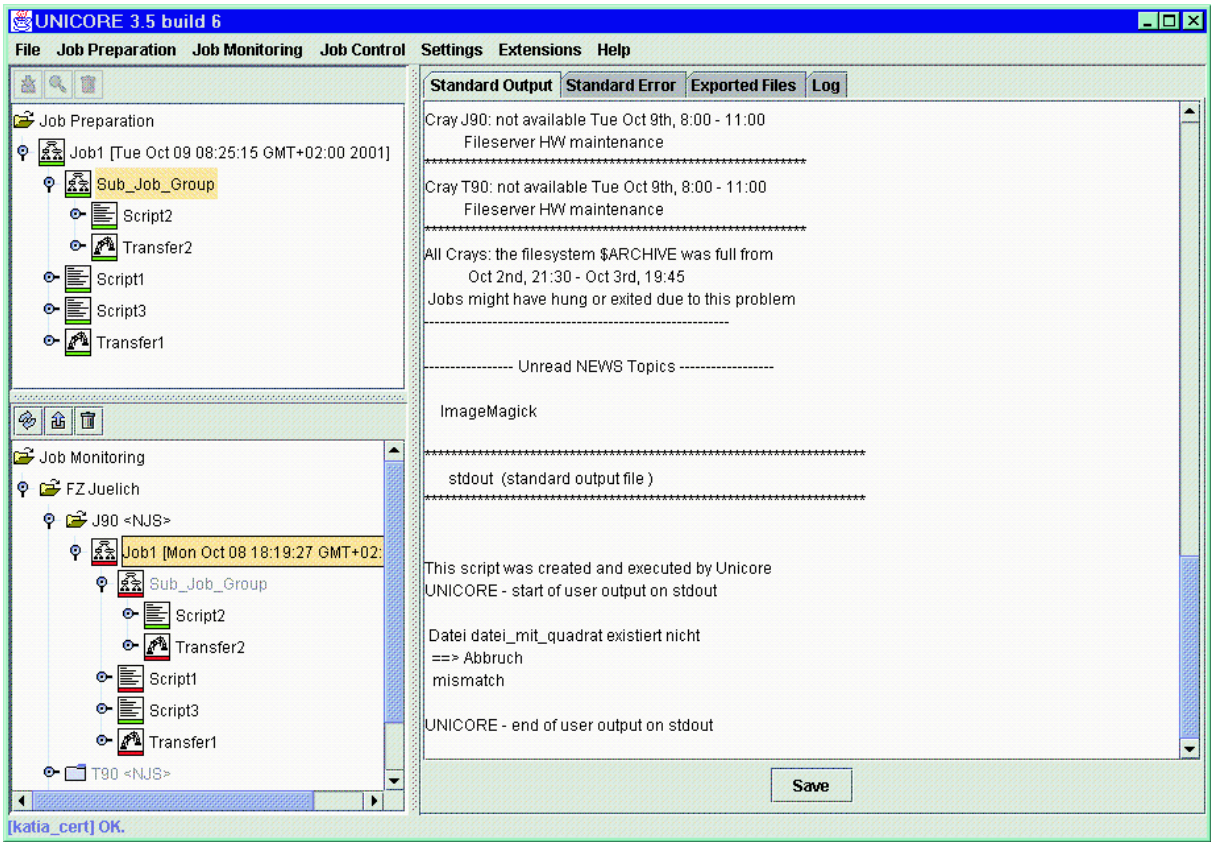


Abbildung 6: UNICORE Standard Output.

Analyse der Taskstruktur

Im ersten Schritt muß das Modifizierprogramm die XML-Struktur analysieren. Das Schema (Abbildung 7) zeigt die zweistufige Taskstruktur. Jedes Kästchen bedeutet eine Task. Die Kästchen zeigen die TASKS, geschrieben im XML-Format. Jede dieser Sub-Tasks kann jeweils nur ein Element aus UNICORE enthalten. Es gibt folgende UNICORE-Elemente: Job Gruppe, Script und Transfer. Dateien, die im Script importiert und aus dem Script exportiert werden, und Ressourcen, sind Sub-Tasks von Script-Task.

Jeder UNICORE-Job hat diese Struktur, die als XML-Script repräsentiert wird. Ein Beispiel wird im folgenden Abschnitt gezeigt.

XML Beispiel für Script

Hier wird ein Teil einer von UNICORE erzeugten XML-Datei als Beispiel präsentiert:

```
<TASKS ID="110" i="1"valuetype="com.pallas.....script.ScriptContainer">
  <PARENT IDREF="1" />
  <name ID="114" valuetype="java.lang.String">Script1</name>
  <RESOURCES ID="115" valuetype="org.unicore.sets.ResourceSet">
  </RESOURCES>
  <IMPORTS ID="117" dim="1" length="2"
    <IMPORTS ID="118" i="0" valuetype="com.pallas.unicore.   ...   .FileImport">
      <sourceName ID="119" valuetype="java.lang.String">C:\Unicore\Test</sourceName>
      <destinationName ID="120" valuetype="java.lang.String">vergl.f</destinationName>
```

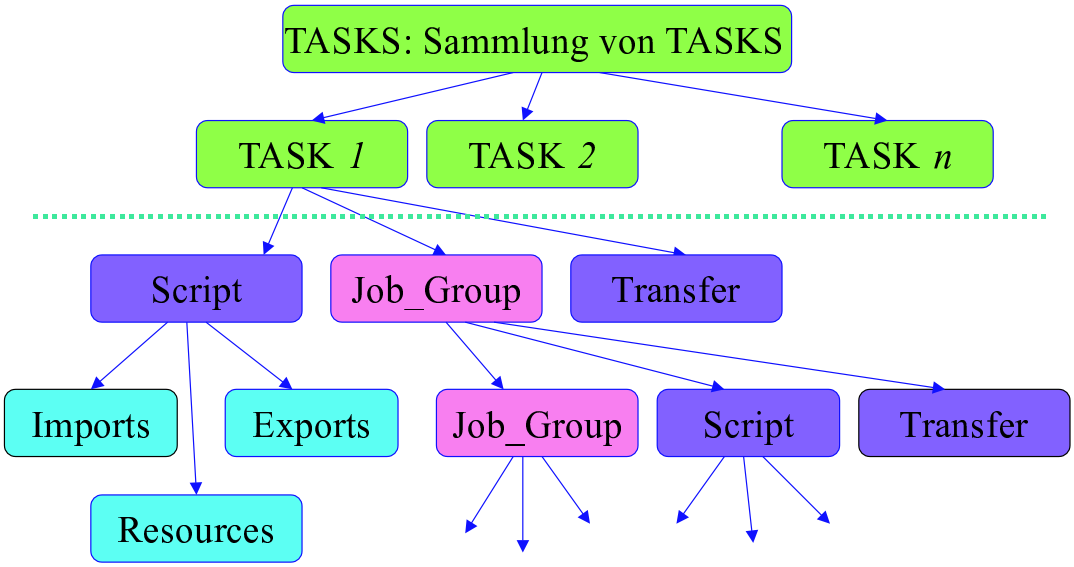


Abbildung 7: Zweistufige Taskstruktur.

```
    </IMPORTS>
    <IMPORTS ID="121" i="1" valuetype="com.pallas.unicore.    ...   .FileImport">
      <sourceName ID="122" valuetype="java.lang.String">C:\Katia\Unicore</sourceName>
      <destinationName ID="123" valuetype="java.lang.String">imp2.f</destinationName>
    </IMPORTS>
  </IMPORTS>
  <EXPORTS ID="124" dim="1" length="0"
  <TRANSFERS ID="125" dim="1" length="0" valuetype="[   ...   .FileTransfer;" />
  <SCRIPTCONTENTS ID="141" valuetype="java. ...">#!/bin/ksh # f90
     wurzel_ziehen.f90 -     o wurzel_ziehen ./wurzel_ziehen</SCRIPTCONTENTS>
  <SCRIPTDIRECTORY IDREF="108" />
</TASKS>
```

An diesem Beispiel kann man sehen, daß jedes UNICORE-Datenelement (JOB, SCRIPT, IMPORT, EXPORT und so weiter) im XML-Format als TASK bezeichnet ist. Jede TASK hat eine eigene ID. Geschachtelte TASKs im XML-Format entsprechen geschachtelten UNICORE Elementen.

Datenstruktur

Abbildung 8 zeigt Datenstrukturen für die Analyse des UNICORE-Jobs. Jeder Datenstruktur entsprechen verschiedene XML-Objekte des Jobs. Diese Objekte sind unterschiedlich in Anzahl und Typ der Parameter.

Jedes UNICORE-Element ist zwei Mal in der Datenstruktur enthalten. Zunächst als formale XML-Task in Matrix 1 und dann detailliert in einer der Matrizen 2-4 entsprechend seinem Typ. Auch ist es zweckmäßig, Daten über Verbindungen zwischen Job-Elementen und ihre besondere Eigenschaften zu trennen.

Es muß betont werden, daß die erste Struktur (Matrix 1: TASKS) keine Eigenschaften der UNICORE Objekte beschreibt. Aber sie ist als formales Element von XML notwendig.

Jedes Element hat seine eigene ID. Die Verbindungen zwischen Elementen bestimmen sich eindeutig durch die ”Parent ID”.

Die Matrizen 4 bis 6 sind dynamisch, weil sie von der Anzahl den entsprechenden Dateien abhängen.

Matrix1: **TASKS**

TASKS ID	PARENT ID	TASKS TYPE	TASKS NAME

Matrix2: **JOBS**

ID	Name	NameID	Usite	UsiteID	Vsite	VsiteID	Mail	MailID

Matrix3: **SCRIPTS**

ID	Name	NameID	Type	TypeID	Res	ResID	ImpID	ImpAnz	ExpID	ExpAnz

Matrix4: **TRANSFERS**

ID	Name	NameID	TransfAnz	Tr1ID	Name1	Tr2ID	Name2	. . .

Matrix5: **IMPORTS**

ID	Imp1ID	Name1	Name1ID	Dir1	Dir1ID	Imp2ID	Name2	Name2ID	Dir2	Dir2ID	...

Matrix6: **EXPORTS**

ID	Exp1ID	Name1	Name1ID	Dir1	Dir1ID	Exp2ID	Name2	Name2ID	Dir2	Dir2ID	...

Matrix7: **RESOURCES**

ID	Memory	MemoryID	Node	NodeID	Processor	ProcessorID	RTime	RTi meID

Abbildung 8: Datenstruktur für XML-Job Analyse.

Allgemeiner Algorithmus des Modifizierprogramms

Jetzt können wir detailiert den Algorithmus der Modifizierprogramms beschreiben:

- Einlesen der Input XML-Datei und Aufbau der Datenstrukturen (Abbildung 8). Auf dieser Stufe ist die Syntax-Analyse möglich.
- Formale Analyse des XML-Programms

  Analyse der Vollständigkeit der Elemente: jedes geöffnete Tag muß ein geschlossenes Tag haben.

  Analyse der richtigen Schachtelung von Tags: Tags, die in andere Tags geschachtelt sind, dürfen sich nicht überschneiden.

  Die Präsentation der Datenstruktur als ”Baum”.
- Änderungen des Benutzers durch das Menü-System.

  Der Benutzer kann jedes UNICORE-Element auswählen.

  Der Benutzer kann jeden Wert ändern.

  Speicherung der Veränderungsliste.

- Schreiben des neuen XML-Programms mit den veränderten Werten.

  Lese eine Zeile aus der Input Datei.

  Wenn diese Zeile in der Veränderungsliste steht, schreibe diese Zeile in die Outputdatei mit dem neuen Wert.

  Wenn diese Zeile nicht in das Veränderungsliste steht, schreibe diese Zeile unverändert in die Outputdatei.

  Solange das Ende der Inputdatei nicht erreicht ist, wiederhole die letzten drei Unterpunkte.

Ergebnisse

Das UNICORE-System soll automatisch getestet werden. Für ß-Tests gibt es in UNICORE eine Möglichkeit, alle Informationen in eine XML-Datei zu schreiben. Das XML-Format ist bequem für die Datenstruktur-Analyse.

Der Algorithmus des automatischen UNICORE-Tests auf Basis der XML-Dateien wurde vorgestellt.

Ein wichtiger Teil des Algorithmus ist ein Modifizierprogramm für in XML-Format geschriebene UNICORE Jobs. Das Programm für relativ einfache Modifizierungen wurde erstellt.

Der Algorithmus und entsprechende Datenstrukturen für den allgemeinen Fall wurde behandelt.

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die eine erfolgreiche Durchführung dieser Arbeit möglich gemacht haben. Mein erster Dank gilt Herrn Dr. R. Esser für die Möglichkeit, die Arbeit im Rahmen des Gaststudentenprogramms durchzuführen. Auch möchte ich Herrn D. Erwin danken. Insbesondere danke ich Frau M. Romberg für die Betreuung.

Literatur

1. M. Romberg, D. Erwin ”UNICORE - Uniformes Interface für Computer Ressourcen”, PIK 03/01, 2001.
2. S. North, P. Hermanns ”XML in 21 Tagen”, SAMS, 1999.
3. S. Graham, Liam Quin ”THE XML SPECIFICATION GUIDE”, WILEY, 1999.
4. M. Romberg ”The UNICORE Architecture. Seamless Access to Distributed Resources”.

ESCAPE: A profiling environment for parallel programs

Thorsten Gellermann

University of Paderborn

tgel@uni-paderborn.de

Abstract: ESCAPE (ESsential CAllgraph Profiling Environment) is a callgraph-based profiler for MPI, OpenMP and hybrid applications. ESCAPE is built on top of the PMPI, POMP, and PGI compiler's internal profiling interface to create an execution time profile for MPI functions, OpenMP parallel regions, and user functions. The profile can be visualized using the graph-layout program VCG.

Introduction

OpenMP has emerged as the standard for shared memory parallel programming, allowing users to write applications that are portable across most architectures. MPI is the de facto industry standard adopted by major commercial vendors used for programming distributed memory machines. However, in order to achieve high performance on SMP clusters, application developers still face a large number of application performance problems, such as load imbalance.

In the context of the guest student program at the Central Institute for Applied Mathematics, I developed a little profiling tool, called ESCAPE. It analyzes parallel applications running on clusters of shared memory systems using MPI, OpenMP and the Portland Group Compilers. ESCAPE consists primary of two parts. The library `libescape` for runtime data collection and the analyzing tool `read_prof` for the following data examination. To activate PGI profiling a special compiler flag is necessary and until POMP is not implemented as a standard profiling interface, the source must be preprocessed by OPARI [1]. First, the several profiling interfaces are presented. Later the basic approach and some implementation considerations are illustrated. Finally, I will show some examples of profiles.

Performance Analysis

We will break down the performance analysis process into three steps: data collection, data transformation and visualization.

Data collection is the step, where data about program performance are obtained from an executing program. There are two basic techniques of data collection:

- *Profiles* record the times spent on a each part of the program.
- *Event Traces* log each occurrence of specified events and in general produce a large amount of data.

It is usually not possible to make statements about program performance on raw data because of its complexity. Thus, data transformation is often used to outline performance problems by reducing data.

```
void main(){
  foo();
  bar();
}
void foo(){
  bar();
}
void bar(){};
```

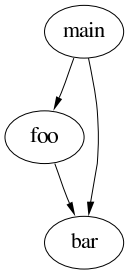


Figure 1: Example of Call Graph

For example, a profile of the time spent in each subroutine on each processor might be transformed to determine the mean time spent in each subroutine on each processor.

Although data reduction techniques can be used in many situations to compress performance data to meaningful values, this process can often benefit from the use of data visualization techniques.

Reasons for and against Profiling

Profiles have two important advantages. They can be obtained automatically, at relatively low cost, and they can provide a high-level view of program behavior that allows the programmer to identify problematic program components without generating huge amounts of data. In general, the amount of data associated with a profile is both small and independent of execution time. Therefore, a profile should be the first technique considered when seeking to understand the performance of a parallel program.

Profiles also have limitations. In particular, they do not incorporate temporal aspects of program execution.

Basic Terminology

Timing Expressions

The basic timing expressions are

- *Exclusive time:* The execution time of a given function not including the execution time of functions called by that function.
- *Inclusive time:* The execution time both of a given function and of any functions called by that function.
- *Exclusion time:* I will denote the time of any functions that were called by a given function exclusion time, although this term is not common.

Call Graph

Another expression is „call graph”. A call graph is a directed graph with a node for each function. An edge from function a to function b displays that a has called b. You can see an example in figure 1.

Profiling Interfaces

To minimize the programmer's intervention for instrumenting the application for performance analysis, we make use of the system-provided profiling interfaces, which make it possible to execute special profiling code.

The MPI Profiling Interface

MPI provides a profiling interface through which all of the MPI functions defined may be accessed with the prefix `PMPI_` instead of `MPI_`. To integrate profiling routines you have to write some wrapper functions with the same name measuring time before and after the call the related PMPI function and return the value returned by PMPI. For an example see below.

Correct linking is automatically done by `mpicc`. The MPI standard also ensures that those MPI functions which are not replaced may still be linked into an executable image without causing name clashes. Profiling can be controlled by the `MPI_PControl` function. Thus it is possible for the implementor of the profiling system to intercept all MPI calls which are made by the user program. For exact informations see [2].

The OpenMP Profiling Interface POMP

POMP [1] is a performance tools interface for OpenMP similar to the MPI profiling interface. It supports special function calls before and after every OpenMP construct. For example

```
#pragma OMP PARALLEL
  <structured block>
```

is replaced by

```
omperf_parallel_fork(d);
#pragma OMP PARALLEL
{
  omperf_parallel_begin(d);
  <structured block>
  omperf_parallel_end(d);
}
omperf_parallel_join(d);
```

with `d` as a descriptor containing a unique id, names and source code information. New OpenMP directives further allow profiling of user functions and arbitrary code regions. The directive transformation can be made by the tool OPARI [1].

The Profiling Interface of the Portland Group Compiler

The PGI compilers support a switch `-Mprof=func` which forces the compiler to execute specific pieces of code at the beginning and at the end of a function call. Special data structures are available, which contain information about the function call.

General Approach

Profiler Data

This profiler measures inclusive and exclusive time and counts the execution of OpenMP statements, communication and user functions, for each calling context.

Profiling of MPI Routines

MPI routines are measured by wrapper functions as explained below. Time is measured before and after the call and stored in the data structures.

As MPI functions can call each other, it has to be assured that they are not measured two times. For this reason, profiling for MPI routines is switched off during the execution of the PMPI routines.

```
int MPI_<function>( <parameters> )   {
  int thread, retval;
  double starting_time, stopping_time;
  thread = escape_get_thread_num();
  if(escape_mpi_profiled[thread]){
    escape_mpi_profiled[thread] = 0;
    starting_time = escape_get_time();
    retval = PMPI_Buffer_<function> ( <parameters> );
    stopping_time = escape_get_time();
    <store profiling data>
    escape_mpi_profiled[thread] = 1;
  }
  else{
     retval = PMPI_Buffer_<function> ( <parameters> );
  }
  return retval;
}
```

Profiling of OpenMP Constructs and User Functions

OpenMP statements or user functions can call other constructs, which have to be profiled. If a user function or an OpenMP construct is called, a profiling routine in `libescape` is entered and puts the starting time and a zero for the initial exclusions onto the stack.

```
void escape_enter(int thread, descriptor *d){
  struct escape_stack_element e;
  e.starting_time = escape_get_time();
  e.exclusions = 0;
  escape_push_stack(e);
}
```

At return of any of this constructs, another escape function is called and the current time is measured, the inclusive and exclusive time spent in the executed region is computed and the exclusion of the calling construct is updated.

```
void escape_leave(int thread, descriptor *d){
  double stopping_time;
  escape_stack_element e;
  e = pop_stack();
  stopping_time = escape_get_time();
  inclusive_time = stopping_time
      - starting_time;
  exclusive_time = stopping_time
      - starting_time - e.exclusions;
  increment_counter(d);
  <store information in data structures>
  <update exclusions on stack>
}
```

At program exit, all data is dumped on each host to a file named `escape_data.<MPI_Rank>` in clear text.

Data Transformation and Visualisation

Data transformation is done by a little program called `read_prof`. It can generate user defined statistics about function calls and other constructs. It is also possible to create a call graph for special graph layouters (`vcg` [4] and `dot`[5]). In this case, accumulated inclusive time is visualized by the nodes width and the accumulated exclusive time by the nodes height. `vcg` also supports special info tags which are used to provide additional information, qualified by the user on the command line. These include:

- `-t` Shows information separated by threads
- `-h` Shows information separated by hosts
- `-c` Shows information separated by the calling functions
- `-n` Determines the number of functions listed
- `-o` Sort output by summed exclusive times (se), accumulated inclusive times (si), maximum exclusive (me) or maximum inclusive (mi). You can specify the sort order as a parameter. For example `-o se` sorts the data by accumulated exclusive times
- `-g` Makes `read_prof` to generate a call graph in *vcg* format. This switch takes an output file as a parameter.
- `-d` Makes `read_prof` to generate a call graph in *dot* format. This switch takes an output file as a parameter.
- `-i` This switch is used for user-defined queries returning raw-data like informations. The first parameter of this switch denotes the attributes to select and the following the values to compare to. For example -i th 1 4 shows all information about thread 1 of host 4. -i ct 1 2 shows all information about functions called by function 1 of thread 4.
- `-a` This switch is similar to `-i`, but aggregates all data of a function.
- `-gt` Similar to `-i` switch but used for additional information in vcg info-tags.

Implementation Considerations

Initialization

First, the profiled program has to be linked with the library `libescape`. To support profiling of user functions the program should also be compiled with the switch mentioned above.

The way the main initialization routine `escape_init` of `libescape` is called depends on the profiling interfaces used. If the PGI interface is activated, everything is done automatically. If you use MPI without PGI, `escape_init` is called during the MPI wrapper function of `MPI_Init`. If you use only OpenMP without MPI and PGI or you use OpenMP constructs before `MPI_Init` you have to insert the special POMP instruction `#pragma pomp init` in C notation and `$POMP INST INIT` in Fortran syntax respectively.

It does not matter how often `escape_init` is called. The main issue is that it is called at a point before any profiling interface triggers any other routine of `libescape`.

All memory needed for profiling is allocated during initialization to reach a higher accuracy of timing informations, because allocation is expensive in time.

Notes on the Data Structures

At runtime, each thread has its own stack and its own elements in a table structure for storing the profiling results.

The table structure contains a reference to the construct's descriptor, provided by the profiling interface, and a reference to a table containing profiling information distinguished by the calling construct.

Time Measurement

There are two ways measuring time. The Real Time Clock (RTC) and the cycle counter (TSC). The main advantage of the real time clock is that it is easy to use and available in every system. In UNIX™you can get the RTC time via *gettimeofday*. The main disadvantage is, that it is slow and inaccurate.

The TSC provides a much better accuracy and speed evaluating the actual time. On the Intel's IA-32 architecture a cycle counter is available for all Pentium 2 and above processors. The TSC can be read by the `rdtsc` [6] instruction. If the processor speed is known the time can be computed by means of the clock speed.

ESCAPE provides both, TSC and RTC, defined via a switch during compilation of `libescape`.

Memory Management

Without analyzing the source code it is not possible to predict the memory consumption of a program. By this reason the the stack size, the number of profiled constructs and the quantity of functions calling a function can be set up in a file named `escape.rc`. If nothing is specified there, some default values are assumed.

The maximum number of used threads can be declared by the environment variable "`POMP_MAX_THREADS`".

```
   id name              subname              file                    from to
  fun. host thr. count(max/sum)  incl.(max/avg/sum)     excl.(max/avg/sum)

*   3 do                (none)               sweep.f                  355  507
  all  all  all 144000/1152000   1.4e+02/0.00092/1.1e+03  1.4e+02/0.00092/1.1e+03

*   8 $IDLE$            (none)               (none)                     0    0
  all  all  all      1/6         1.8e+02/1.8e+02/1.1e+03       48/    44/2.7e+02

*  10 MPI_Recv          (none)               (none)                     0    0
  all  all  all    576/1152           17/ 0.028/    32         17/ 0.028/    32

*   9 MPI_Send          (none)               (none)                     0    0
  all  all  all    576/1152           16/ 0.026/    30         16/ 0.026/    30

* 119 sweep             (none)               sweep.mod.F                2    0
  all  all  all     12/24        1.8e+02/    15/3.5e+02        13/     1/    25
```

Figure 2: `sweep3d` results

Thread Identification

The OpenMP call `omp_get_thread_num` is used to identify which thread you are actually inspecting. Unfortunately, if there is nested parallelisation there is no way to obtain a unique identification of a thread only using OpenMP and POMP instruments.

The only working technique to identify a thread is to use system calls. In the case of the PGI compilers, parallelisation is done with pthreads, so you can identify a thread easily by a *getpid* call, but this prevents ESCAPE to be ported to other OpenMP implementations, which do not use pthreads. You can however activate this mode at compile time of ESCAPE.

Examples

A profile of `sweep3d`

`sweep3d` [7] is a benchmark for parallel computers. In its readme file it is described that

> "it solves a 1-group time-independent discrete ordinates 3D cartesian geometry neutron transport problem. The XYZ geometry is represented by an IJK logically rectangular grid of cells. The angular dependence is handled by discrete angles with a spherical harmonics treatment for the scattering source. The solution involves two steps: the streaming operator is solved by sweeps for each angle and the scattering operator is solved iteratively."

`read_prof` *Standard Information*

Figure 2 shows the `read_prof` output of `sweep3d` executed on an SMP cluster consisting of two nodes each equipped with four Pentium III™ processors and two gigabyte RAM per node. The informations are sorted by accumulated exclusive times. It is easy to see that the OpenMP parallel do loop of file `sweep.f` from line 355 to 507 takes most time. It was cycled 1152000 times.

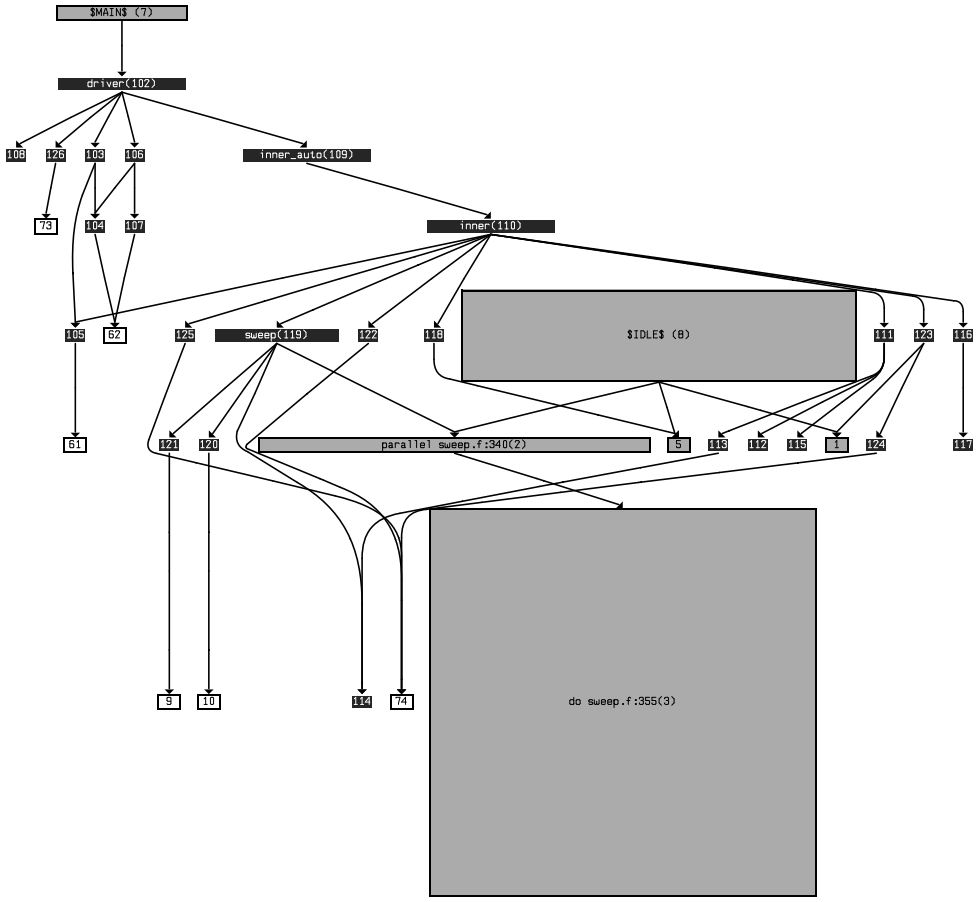


Figure 3: ESCAPE's call graph of `sweep3d`

`read_prof` *Call Graph visualized with* `vcg`

The call graph of `sweep3d` is shown in Figure 3. The different colors determine the construct types. If there is enough space, the complete name with code information is displayed. If not, only the name or only its id is displayed. Nodes have a minimum size for tge size that ensures that the id can be displayed.

Future Work

This software can easily be extended to generate more statistical information like traffic caused by MPI calls, better exploration of OpenMP idle states or evaluation of hardware counters. Of course, this comes along with even more memory consumption and a higher overhead.

References

1. Bernd Mohr and others: *Design and Prototype of a Performance Tool Interface for OpenMP.* Proceedings of the 2001 2nd Annual Los Alamos Computer Science Institute Symposium (LACSI 2001)
   also Technical Report FZJ-ZAM-IB-2001-09, Research Centre Jülich
2. Message Passing Interface Forum: *MPI: A Message-Passing Interface Standard*
   URL: `http://www.mpi-forum.org`
3. *OpenMP specifications:* URL: `www.openmp.org`
4. Georg Sander: *VCG: Visualisation of Compiler Graphs*
5. Eleftherios Koutsofios, Stephen C. North: *Drawing Graphs with dot.* AT&T® Bell Laboratories
6. Intel® Corporation: *IA-32 Intel Architecture Software Developer's Manual Volume 2: Instruction Set Reference.*
   URL: `http://developer.intel.com`
7. The Lawrence Livermore National Laboratory: *The ASCI sweep3d Benchmark Code.*
   URL: `http://www.llnl.gov/asci_benchmarks/asci/limited/sweep3d/asci_sweep3d.html`

Zellmotilität

Michael Klocke

Gerhard-Mercator-Universität Duisburg
Fakultät für Naturwissenschaften
Institut für Physik

M.Klocke@uni-duisburg.de

Zusammenfassung: Bewegungsfähige Zellen unterscheiden sich von anderen Zellen durch ihr Diffusionsverhalten. Der Persistent Random Walk beschreibt dieses Verhalten und soll von einem zweidimensionalen Zellmodell mit Hilfe von Gitter-Monte-Carlo-Simulationen nachgestellt und näher untersucht werden.

Einleitung

Bestimmte Zellen müssen sich aktiv bewegen können, um ihre spezifischen Aufgaben ausführen zu können, so z.B. müssen Neutrophile (Leukozyten) in der Lage sein, um Bakterien herumzufließen, um sie anschließend zu zerstören. Die Bewegungsfähigkeit wird als Motilität bezeichnet und ist durch eine aktive, vom Organismus selbst ausgehende Bewegung gekennzeichnet. Die Bewegung kann durch äußere Einflüsse induziert sein, in diesem Fall spricht man von *Taxis* [4]. Je nach Art des Stimulus unterscheidet man z.B. zwischen Thigmotaxis bei Berührung, Thermotaxis bei Wärme oder Chemotaxis bei chemischen Stoffen. Ohne entsprechende Stimuli vollführt die Zelle einen *Persistent Random Walk*. Dabei läuft die Zelle eine gewisse Zeit lang in eine bestimmte Richtung und ändert dann ihre Richtung zufällig (siehe Abb. 1). Das Verhalten unterscheidet sich deutlich von dem eines Brownschen Partikels [3].

Ziel dieses Projektes ist es, ein zweidimensionales Zellmodell zu entwickeln, welches den Effekt des Persistent Random Walk zeigt.

Biologische Grundlagen zur Zellbewegung

Zellbewegung wurde an vielen unterschiedlichen Zellen untersucht, klassische Beispiele sind Keratozyten (Epithel-Zellen, u.a. zur Wund-Heilung), Fibroblasten (Hautzellen, Aufbau von Bindegewebe) [22], oder Amöben (bisweilen wird die Zellbewegung daher auch amöboide Bewegung genannt) [23]. Es zeigen sich ganz unterschiedliche Eigenschaften in jeder dieser Zellen und es ist nicht auszuschließen, dass ganz unterschiedliche Mechanismen existieren [24]. Ich möchte in diesem Projekt nur auf die Bewegung von Keratozyten eingehen, da dabei auch Einigkeit über die Grundlagen der Zellbewegung herrscht.

Keratozyten sind mit $30\ \mu\mathrm{m}/\mathrm{min}$ relativ schnelle Zellen [5]. Man untersucht die Bewegung üblicherweise nur in zwei Dimensionen, z.B. auf einer beschichteten Glasplatte. Ein noch einfacheres System bilden zytoplasmische Fragmente von Zellen, z.B. von Keratozyten. Das sind Zellen, bei denen der Zellkern und die meisten Organellen fehlen [6]. Es zeigt sich, dass diese Fragmente immer noch bewegungsfähig sind. Bei der Bewegung ändert sich die Form der Zelle in eine halbmondförmige Gestalt, während die Zelle ohne Bewegung kugelsymmetrisch ist (siehe Bild 2) [7]. In Bewegungsrichtung bildet sich ein

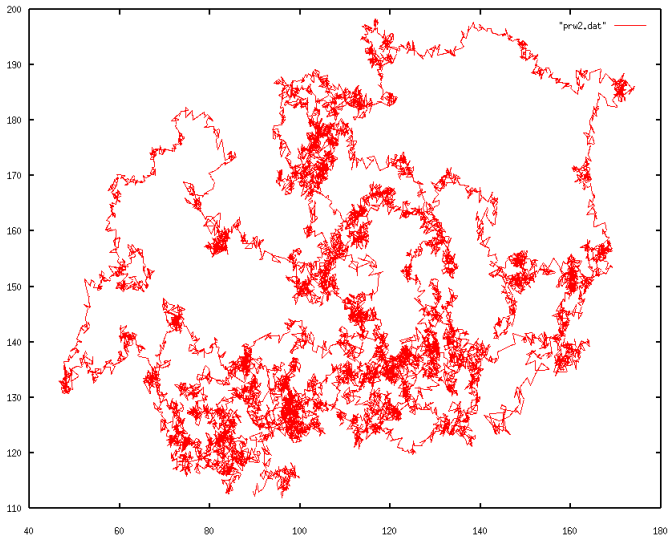


Abbildung 1: Trajektorie eines Persistent Random Walk. Man erkennt längere gleichgerichtete Phasen. Über lange Zeitskalen ergibt sich aber wieder das Bild eines Random Walk.

breiter, sehr flacher Vorstoß der Zelle aus. Diese Formation nennt man *Lamellipodium.* Die Vorderkante (in Bewegungsrichtung) nennt man auch *Leading Edge*.

Für die Bewegungsfähigkeit werden grundlegende Eigenschaften des Zytoskeletts verantwortlich gemacht [8]. Das Zytoskelett besteht aus drei Sorten von festen Strukturen, den Mikrotubuli, den intermediären Filamenten und den Mikrofilamenten. Diese Unterscheidung ergibt sich durch die typischen Durchmesser der Filamente von $25 - 30\ \mathrm{nm}$ bei Mikrotubuli bis zu $4 - 5\ \mathrm{nm}$ bei Mikrofilamenten. Mikrofilamente sind die Hauptkomponenten der Zellbewegung. Sie bestehen aus Actin, einem ca. $55\ \mathrm{kD}$ schweren Protein [14]. Von Actin ist bekannt, dass es in Verbindung mit Myosin für Muskelkontraktion sorgt [4]. Bei der Zellbewegung kommt dieser Mechanismus allerdings nicht zum Tragen. Zwar ist auch Myosin in der Zelle vorhanden, die genaue Bedeutung ist allerdings weitgehend unverstanden. Die Mikrotubuli sind für die Zellbewegung nicht notwendig, dennoch fallen ihnen bestimmte Aufgaben zu [10]. Die Intermediären Filamente scheinen allerdings keine Rolle zu spielen [11].

Actin bildet Filamente unter Polymerisation aus. Man spricht bei einzelnen Actin-Molekülen daher von Monomeren oder auch von G-Actinen. Die Filamente sind geradlinig und erscheinen als gewundene

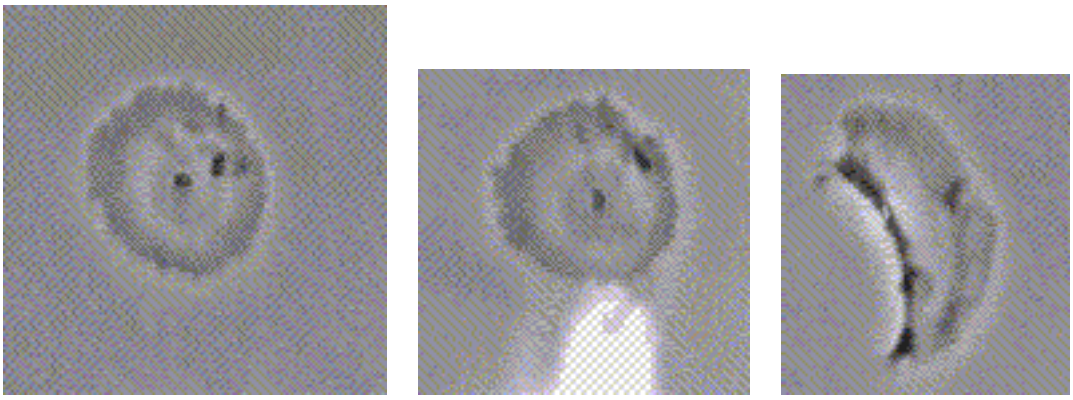


Abbildung 2: Aufnahmen eines Keratozyten-Fragments. (a) Ohne Bewegung zeigt die Zelle eine kugelsymmetrische Form. (b) Bewegung wird durch mechanische Reizung initiiert. (c) Die Zelle bewegt sich nach links oben. Die Gestalt ist nun halbmondförmig. (Aufnahmen von A.B. Verkhovsky, T.M. Svitkina und G.G. Borisy) [7]

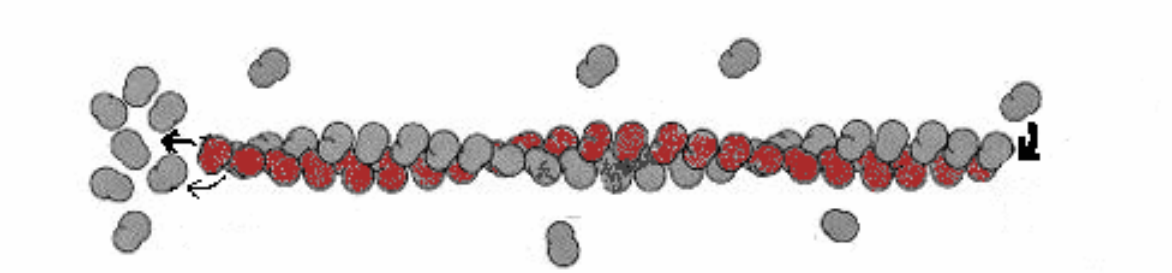


Abbildung 3: Der Treadmilling-Effekt. Am Minus-Ende werden vornehmlich Monomere depolymerisiert. Diese diffundieren dann durch die Zelle. Am Plus-Ende werden Monomere polymersiert. Damit bewegt sich das Filament nach rechts.

Doppelstränge (siehe Abb. 3) [15]. Eine wesentliche Eigenschaft dieser Filamente ist, dass sie polar sind, d.h. es gibt unterschiedliche Polymerisierungs- und Depolymerisierungsraten an beiden Enden [12]. Aufgrund ihres Erscheinungsbildes unter dem Mikroskop wird das Ende, an dem bevorzugt depolymerisiert wird, das *Pointed End* oder auch *Minus-Ende* genannt, das andere Ende wird als *Barbed End* oder auch *Plus-Ende* bezeichnet.

Die Konzentration $[M]$ von G-Actinen läßt sich durch die Differentialgleichung

$$\frac{d[M]}{dt} = -k^+ \cdot [M] + k^- \tag{1}$$

beschreiben, wobei k^+ und k^- die Polymerisierungs- bzw. Depolymerisierungsraten sind. Es existiert eine kritische Konzentration $[M]_C$ bei der Stationarität herrscht. Diese kritische Konzentration muss an beiden Enden des Filaments gleich sein, ansonsten ergäbe sich für eine Konzentration $[M]_C^B < [M] < [M]_C^P$, wobei B und P für jeweils Barbed End und Pointed End stehen, ein Fluss von Monomeren vom Pointed zum Barbed End. Da wir aber Stationarität voraussetzen, d.h. wir uns im thermodynamischen Gleichgewicht befinden, dürfen nach dem zweiten Hauptsatz keinerlei Ströme mehr auftreten [16].

Die Situation ändert sich allerdings, wenn dem System zusätzlich Energie hinzugefügt wird. Der in der Biologie üblich Energieträger ist das ATP (Adenosin-Tri-Phosphat), das nach der Reaktion

$$\text{ATP} \longrightarrow \text{ADP} + \text{P}^{(Ion)} + 7\ \text{kcal/mol}$$

in ADP (Adenosin-Di-Phosphat) zerfällt [4]. Sowohl ATP als auch ADP binden an Actin, es ergeben sich aber andere Polymerisierungs- und Depolymerisierungsraten. Damit ergibt sich folgendes Bild: An einem bestehenden Filament bindet sich ein mit ATP gebundenes G-Actin an das Barbed End. Durch weiteren Anbau vorne und Abbau hinten wandert es allmählich vom Barbed End zum Pointed End (im Bezugssystem des Filaments). Es hydrolisiert das ATP zu ADP. Am Pointed End wird es schließlich vom Filament getrennt und diffundiert wieder als freies Monomer. Während dieser Diffusion wird ADP wieder durch ATP ersetzt. Das Monomer kann zum Barbed End diffundieren und der Prozess beginnt erneut. Man nennt diesen Mechanismus auch Treadmilling-Effekt. (siehe Abb.3) [20].

Die Zellbewegung basiert auf diesem Effekt. Die Filamente sind über bestimmte Protein-Komplexe relativ fest an das unterliegende Substrat oder die Extrazelluläre Matrix gebunden [9]. Die Actin-Filamente wachsen bis zur Membran und können von ihr nicht ins Innere der Zelle zurückgeschoben werden. Die wachsenden Filamente bilden einen Vorstoß der Membran aus, das Lamellipodium. Das gesamte Actin-Netzwerk bleibt relativ zum Substrat in Ruhe. Dies läßt sich auch experimentell bestätigen [13]. Am Trailing Edge (Hinteres Ende) wird das Netzwerk abgebaut und die Zelle wird durch die Spannung der Membran “nachgezogen”. Die Zelle kann sich also allein durch die Bildung der Actin-Filamente bewegen [8].

Es gibt viele Möglichkeiten, den Treadmilling-Effekt zu steuern. Zum einen hängen die Polymerisierungs- und Depolymerisierungsraten von der Konzentration anderer Stoffe ab, z.B. Ca^{2+} oder Mg^{2+} [17]. Wei-

terhin gibt es eine ganze Menge weiterer Proteine, die an der Polymerisation von Actin beteiligt sind [11] und dabei unterschiedliche Einflüsse ausüben können, z.B. kann die Polymerisierung für ein einzelnes Monomer verhindert werden. Als besonders wichtig erscheint der Arp2/3-Komplex. Dieser hat gleich mehrere wichtige Aufgaben, er sorgt für die Nukleation neuer Filamente, der schützt die Pointed Ends vor Depolymerisation (Capping), und es ermöglicht die Verzweigung der ansonsten geradlinigen Filamente. Diese Verzweigungen bilden einen Winkel von ca. $70\ ^\circ$. Dadurch stehen mehr Barbed Ends zur Verfügung und die Filamente können wesentlich schneller wachsen, was die Zellbewegung beschleunigt. Der Arp2/3 Komplex muß zunächst "aktiviert" werden, bevor er auch an G-Actine binden kann. Dies geschieht an den an der Membran sitzenden WASP/Scar-Komplexen. Sie stellen ein wichtiges Zwischenstück auf dem Signalweg zwischen externem Stimulus und Umsetzung in Zellbewegung dar [21].

Modellierung

Zunächst müssen einige Modellannahmen eingeführt werden, die sich aus den experimentellen Ergebnissen rechtfertigen lassen. Wir können nicht die gesamte Zelle simulieren, einmal, da viele beteiligte Proteine in ihrer Struktur unbekannt sind, zweitens, die Simulation eines derartig großen Systems über Zeitskalen im Minutenbereich ist auch in ferner Zukunft nicht möglich. Daher reduzieren wir die Zelle auf die Schlüssel-Elemente, die zur Zellbewegung notwendig sind.

Die Motilität von zytoplasmischen Fragmenten zeigt bereits eine wichtige Reduktion auf: wir müssen nicht die gesamte Zelle mit Zellkern und Organellen simulieren. In einem Fragment gibt nur noch vier Inhalte: das Zytoskelett, frei diffundierende Proteine und andere Substanzen, die Membran und integrale Membran-Proteine. Betrachtet man das Zytoskelett, so wurde oben bereits erwähnt, dass der Einfluss der Mikrotubuli sekundär ist. Da die Zellbewegung auch ohne diese funktioniert, werden sie in unserem Modell weggelassen. Da die Intermediären Filamente keinen nachweisbaren Einfluss auf die Zellbewegung hat, werden auch sie nicht im Modell aufgenommen.

Es wurde bereits erwähnt, dass Ionen wie Calcium oder Magnesium die Polymerisierungsraten von Actin ändern. Wir nehmen für unser Modell eine homogene, zeitlich konstante Verteilung solcher Ionen an, so dass der Einfluss implizit in den Ratenkonstanten steckt. Insbesondere werden damit keine lokalen Konzentrationsschwankungen berücksichtigt, so dass sich auch keine Chemotaxis aufgrund von Ionenkonzentrationsänderungen simulieren läßt. Damit besteht auch keine Notwendigkeit, z.B. Ionenkanäle (integrale Membran-Proteine, die Ionen über die Membran hinweg transportieren) zu berücksichtigen. Unter allen weiteren Proteinen, die sich frei in der Zelle diffundieren, berücksichtigt unser Modell lediglich das Arp2/3. Die Verzweigung von Filamenten ist eine wesentliche Eigenschaft des Mechanismus und wird offenbar nur von diesem Protein-Komplex ermöglicht. Der für die Kopplung der Filamente an das Substrat verantwortliche Protein-Komplex wird implizit angenommen, indem die Filamente direkt über die Membran hinweg an das Substrat gebunden werden. Außer dem G-Actin, das fundamental für die Motilität ist, werden alle weiteren Proteine nicht berücksichtigt. Auch werden keine weiteren integralen Membran-Proteine berücksichtigt.

Simulation

Zur Simulation lassen sich zwei Methoden einsetzen die jeweils ihre Vor- und Nachteile haben. Die Brownsche Dynamik Simulation (BD) stellt eine Möglichkeit dar. Dabei wird zunächst eine dem Problem angemessene Langevin-Gleichung der Art

$$\dot{\vec{r}}_i = \text{const} \frac{\partial}{\partial \vec{r}_i} U(\{\vec{r}\}) + \Gamma_i(t) \tag{2}$$

wobei $\vec{r}_i$ die Ortsvektoren der einzelnen Partikel, $U((\{\vec{r}\})$ entsprechende Potentiale zwischen den Par-

tikeln und $\Gamma_i(t)$ einen Rauschterm darstellt. Die Simulation dieser stochastischen Differentialgleichung benötigt allerdings viel Rechenzeit, außerdem ist die Wahl der Potentiale numerisch nicht trivial [26]. Da wir diese Methode in diesem Projekt nicht benutzt haben, möchte ich auch nicht weiter darauf eingehen.

Die zweite Möglichkeit ist die Monte-Carlo-Methode. Es läßt sich zeigen, dass beide Methoden (unter bestimmten Voraussetzungen) zueinander äquivalent sind [18, 19]. Eine Simulation mit der Monte-Carlo-Methode besteht aus der Auswahl eines Weges durch den Konfigurationsraum des Systems. Der Weg besteht aus einer Folge von Konfigurationen $K_1, K_2, \ldots$, wobei diese Folge i.d.R. als Markov-Prozeß konstruiert wird, d.h. die Konfiguration K_i hängt lediglich vom Vorgänger K_{i-1} ab. Als wichtige Bedingung für einen MC-Algorithmus gilt die *detailed balance*-Bedingung:

$$\frac{W(K \to K')}{W(K' \to K)} = \exp(-\beta E(K') - E(K)) \tag{3}$$

Dabei ist $W(K' \to K)$ die Übergangswahrscheinlichkeit von Konfiguration K' nach Konfiguration K, β die inverse Temperatur und $E(K')$ die Gesamtenergie für die Konfiguration K'. Diese Bedingung ist notwendig für die Beschreibung eines Gleichgewichtszustandes. Ein möglicher Algorithmus, der diese Bedingung erfüllt, ist der Metropolis-Algorithmus: In einem System mit N Teilchen und der Konfiguration $X = (x_1, \ldots, x_n)$ wird zufällig ein Teilchen j ausgewählt. Dieses Teilchen wird um $dx \propto \eta$ verschoben, wobei η eine gleichverteilte Zufallszahl zwischen -1 und 1 ist. Man betrachtet dann zunächst eine Testkonfiguration $X_t = (x_1, \ldots, x_j + dx, \ldots, x_N)$. Von dieser Konfiguration berechnet man nun die Gesamtenergie des Systems. Die detailed balance Bedingung läßt sich erfüllen, wenn die Testkonfiguration nur dann sofort akzeptiert wird, wenn die Energie kleiner geworden ist, d.h. $E(X_t) \leq E(X)$. Ansonsten wird die Änderung nur mit einer Wahrscheinlichkeit μ mit

$$\mu = \exp(-\beta(E(X_t) - E(X)) \tag{4}$$

akzeptiert. Man spricht bei N Versuchen, die Konfiguration zu ändern, von einem Monte-Carlo-Schritt [27, 28].

Auf einem Gitter wird dieser Algorithmus wesentlich vereinfacht. Nehmen wir zwischen den Teilchen ein Kastenpotential an (Excluded Volume Interaction). Der Algorithmus sieht dann so aus:

1. Wähle ein Teilchen j aus.
2. Wähle zufällig eine Richtung aus und setze das Teilchen auf nächsten Gitterplatz.
3. Ist bereits ein weiteres Teilchen auf diesem Gitterplatz: Dann ist die Energie der Testkonfiguration divergent, damit wird nach (4) $\mu = 0$, d.h. die Änderung wird ohne Berechnung einer Wahrscheinlichkeit (also aufwendiges Berechnen einer Zufallszahl) verworfen.
4. Ist kein Teilchen auf dem Gitterplatz, ändert sich die Gesamtenergie nicht. In diesem Fall ist wieder nach (4) $\mu = 1$, und die Änderung wird wieder ohne Berechnung einer Wahrscheinlichkeit akzeptiert.

Dieses Verfahren ist auf dem Gitter also sehr effizient.

Die Membran läßt sich in zwei Dimensionen als ein geschlossenes Band betrachten. Auf einem Gitter läßt sich dieses Band als eine Reihe von miteinander verbundenen Teilchen ansehen, zwischen denen ein stark attraktives Potential existiert. Während der obige Algorithmus garantiert, dass keine zwei Teilchen auf einem Gitterplatz existieren, muss bei den Membran-Teilchen auch darauf geachtet werden, dass sie sich nicht zu weit voneinander entfernen. Nimmt man wiederum ein Kastenpotential an, nun aber so, dass

die Energie divergent für Abstände zwischen zwei benachbarten Teilchen größer als eine Gitterkonstante ist, so ergibt sich im wesentlichen der gleiche Algorithmus wie oben. Die Membran-Teilchen können aber im Gegensatz zu den anderen Teilchen auch auf dem gleichen Gitterplatz liegen. Da die Membran undurchdringbar sein soll, wird für die Wechselwirkung Membran-Teilchen–Actin ein Kastenpotential angenommen. Solange die Membran-Teilchen auf Abstände von einer Gitterkonstante beschränkt sind, können keine Actine aus der Zelle entweichen [29].

Ein Problem ergibt sich aber mit der Wahl des Gitters. Die Gestalt des Gitters überträgt sich in gewisser Weise auf die Gestalt der Membran. So führt ein qudratisches Gitter auch zu eher quadratischen Membran-Formen. Zellen haben aber üblicherweise keine solche Form. Eine bessere Wahl ist das Dreiecks-Gitter [25]. Hier ergeben sich durch die höhere Symmetrie auch natürlichere Formen. Ein weiterer Vorteil liegt in den Filament-Verzweigungen, da sich der experimentell ermittelte Winkel von $70\ ^\circ$ auf einem Dreieck besser annähern läßt.

Die Filamente werden nicht als Doppelstrang betrachtet, sondern als einfache lineare Kette. Das Haften an das Substrat läßt sich damit berücksichtigen, dass die Filamente keine Bewegung ausführen können. Wir nehmen also eine extrem starke Bindung der Filamente an das Substrat (bzw. über die entsprechenden Protein-Komplexe) an. Die Depolymerisation erfolgt nach festgelegten Raten. Dabei werden für beide Enden jeweils getrennte Raten angegeben. Die Polymerisation wird ebenfalls über Raten gesteuert, allerdings erfolgt nur dann eine Polymerisation, wenn auch ein freies G-Actin höchstens einen Gitterplatz von dem aktuellen Ende entfernt liegt. Wenn nötig, wird das Teilchen dann noch auf den richtigen Gitterplatz verschoben, so dass sich auch ein geradliniges Filament ergibt. Zwischen den Filamenten und den G-Actinen besteht keine Exluded Volume Interaction, d.h. die G-Actine können über die Filamente hinweg diffundieren. Dies erweist sich als notwendig, da ansonsten der Treadmilling-Effekt bei hoher Filament-Dichte nicht eintreten kann, weil der “Nachschub” von freien Monomeren ausbleibt. Die Nukleation neuer Filamente geschieht mit einer gewissen Wahrscheinlichkeit spontan, d.h. liegen zwei freie G-Actine nur einen Gitterplatz voneinander entfernt, kann sich daraus ein neues Filament ergeben. Barbed End und Pointed End werden dabei zufällig gesetzt.

Auch Arp2/3 wird nach dem gleichen Algorithmus simuliert, wie bereits die freien G-Actine. Allerdings ist der Komplex wesentlich schwerer als die G-Actine und diffundiert daher auch langsamer. Das Arp2/3 hat einen aktivierten und einen deaktivierten Zustand. Im deaktivierten Zustand diffundiert es in der Zelle herum, ohne eine Wechselwirkung mit anderen Teilchen einzugehen. Ist es aktiviert, so bindet es, falls vorhanden, an ein G-Actin. Dieser Verbund diffundiert gemeinsam weiter, bis es entweder ein anderes freies G-Actin trifft, um ein neues Filament zu bilden, oder auf ein bereits bestehendes Filament trifft, und dort eine Verzweigung aufbaut. Mit Bindung an ein G-Actin wird das Arp2/3 wieder deaktiviert. Die Aktivierung erfolgt an der Membran. Dem liegt die Vorstellung zugrunde, dass die Aktivierung an dem WASP/Scar-Komplex stattfindet, welches an die Membran gebunden ist.

Insgesamt gehen bereits jetzt viele Parameter in dieses Modell ein:

- Vier Raten: Polymersierungs- und Depolymerisierungsraten für Barbed und Pointed End
- Dichte von freien G-Actinen und Dichte von Arp2/3

Eine Erweiterung des Modells berücksichtigt zusätzlich die Bindung von Actinen an ATP oder ADP. Ist ein Actin polymerisiert, wird mit bestimmter Wahrscheinlichkeit das ATP in ADP umgewandelt. Ein mit ADP gebundenes Actin hat eine höhere Wahrscheinlichkeit, depolymerisiert zu werden. Bei der Depolymerisierung wird das ADP wieder durch ATP ersetzt.

Dieses Modell eignet sich für Untersuchungen allerdings weniger gut, da nun bereits 11 freie Parameter vorliegen: Neben den sechs oben genannten Parametern kommen nochmal vier Polymerisierungs-, bzw.

Depolymersierungsraten (für ADP-gebundene F-Actine) hinzu. Zusätzlich kommt noch die Zerfallsrate von ATP nach ADP hinzu.

Eine weitere Modellannahme sieht vor, Pointed Ends, auf die die Membran stößt, sofort zu depolymerisieren. Die Idee, die hinter dieser Annahme steht, ist die experimentell gefundene Tatsache, dass die Filamente zur Membran hin ausgerichtet sind [30].

Resultate

Die Wahl der Parameter ist eine schwierige Angelegenheit. Nur in bestimmten Bereichen läßt sich auch der Treadmilling-Effekt beobachten. Sind z.B. die Depolymerisationsraten zu groß gewählt, so werden die Filamente innerhalb kürzester Zeit wieder aufgelöst, und es ergibt sich kein Netzwerk von Actin-Filamenten. Ist die Depolymerisationsrate zu klein gewählt, so bleiben die Filamente bestehen, und die Zelle kann sich aufgrund der Bindung an den Untergrund nicht bewegen. Durch eine visualisierte Animation der Simulation lassen sich diese extremen Abweichungen relativ leicht erkennen.

Dennoch bleibt, auch durch die relativ hohe Anzahl an freien Parametern, ein gewisser Spielraum. Ziel ist es, eine geeignete Kombination von Parametern zu finden, so dass die Simulation den Persistent Random Walk zeigt. Neben der Trajektorie ist das Mean Square Displacement aussagekräftig. Im Falle der Persistenz wächst es quadratisch mit der Zeit an. Die Abbildung 4 zeigt das Mean Square Displacement geteilt durch die Zeit aufgetragen über die Zeit an. Man erkennt für mittlere Zeiten einen linearen Verlauf. Dies ist der Persistenz-Bereich, in welchem die Zelle in eine Richtung läuft. Für große Zeiten geht das Verhalten wieder in ein normales Random Walk-Verhalten über. Die Schwankungen entstehen durch Fehler in der Mittelung.

Unser hier entwickeltes zweidimensionales Zellmodell zeigt also tatsächlich das Verhalten des Persistent Random Walk. Es bleibt die Frage nach der Abhängigkeit der Länge des persistenten Bereichs von den Parametern. Eine systematische Untersuchung gestaltet sich aufgrund des großen Parametersatzes als schwierig.

Danksagung

Ich danke meinem Betreuer, Herrn Dr. A. Baumgärtner, für das Angebot, an diesem interessanten Thema zu arbeiten, und für die große Unterstützung.

Literatur

1. R. Sambeth *Theoretical Studies of Biological Cell Motility* Dissertation, Univ. Duisburg, 2001
2. R. Sambeth, A. Baumgärtner Autocatalytic Polymerisation Generates Persistent Random Walk of Crawling Cells *Phys.Rev.Lett.*, 86 (2001) 5196
3. S. J. Singer and A. Kupfer. The directed migration of eukaryotic cells. *Ann. Rev. Cell Biol.*, 2:337–365, 1986.
4. N. A. Campbell *Biologie* Spektrum Verlag, 1997
5. K. I. Anderson and R. Cross. Contact dynamics during keratocyte motility. *Curr. Biol.*, 10:253–260, 2000.
6. H. U. Keller. Migration and chemotaxis of anucleated cytoplasmic leukocyte fragments. *Nature*, 258:723–724, 1975.
7. A. B. Verkhovsky, T. M. Svitkina, and G. G. Borisy. Self-polarization and directional motility of cytoplasm. *Curr. Biol.*, 9:11–20, 1999.
8. G. G. Borisy and T. M. Svitkina. Actin machinery: pushing the envelope. *Curr. Opin. Cell Biol.*, 12:104–112, 2000.
9. D. Bray. *Cell movements.* Garland, New York, 1992.
10. U. Euteneuer and M. Schliwa. Persistent, directional motility of cells and cytoplasmic fragments in the absence of microtubules. *Nature*, 310:58–61, 1984.
11. F. Richelme, A.-M. Benoliel, and P. Bongrand. The leukocyte actin cytoskeleton. *Bull. Inst. Pasteur*, 94:257–284, 1996.
12. D. Sept, A. H. Elcock, and A. McCammon. Computer simulations of actin polymerization can explain the barbed-pointed end asymmetry. *J. Mol. Biol.*, 294:1181–1189, 1999.
13. J. A. Theriot and T. J. Mitchison. Actin microfilament dynamics in locomoting cells. *Nature*, 352:126–131, 1991.

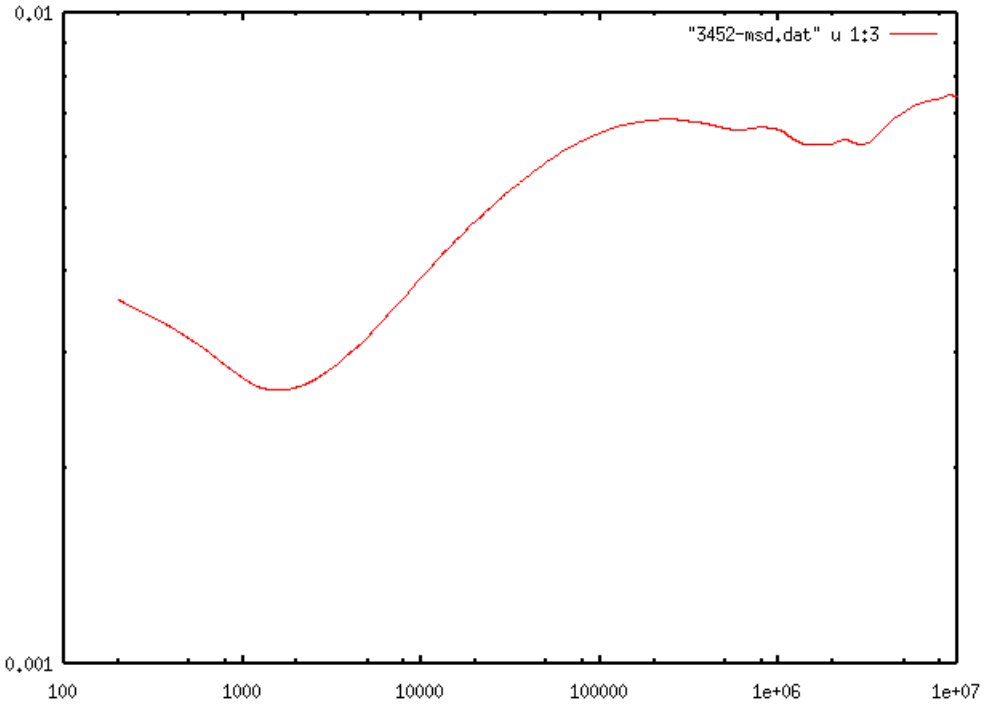


Abbildung 4: Mean Square Displacement geteilt durch Zeit über die Zeit aufgetragen. Für kleine Zeiten wird die Zelle wegen der Bindung der Filamente an das Substrat unbeweglicher. Für mittlere Zeiten läuft die Zelle vornehmlich in eine Richtung, siehe auch Trajektorie (Abb.1). Für große Zeiten ergibt sich wieder der Random Walk.

14. M. O. Steinmetz, D. Stoffler, A. Hoenger, A. Bremer, and U. Aebli. Actin: From cell biology to atomic detail. *J. Struct. Biol.*, 119:295–320, 1997.
15. J. Käs, H. Strey, J. X. Tang, D. Finger, R. Ezzell, E. Sackmann, and P. A. Janmey. F-actin, a model polymer for semiflexible chains in dilute, semidilute, and liquid crystalline solutions. *Biophys. J.*, 70:609–625, 1996.
16. F. Oosawa and S. Asakura. *Thermodynamics of the Polzmerization of Protein*. Academic Press, London, 1975.
17. M.-F. Carlier. Actin: Protein structure and filament dynamics. *J. Biol. Chem.*, 266:1–4, 1991.
18. N. G. van Kampen. *Stochastic processes in physics and chemistry*. North-Holland, Amsterdam, 1992.
19. K. Kikuchi, M. Yoshida, T. Maekawa, and H. Watanabe. Metropolis monte carlo method as a numerical technique to solve the Fokker-Planck equation. *Chem. Phys. Lett.*, 185(3,4):335–338, 1991.
20. M. Wanger, T. Keiser, J. M. Neuhaus, and A. Wegner. The actin treadmill. *Can. J. Biochem. Cell Biol.*, 63:414–421, 1985.
21. K. J. Amann and T. D. Pollard. The Arp2/3 complex nucleates actin filament branches from the sides of pre-existing filaments. *Nature Cell Biol.*, 3:306 – 310, 2001.
22. M. Dembo and Y.-L. Wang. Stresses at the cell-to-substrate interface during locomotion of fibroblasts. *Biophys. J.*, 76:2307–2316, 1999.
23. L. Grebecka, P. Pomorski, A. Grebecka, and L. Lopatowska. Adhesion-dependent F-actin pattern in amoeba proteus as a common feature of amoebae and the metazoan motile cells. *Cell Biol. Int.*, 21:565–573, 1997.
24. Cell Motility and Mechanics *http://zk.bwh.harvard.edu/projects/motility/index.html* Cell Biology and Cytoskeleton Group Division of Hematology, Brigham Women's Hospital, Harvard Medical School
25. J. Reiter. Monte carlo study of diffusion of an ideal ring polymer in a network of obstacles on a cubic and square lattice. *J. Chem. Phys.*, 95(2):1290–1294, 1991.
26. W. F. van Gunsteren and H. J. C. Berendsen. Algorithms for brownian dynamics. *Molec. Phys.*, 45(3):637, 1982.
27. D. W. Heermann. *Computer Simulation Methods in Theoretical Physics*. Springer-Verlag, Berlin, 1990.
28. J. Schnakenberg. *Algorithmen in der Quantentheorie und Statistischen Physik*. Zimmermann-Neufang, Ulmen, 1995.
29. K. Binder and W. Paul. Monte carlo simulations of polymer dynamics: Recent advances. *J. Polym. Sci.*, 35:1–31, 1997.
30. Y. L. Wang. Exchange of actin subunits at the leading edge of living fibroblasts: possible role of treadmilling. *J. Cell Biol.*, 101:597–602, 1985.

Regionenbeschreibende Formmerkmale zur Analyse segmentierter 3D-Regionen des Gehirns

Stefan Kunis[1], Gudrun Wagenknecht[2]

[1]Medizinische Universität zu Lübeck
Fachbereich Informatik
[2]Forschungszentrum Jülich
Zentrallabor für Elektronik

kunis@informatik.mu-luebeck.de

Einleitung

Seit der Entwicklung von bildgebenden Verfahren (z. B. MRT), die 3D-Bilddaten von Teilen des menschlichen Körpers erzeugen, ist man an Techniken interessiert, mit denen die gewonnenen Daten auf hohem Niveau verarbeitet/analysiert werden können. Hier sollen nun Methoden untersucht werden, die Regionen in bereits segmentierten Volumendaten des menschlichen Gehirns durch aussagekräftige Merkmale beschreiben. Mögliche Herangehensweisen sind:

1. Normalisiere (nichtlinear) die gewonnenen Daten auf ein Standardmodell und berechne dann gewisse Merkmale (z. B. probability maps).

2. Berechne Merkmale, die unter Ähnlichkeitstransformationen (Translation, Skalierung, Rotation und eventuell Spiegelung) invariant sind.

Im folgenden soll die zweite Idee verfolgt werden. Für den 2D-Fall gibt es hierzu Ansätze, da es beispielsweise in Computer-Vision-Anwendungen darum geht, Objekte mit Hilfe von Merkmalen zu erkennen. Drei weitverbreitete Methoden dort sind: Momentinvarianten, Morphometrie, Fourierdeskriptoren.

Momentinvarianten

Momentinvarianten traten Anfang der sechziger Jahre [8] in das Blickfeld der Mustererkennung, Anwendung fanden sie beispielsweise in der Flugzeugidentifikation [3] und der Buchstabenerkennung. Momente lassen sich in beliebigen Dimensionen definieren. In welcher Art Merkmale aus 3D-Bilddaten zu extrahieren sind, soll im folgenden untersucht werden.

Morphometrie

Morphologische Methoden spielen in der Binärbildverarbeitung ganz allgemein eine Rolle bei der Beschreibung der Form von Objekten. Weiterhin läßt sich auch ein Maß ableiten, welches die Komplexität und die Größenverteilung eines Objektes mißt. Dieses soll hinsichtlich seiner Eignung als Merkmal untersucht werden.

Fourierdeskriptoren

Fourierdeskriptoren (FD) fassen den Rand eines Objektes (implizit: keine Einschlüsse) als Funktion $f : [0, 2\pi] \to \mathbb{C}$ mit $f(t) = x(t) + iy(t)$ (t Zeit für "Durchlauf") auf. Diese läßt sich dann in eine Fourierreihe entwickeln. Die Fourierköffizienten lassen sich sehr einfach gegen Skalierung, Translation, Rotation und den "Startpunkt" der Kurve normalisieren. FD machen starken Gebrauch von der Isomorphie zwischen dem $\mathbb{R}^2$ und $\mathbb{C}$, eine Übertragung ins Dreidimensionale liegt somit nicht nahe.
Weiterhin sind im 2D-Fall sogenannte polare FD bekannt, hier wird der Radius von einem Punkt (z. B. Schwerpunkt des Objektes) mit dem Winkel zur x-Achse parametrisiert. Die Randkurve des Objektes wird also als Funktion auf einem Kreis aufgefaßt, dies führt zu starken Einschränkungen. Die Idee läßt sich auch ins Dreidimensionale übertragen und führt zu Funktionen auf der Kugel. Inwieweit dort Normalisierungen bezüglich der Ähnlichkeitstransformationen möglich sind, ist offen. Fourierbasierte Techniken sollen im folgenden nicht betrachtet werden.

Bild, Schreibweise

Im folgenden soll ein Binärbild als Abbildung $B : \mathbb{B} \to \{o, i\}$ betrachtet werden, wobei $\mathbb{B} = \mathbb{R}^d$ oder $\mathbb{B} = \mathbb{Z}^d$. Weiterhin möchte ich die Schreibweise $b \in B$ verwenden, wobei dann $B = \{b \in \mathbb{B} : B(b) = i\}$ (Urbild von i) meint.

Momentbasierte Merkmale

In diesem Abschnitt soll versucht werden, Merkmale durch sogenannte Momentinvarianten zu gewinnen. Die Theorie dazu ist anspruchsvoll, weshalb zunächst dieser Hintergrund geklärt werden soll.
Zur Vereinfachung der Schreibweise ist im folgenden: $dx = dx_1 dx_2 \dots dx_d$, $x^p = x_1^{p_1} x_2^{p_2} \dots x_d^{p_d}$ und $\int_B dx$ ist das Integral über das d-dimensionale Volumen des Objektes. Weiterhin nennt man $\sum p$ Ordnung eines Moments ($m_p = m_{p_1,\dots,p_d}$).

Normalisierung

Seien zunächst die Momente nullter und erster Ordnung eingeführt:

$$\text{Masse:} \quad m_0 = \int_B dx$$

$$\text{Schwerpunkt:} \quad \overline{m}_{i=1\dots d} = \left(\frac{\int_B x_i dx}{m_0} \right)_{i=1\dots d}$$

Da wir an Merkmalen interessiert sind, die gewisse Invarianzeigenschaften aufweisen, werden die Momente skalierungs- und translationsnormiert:

$$\mu_p = \frac{\int_B (x - \overline{m})^p dx}{m_0^{\frac{d+\sum p}{d}}}$$

Die zentralen, massenormierten Momente dritter Ordnung im 3D-Fall sind beispielsweise:

$$\left\{ \frac{\int_B (x_3 - \overline{m_3})^3 dx}{m_0^2}, \frac{\int_B (x_2 - \overline{m_2})(x_3 - \overline{m_3})^2 dx}{m_0^2}, \frac{\int_B (x_2 - \overline{m_2})^2 (x_3 - \overline{m_3})^1 dx}{m_0^2}, \dots, \frac{\int_B (x_1 - \overline{m_1})^3 dx}{m_0^2} \right\}$$

Die Translationsinvarianz folgt unmittelbar aus der Additivität des Schwerpunktes. Zur Skalierungsinvarianz: $\hat{x} = \alpha x \Rightarrow \int_{\hat{B}} \hat{x}^p d\hat{x} = \int_B \alpha^{\sum p} x^p \alpha^d dx$ und $\alpha = \sqrt[d]{m_0}$. Im weiteren sind Momente in diesem

Sinne normiert. Die Anzahl der Momente ist mit $\#\left\{\mu_{\hat{p}} : \sum \hat{p} \leq \sum p\right\} = \binom{d+\sum p}{\sum p}$ gegeben (für eine feste Ordnung: Ziehen mit Zurücklegen).

Funktionalanalytische Sicht

Betrachtet man ein 1D-Bild als eine Funktion $f : [-1, 1] \to \mathbb{R}$, so lassen sich Momente als Skalarprodukt:

$$\langle f, x^n \rangle = \int_{-1}^{1} f\,(x)\, x^n dx$$

deuten. Momente bis zu einer gewissen Ordnung (n bzw. $\sum p$) charakterisieren also eine Funktion mittels ihrer Projektion in den Π_n beziehungsweise $\bigcup_p \Pi_{p_1} \times \Pi_{p_2} \times \ldots \times \Pi_{p_d}$. Für ein abgeschlossenes Gebiet, wie z. B. $[-1, 1]^d$ wird in [9] ein schneller Algorithmus basierend auf Legendrepolynomen (Orthonormalbasis) präsentiert; inwieweit daraus Merkmale ableitbar sind, bleibt offen.

Trägheitstensor

In der Physik wird der Trägheitstensor (für 3D) zur Beschreibung von Rotationen von Vielteilchensystemen benutzt. Betrachtet man den Trägheitstensor für beliebige Dimensionen:

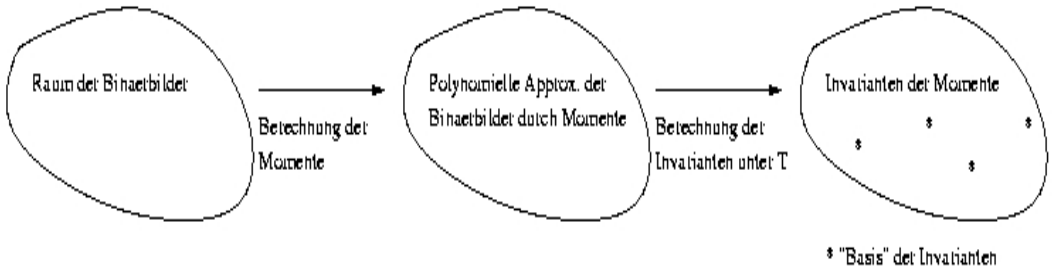
$$J = \begin{pmatrix} \mu_{2,0,\ldots,0} & \cdots & -\mu_{1,0,\ldots,1} \\ \vdots & \ddots & \vdots \\ -\mu_{1,0,\ldots,1} & \cdots & \mu_{0,\ldots,0,2} \end{pmatrix},$$

so geben dessen Eigenvektoren die Hauptausrichtungen und die Eigenwerte die Längen der Halbachsen des entstehenden Ellipsoids an. Kanonische Invarianten (gegenüber starren Abbildungen) zweiter Ordnung sind also die Eigenwerte $(\lambda_1, \ldots, \lambda_d)$ des Trägheitstensors. So werden also aus den $\frac{(d+1)d}{2}$ Momenten zweiter Ordnung d Invarianten berechnet. Neben den Eigenwerten sind außerdem noch sogenannte Spurinvarianten ($spur\,(J), spur\left(J^2\right), \ldots$) in der Literatur (z. B. [10]) bekannt, diese geben dann eben ($\sum \lambda, \sum \lambda^2, \ldots$) an. Für den 2D-Fall wird weiterhin die Exzentrizität $\epsilon = \frac{(\mu_{2,0}-\mu_{2,0})^2+4\mu_{1,1}^2}{(\mu_{2,0}+\mu_{2,0})^2} = \left(\frac{\lambda_1-\lambda_2}{\lambda_1+\lambda_2}\right)^2$ definiert. Ein interpretierbares Merkmal in 3D wäre beispielsweise $\frac{\lambda_{min}}{\lambda_{max}}$.

Momentinvarianten

Bei Momenten höherer Ordnung stellen sich folgende Fragen:

- Unter welchen Transformationen (T) sollen die Merkmale invariant sein? - hier: Ähnlichkeitstransformationen
- Beschreibt eine Menge der Merkmale aus Momenten gewisser Ordnung das Objekt hinreichend für diese Ordnung? - Vollständigkeit
- Sind in einer solchen Menge redundante Informationen enthalten? - Unabhängigkeit



Invarianten

Sei das homogene Polynom f eine algebraische Form in d Veränderlichen der Ordnung $\sum p$, d. h.:

$$f(x) = \sum_{p:\sum p=const.} \frac{(\sum p)!}{p_1!\ldots p_d!} a_p x^p,$$

dann heißt ein Polynom $I(a)$ der Koeffizienten Invariante unter einer Abbildung T ($\Delta = \det(T)$), falls:

$$I(a) = \Delta^\omega I(\hat{a})$$

für $\hat{x} = Tx$ und $\hat{a}$ entsprechend ($\hat{f}(\hat{x}) = f(x)$). Die Ordnung k einer Invariante ist der Grad des Polynoms $I(a)$, ω nennt man Gewicht (für $\omega = 0$ heisst I absolute Invariante). Ein Menge von Invarianten $\mathcal{I} = \{I_1, I_2, \ldots\}$ heisst "Basis" eines Systems des Tensors $\mathcal{S}$ (durch die Koeffizienten einer algebraischen Form wird ein symmetrischer Tensor geformt), falls sich jede Invariante dieses Systems als ganzzahlig gebrochen rationale Funktion in $\mathcal{I}$ darstellen läßt (Vollständigkeit) und keine der in $\mathcal{I}$ enthaltenen Invarianten sich so durch die anderen ausdrücken läßt (Unabhängigkeit).
Durch [8] wurde das Konzept der algebraischen Invarianten erstmals in den Bereich der Mustererkennung eingeführt. Weitere Arbeiten wurden mit [3], [14] und [12] veröffentlicht.
Der Fundamentalsatz der Momentinvarianten ([14], [12]) sagt nun, daß für jede algebraischen Invariante $I(a)$ eine Momentinvariante existiert mit:

$$I(\hat{\mu}) = \Delta^\omega |\Delta|^k I(\mu)$$

Beispielsweise läßt sich die oben gezeigte Skalierungsinvarianz auch in diesem Zusammenhang beweisen ($k = 1$).

Spezielle Invarianten 2D

Für den 2D-Fall wurden seit Beginn der sechziger Jahre mehrere Arbeiten veröffentlicht, hier sollen nun die Ergebnisse zweier sehr wichtiger Arbeiten präsentiert werden ([8], [5]).

Hu

Hu formulierte 1962 erstmals den Fundamentalsatz der Momentinvarianten, allerdings in nicht vollständiger Form. Er leitete daraus die aufgeführten Invarianten ab:

$$\begin{aligned}
\phi_1 &= \mu_{2,0} + \mu_{0,2} \\
\phi_2 &= (\mu_{2,0} - \mu_{0,2})^2 + 4\mu_{1,1}^2 \\
\phi_3 &= (\mu_{3,0} - 3\mu_{1,2})^2 + (3\mu_{2,1} - \mu_{0,3})^2 \\
\phi_4 &= (\mu_{3,0} + \mu_{1,2})^2 + (\mu_{2,1} + \mu_{0,3})^2 \\
\phi_5 &= (\mu_{2,0} - \mu_{0,2})(\mu_{3,0} + \mu_{1,2})\left((\mu_{3,0} + \mu_{1,2})^2 - 3(\mu_{2,1} + \mu_{0,3})^2\right) + \\
&\quad (3\mu_{2,1} - \mu_{0,3})(\mu_{2,1} + \mu_{0,3})\left(3(\mu_{3,0} + \mu_{1,2})^2 - (\mu_{2,1} + \mu_{0,3})^2\right) \\
\phi_6 &= (\mu_{2,0} - \mu_{0,2})\left((\mu_{3,0} + \mu_{1,2})^2 - (\mu_{2,1} + \mu_{0,3})^2\right) + \\
&\quad 4\mu_{1,1}(\mu_{3,0} + \mu_{1,2})(\mu_{2,1} + \mu_{0,3}) \\
\phi_7 &= (3\mu_{2,1} - \mu_{0,3})(\mu_{3,0} + \mu_{1,2})\left((\mu_{3,0} + \mu_{1,2})^2 - 3(\mu_{2,1} + \mu_{0,3})^2\right) - \\
&\quad (\mu_{3,0} - 3\mu_{1,2})(\mu_{2,1} + \mu_{0,3})\left(3(\mu_{3,0} + \mu_{1,2})^2 - (\mu_{2,1} + \mu_{0,3})^2\right)
\end{aligned}$$

Leider haben diese weitverbreiteten Momentinvarianten den Nachteil, daß $\phi_3 = \frac{\phi_5^2 + \phi_7^2}{\phi_4^3}$, d. h. diese Menge ist nicht unabhängig. Weiterhin wird durch den folgenden Abschnitt klar, daß diese Menge nicht vollständig sein kann.

Flusser

Flusser macht nun, was ja für den 2D-Fall eine gute Idee ist, Gebrauch von der Isomorphie zwischen dem $\mathbb{R}^2$ und $\mathbb{C}$. Er benutzt sogenannte komplexe Momente:

$$c_{p,q} := \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)(x+iy)^p(x-iy)^q\,dxdy$$

Zu den oben definierten Momenten existiert folgender Zusammenhang:

$$c_{p,q} = \sum_{k=0}^{p}\sum_{j=0}^{q}\binom{p}{k}\binom{q}{j}(-1)^{q-j}\,i^{p+q-k-j}\mu_{k+j,p+q-k-j}$$

Motiviert wird die Einführung dieser Momente durch einfachere Eigenschaften bei Rotation des Objektes um den Ursprung mit einem Winkel α. Für das rotierte Objekt f' gilt (Beweis: Polardarstellung, dann analog zum Verschiebungssatz der Fouriertransformation):

$$c'_{p,q} = e^{-i(p-q)\alpha}c_{p,q}$$

Somit sind also $\{|c_{p,q}| : p,q \in \mathbb{N}\}$ rotationsinvariant, allerdings ist diese Menge keine hinreichende Beschreibung im oben definierten Sinn.
Flusser konstruiert eine "Basis" von Momentinvarianten (beliebige Ordnung), von denen im folgenden die aufgeführt sind, die auch in [8] konstruiert werden sollten (ψ_4 ist nicht durch Hu's Momente darstell-

bar).

$$\begin{aligned}
\psi_1 &= c_{1,1} = \phi_1 \\
\psi_2 &= c_{2,1}c_{1,2} = \phi_4 \\
\psi_3 &= Re\left(c_{2,0}c_{1,2}^2\right) = \phi_6 \\
\psi_4 &= Im\left(c_{2,0}c_{1,2}^2\right) \\
\psi_5 &= Re\left(c_{3,0}c_{1,2}^3\right) = \phi_5 \\
\psi_6 &= Im\left(c_{3,0}c_{1,2}^3\right) = \phi_7
\end{aligned}$$

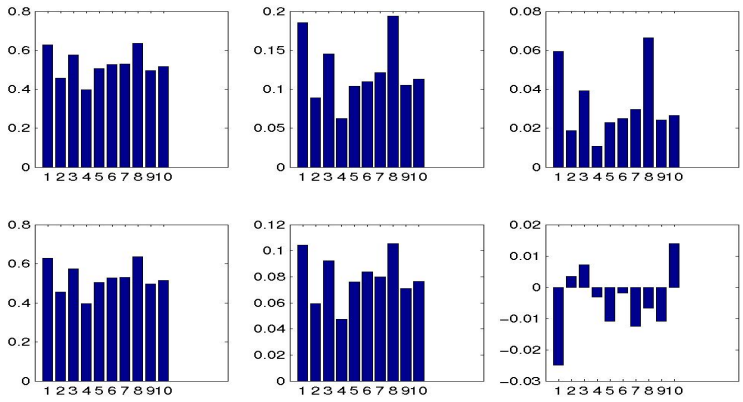
Spezielle Invarianten 3D

Im 3D-Fall sieht es wiederum ungünstiger aus. Zwar stellt Reiss in [14] grundsätzliche Methoden zur Konstruktion von Momentinvarianten dar; dennoch läßt sich mit einfachen Mitteln nur die Diskriminate/charakteristische Invariante ableiten:

$$\begin{aligned}
\gamma_1 &= \mu_{2,0,0} + \mu_{0,2,0} + \mu_{0,0,2} \\
\gamma_2 &= \mu_{2,0,0}\mu_{0,2,0} + \mu_{2,0,0}\mu_{0,0,2} + \mu_{0,2,0}\mu_{0,0,2} - \mu_{1,1,0}^2 - \mu_{1,0,1}^2 - \mu_{0,1,1}^2 \\
\gamma_3 &= \mu_{2,0,0}\mu_{0,2,0}\mu_{0,0,2} + 2\mu_{1,1,0}\mu_{1,0,1}\mu_{0,1,1} - \mu_{2,0,0}\mu_{0,1,1}^2 - \mu_{0,2,0}\mu_{1,0,1}^2 - \mu_{0,0,2}\mu_{1,1,0}^2
\end{aligned}$$

Ergebnisse

Als Testdatensatz wurden zehn Volumen der v1-Region (Visueller Kortex) untersucht. Diese wurden von Frau PD Dr. K. Amunts (IME, Forschungszentrum Jülich) zur Verfügung gestellt:



Die obere Zeile zeigt die Spurinvarianten ($spur\left(J\right), spur\left(J^2\right), spur\left(J^3\right)$), die unter die charakteristischen Invarianten ($\gamma_1, \gamma_2, \gamma_3$).
Abschließend wird klar, daß Momentinvarianten ein Mittel sind, um Objekte zu beschreiben. Dies geschieht jedoch in einem sehr formalen Rahmen und die Interpretierbarkeit der gewonnenen Merkmale, d. h. die Repräsentation von Unterschieden bzw. Ähnlichkeiten durch diese, muß weiter untersucht werden.

Morphologische Eigenschaften

Morphologische Bildverarbeitung befaßt sich allgemein mit der Formanalyse von Objekten. Im folgenden wollen wir basierend auf Mengenoperationen und der (euklidischen) Geometrie Merkmale aus Binärbildern extrahieren. Wir betrachten dazu wiederum $b \in B$, falls $B(b) = i$, b also zum Objekt gehört. Arithmetische Operationen wie Addition und Subtraktion oder der (euklidische) Distanzbegriff spielen sich auf dem zugrundeliegenden Gitter (z. B. $\mathbb{R}^n$) ab.
Ein Strukturelement (S) ist eine Menge von Punkten, die mit dem zu analysierenden Objekt verknüpft wird, im weiteren werden dies im wesentlichen (d-dim.) Kugeln sein, die also mittels ihres Radius und Mittelpunktes (Ursprung) vollständig beschrieben sind. Ein morphologischer Operator ist somit durch das Strukturelement (bzw. den Radius einer Kugel) parametrisiert. Durch Verwendung von Kugeln als Strukturelement gelangt man zu translations- und rotationsinvarianten Merkmalen. Wie sich auch Skalierungsinvarianz erlangen läßt, soll später gezeigt werden. Für einen allgemeineren Einstieg in die mathematische Morphologie kann man [7] verwenden.

Operatoren

Dilatation

Ein Punkt p liegt genau dann im dilatierten Bild, falls es einen Punkt im Ausgangsbild gibt, so daß das dorthin verschobene Strukturelement p überdeckt.

$$\delta_S(B) = B \oplus S := \{b + s : b \in B, s \in S\},$$

Erosion

Ein Punkt wird aus dem Ausgangsbild "gelöscht", falls das dorthin verschobene Strukturelement einen Punkt des Komplementes des Ausgangsbildes überdeckt.

$$\epsilon_S(B) = B \ominus S := \{h : \forall s \in S\,(s + h \in B)\},$$

Opening

Durch eine Öffnung (Opening) werden Details gelöscht, da nach der Erosion das Bild durch die Dilatation i. Allg. nicht wieder vollständig rekonstruiert wird. Das Entfernen von Bildpunkten geschieht vorsichtiger als bei der Erosion und soll später die Grundlage zur Merkmalsgewinnung sein.

$$\alpha_S(B) = B \circ S := (B \ominus S) \oplus S,$$

Weiterhin heisst B S-offen falls $B \circ S = B$.

Eigenschaften

Öffnungen besitzen folgende Eigenschaften (Charakterisierung):

Anti-Extensivität:	$B \circ S \subset B$, d. h. "Details" werden gelöscht
Monotonie:	$B_1 \subset B_2 \Rightarrow \alpha_S(B_1) \subset \alpha_S(B_2)$, dieses Löschen geschieht "gutartig"
Idempotenz:	$\alpha_S \alpha_S = \alpha_S$, dito

Größenverteilung

Im folgenden wollen wir ein Maß entwickeln, das die Form des Objektes hinsichtlich Größe/Komplexität beschreibt. Dieses Maß wird keine vollständige Beschreibung sein, es soll lediglich interpretierbare Merkmale liefern.

Granulometrie

Eine Familie $\mathcal{S} := \{S(r) : r > 0\}$ von Strukturelementen mit: $S(s)$ ist $S(r)$-offen für $s \geq r$ heisst Granulometrie. Die Familie der zugehörigen Öffnungen erfüllt $\alpha_s \alpha_r = \alpha_r \alpha_s = \alpha_s$ für $s \geq r$, d. h. nacheinander ausgeführte Öffnungen mit verschiedenen Strukturelementen aus $\mathcal{S}$ können durch eine Öffnung mit dem größeren der beiden ersetzt werden.

Musterspektrum

Sei $\int_B dx = 1$, so heisst die Abbildung $PS : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ mit:

$$PS(r) := -\frac{d}{dr} \int_{B \circ S(r)} dx$$

Musterspektrum (Pattern Spectrum, PECSTRUM, size distribution).

Diskrete Dichtefunktion

Sei $\mathcal{S} := \{S(r) : r \in \mathbb{N}\}$ eine Familie von Kugeln mit ganzzahligem Radius r, so geht $PS(r) = PS_{cdf}(r+1) - PS_{cdf}(r)$ mit: $PS_{cdf}(r) = 1 - \frac{\int_{B \circ S(r)} dx}{\int_B dx}$ in eine diskrete Dichtefunktion über.

Distanztransformation

Die Abbildung $DT : \mathbb{B} \to \mathbb{R}_0^+$ mit:

$$DT(b) := \min\{||b - a|| : a \in B\}$$

heisst Distanztransformation, für $||\cdot|| = ||\cdot||_2$ euklidische Distanztransformation (EDT).

Zusammenhang Erosion

Sei S eine Kugel mit Radius r_S, so gilt:

$$B \ominus S = \{b \in \mathbb{B} : EDT_{B^c}(b) \geq r_S\}$$

Schneller Algorithmus

Über eine längere Zeitspanne gab es keinen Ansatz, die Funktion EDT für digitale Bilder ($\mathbb{B} = \mathbb{Z}^n$) effizient, d. h. in linearer Zeit, zu berechnen. Dies führte unter anderem auch zu anderen Distanzmassen, die EDT approximieren und effizient berechenbar sind.
Durch eine Separation der Dimensionen (Idee: $||x||^2 = \left(x_1^2 + x_2^2\right) + x_3^2$) und dynamische Programmierung wurde in [1] und [18] schließlich ein effizienter Algorithmus vorgestellt. Dieser ist hier für beliebige Dimensionen implementiert.

Opening-Transformation

Da, wie oben festgestellt, ein enger Zusammenhang zwischen Erosion mit einer Kugel und der Distanztransformierten besteht, soll nun ein Algorithmus entwickelt werden, der dies ausnutzt. Zunächst wird die EDT geringfügig modifiziert (passend zu den Kugeln mit ganzzahligem Radius), es soll $ET : \mathbb{B} \to \mathbb{N}_0$ mit:

$$ET(b) := \lceil EDT_{B^c}(b) \rceil - 1$$

Analog zur Distanztransformation gilt:

$$\epsilon_r(B) = \{b \in \mathbb{B} : ET(b) \geq r\}$$

Nun soll die Halbgruppeneigenschaft der Öffnung ausgenutzt werden:

$$\alpha_r \alpha_{r-1} = \alpha_r \Rightarrow \alpha_r \alpha_{r-1} \ldots \alpha_1(B) = \alpha_r(B) = \delta_r(\epsilon_r(B))$$

Weiterhin soll die Dilatation, die jeder Erosion anzuschließen ist, nur auf dem "Rand" des erodierten Bildes ausgeführt werden, d. h.:

$$\begin{aligned} \alpha_r(B) &= \delta_r(\epsilon_r(B)) \\ &= \delta_r(\{b \in \mathbb{B} : ET(b) \geq r\}) \\ &= \delta_r(\{b \in \mathbb{B} : ET(b) = r\}) \cup \{b \in \mathbb{B} : ET(b) \geq r+1\} \end{aligned}$$

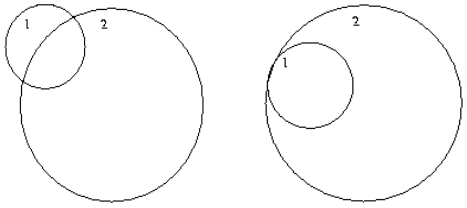
Daraus ergibt sich eine verallgemeinerte Transformation, die jedem Bildpunkt b den Wert r zuweist, so daß nach einem Opening von B mit einer Kugel mit Radius r der Bildpunkt noch erhalten ist, bei einem Opening mit einer größeren Kugel jedoch verschwindet.

$$OT(b) := \max\{r \in \mathbb{N}_0 : b \in \alpha_r(B)\}$$

Dabei läßt sich folgendes beobachten:

1. $\forall a, b \in B : OT_{B\setminus\{b\}}(a) \leq OT_B(a)$, genauer:
   $\max\left(OT_{B\setminus\{b\}}(B), \{(b+v) \times ET(b) : ||v|| \leq ET(b)\}\right) = OT_B(B)$, d. h. das Bild kann beliebig durchlaufen werden, falls die "Dilatation" eines Bildpunktes nur dann zur Öffnungstransformation beiträgt, wenn sie größer ist als der bisherige Wert an der Stelle a ($\max$ ist punktweise zu verstehen).
2. $EDT_{B^c}(b+v) + ||v|| \leq EDT_{B^c}(b) \Rightarrow ball(b+v, ET(b+v)) \subset ball(b, ET(b))$, d. h. "Dilatationskugeln" innerhalb größerer können unbeachtet bleiben.

Die folgenden Abbildungen illustrieren die Beobachtungen. Im ersten Fall wird durch die Hinzunahme des Mittelpunktes von Kreis 2 im Schnittbereich der beiden Kreise ein neues Maximum angenommen. Im zweiten Fall kann Kreis 1 unberücksichtigt bleiben.



Somit ergibt sich folgender Pseudocode:

```
for(b with ET(b) ≥ 0)
 for(v with ||v|| ≤ ET(b))
  OT(b+v) = max{OT(b+v), ET(b)}
  if(EDT(b+v) + ||v|| ≤ EDT(b))
   ET(b+v) = 0
  endif
 endfor
endfor
```

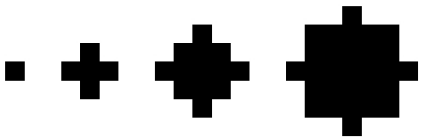
Der Zusammenhang zum Patternspektrum ergibt sich schließlich mit:

$$PS(r) = \int_B 1\{OT(b) = r\}\, dx,$$

d. h. als Histogramm der Öffnungstransformierten.

Diskretes Bild

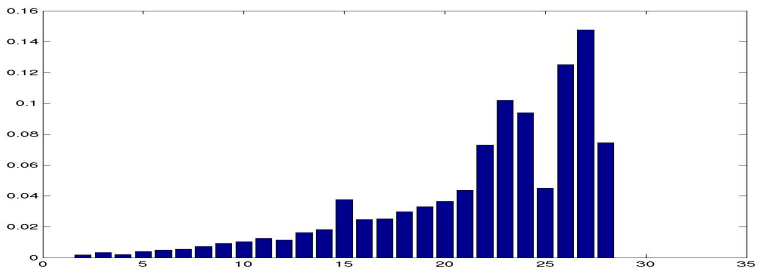
Wird nun vom kontinuierlichen Gitter zu einem diskreten übergegangen, so stellen sich einige Probleme. Zunächst haben die Kugeln folgende Form:



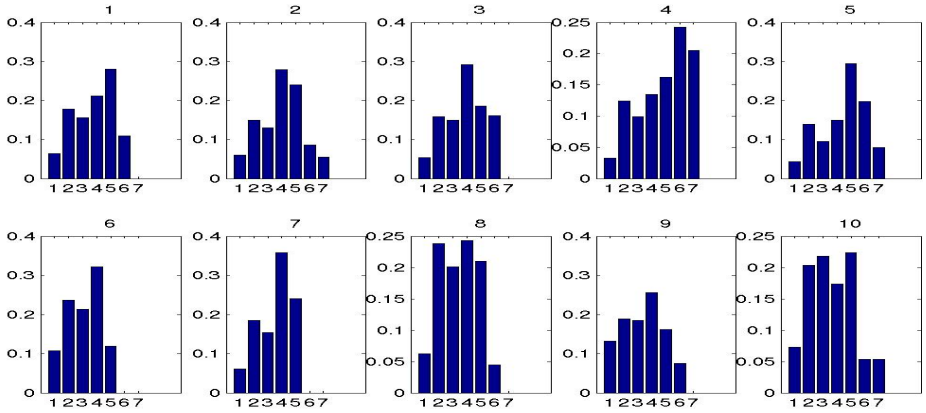
Dies wirkt sich natürlich insbesondere bei Rotationen aus. Ein möglicher Ausweg könnte im Filtern des Bildes liegen. Öffnet man es zunächst mit einer Kugel, die den Radius 1 hat, so erzwingt man beispielsweise $PS(0) = 0$.
Weiterhin scheint das Musterspektrum erst ab gewisser Gesamtgröße des Objektes Sinn zu machen, detailierte Aussagen sind hier noch nicht möglich.

Ergebnisse

Als Testdatensatz wurde die isotropisierte Version von “110_3.buchar” aus den “IBSR”-Daten verwendet. Es wurde das Musterspektrum für das komplette Gehirn berechnet:



Weiterhin wurden zehn Volumen der v1-Region (Visueller Kortex) untersucht:



In den Diagrammen ist jeweils $PS(r)$ gegen den Radius der Kugel abgetragen. Inwieweit sich dieses Maß als (interpretierbare) Beschreibung der Form eignet, muß weiter untersucht werden. Weiterhin ist bis zum jetzigen Zeitpunkt keine Skalierungsinvarianz gewährleistet. Da sich eine Skalierung als Verschiebung im Musterspektrum bemerkbar machen sollte, könnte bei Vergleichen die Kreuzkorrelation verwendet werden. Um direkt Merkmale abzuleiten, müßte man das Musterspektrum verschiebungsinvariant beschreiben.

Implementierung

Die Routinen wurden alle in C implementiert. Die Subroutinen reservieren selbst Speicher für das Ergebnis. Die Daten werden derart durchlaufen, daß die letzte Dimension (z für 3D) direkt durchlaufen wird, für die anderen ergibt sich ein entsprechender Offset.

Momente

Zum Berechnen der Momente wurde ein Brute-Force-Algorithmus implementiert.

- int moments(float** result_p,int dims_size,int* dims,int order,short* img,int type):
  Funktion: berechnet die Momente des Binärbildes
  type: aus {MASS, MEAN, NORMALIZED}
  result: ($\mu_{0,\ldots,0}$),
  ($\mu_{0,\ldots,1}, \ldots, \mu_{1,\ldots,0}$) bzw.
  ($\mu_{0,\ldots,2}, \mu_{0,\ldots,1,1} \ldots \mu_{2,0,\ldots}, \mu_{0,\ldots,3}, \mu_{0,\ldots,1,2}, \ldots$)

- int moment_features_2d(float** result_p,int* dims,short* img,int invariant):
  Funktion: berechnet die angegebenen Invarianten
  invariant: aus {EIGEN_INVARIANT, TRACE_INVARIANT, CHARACTERISTIC_INVARIANT, HU_INVARIANT, FLUSSER_INVARIANT}
  Ordnung: 3 für invariant aus {HU_INVARIANT, FLUSSER_INVARIANT}
  2 sonst

- int moment_features_3d(float** result_p,int* dims,short* img,int invariant):
  Funktion: berechnet die angegebenen Invarianten
  invariant: aus {TRACE_INVARIANT, CHARACTERISTIC_INVARIANT}
  Ordnung: 2

Musterspektrum

Für das Musterspektrum wurde der oben entwickelte Algorithmus implementiert. Er verwendet die Distanztransformation nach [1] und die vorgestellten Vereinfachungen.

- void euclidean_distance_trafo(float** result_p,int dims_size,int* dims, short* img,int mode):
  Funktion: berechnet die euklidische Distanztransformierte
  mode: aus {COMPLEMENT_DISTANCE, SELF_DISTANCE}
- void opening_trafo(int** result_p,int dims_size,int* dims,short* img):
  Funktion: berechnet die Öffnungstransformierte
- void pecstrum(float** result_p,int dims_size,int* dims,short* img):
  Funktion: berechnet das Musterspektrum

Literatur

1. W.H. Hesselink A. Meijster, J.B.T.M. Roerdink. A general algorithm for computing distance transforms in linear time. Technical Report, Institute for Mathematics and Computing Science, University of Groningen, The Netherlands, 2000.
2. R.M. Brannon. Elementary vector and tensor analysis. University of New Mexico, Albuquerque, June 2001.
3. E.L. Hall, F.A. Sadjadi. Three-dimensional moment invariants. *IEEE Trans. Pattern Analysis Machine Intelligence*, page 127 136, March 1980.
4. J. Flusser. Fast calculation of geometric moments of binary images. In M. Gengler, editor, *Pattern Recognition and Medical Computer Vision*, pages 265–274, Illmitz, 1998. ÖCG.
5. J. Flusser. On the independence of rotation moment invariants. *PR*, 33(9):1405–1410, September 2000.
6. J. Flusser and T. Suk. Pattern recognition by affine moment invariants. *Pattern Recogn.*, 26(1):167-174, 1993.
7. H. J. A. M. Heijmans. Mathematical morphology: basic principles. In *Proceedings of Summer School on "Morphological Image and Signal Processing"*, Zakopane, Poland, 1995.
8. M. K. Hu. Visual pattern recognition by moment invariants. *IRE Trans. Information Theory*, page 179-187, February 1962.
9. D. Shen, J. Shen, W. Shen. On geometric and orthogonal moments. *IEEE Trans. Pattern Analysis Machine Intelligence*, 14(7):875-894, 2000.
10. Bernd Jaehne. *Digitale Bildverarbeitung.* Springer-Verlag, 1997.
11. Ming-Chin Lu. *Computer modeling and simulation techniques for computer vision problems.* PhD thesis, State University of New York at Stony Brook, 1993.
12. A.G. Mamistvalov. n-dimensional moment invariants and conceptual mathematical theory of recognition n-dimensional solids. *IEEE Trans. Pattern Analysis Machine Intelligence*, page 819 831, August 1998.
13. P.F.M. Nacken. *Image analysis methods based on hierarchies of graphs and multi-scale mathematical morphology*. PhD thesis, Universiteit van Amsterdam, 1994.
14. T.H. Reiss. *Recognizing planar objects using invariant image features*, volume 676 of *Lecture notes in computer science*. Springer-Verlag, 1993.
15. John C. Russ. *The image processing handbook.* CRC Press, Inc., 1995.
16. R.M. Haralick, S. Chen. Recursive opening transform. Technical report, Department of electrical engineering, FT-10, University of Washington, Seattle, Washington 98195, ?
17. Issai Schur. *Vorlesungen über Invariantentheorie.* Springer-Verlag, 1968.
18. J.B.T.M. Roerdink, W.H. Hesselink, A. Meijster. An exact euclidean distance transform in linear time. Technical Report 9, Institute for Mathematics and Computing Science, University of Groningen, The Netherlands, April 1999.

Lösung des verallgemeinerten Eigenwertproblems in quantenchemischen Rechnungen

Katja Lord

Universität Bayreuth

katja.lord@stud.uni-bayreuth.de

Zusammenfassung: Unter Verwendung von Programmbibliotheken werden zwei verschiedene Herangehensweisen zur Lösung des verallgemeinerten Eigenwertproblems miteinander verglichen. Die Reduktion auf ein Standardeigenwertproblem erfolgt dabei zum einen mittels Cholesky-Faktorisierung, zum anderen mit Löwdin-Orthogonalisierung. Die verwendeten Routinen sind den Programmbibliotheken ScaLAPACK und PBLAS bzw. bei sequentieller Rechnung LAPACK und BLAS entnommen. Alternativ wird die Routine RDIAG, die an der Universität Karlsruhe entwickelt wurde, als Eigenwertlöser untersucht. Dabei steht der Vergleich von Berechnungszeiten und Speicherbedarf sowie Skalierungen hinsichtlich der Prozessoranzahl im Vordergrund.

Einleitung

Quantenchemische Berechnungsmethoden haben in den letzten Jahren zunehmende Bedeutung in der Behandlung praxisrelevanter chemischer und biologischer Systeme erlangt. Abhängig von der Größe der zu berechnenden Systeme sowie von verfügbarer Rechenkapazität werden Systeme entweder vollständig quantenchemisch oder durch die Kopplung von quantenchemischen Methoden mit klassischer Molekülmechanik behandelt.

Konventionelle quantenmechanische Methoden sind im Allgemeinen sehr rechenzeitintensiv. Der Rechenzeitaufwand steigt mit der vierten Potenz der Molekülgröße. Korrelationsmethoden haben eine noch ungünstigere Skalierung. Sehr schnell werden bereits kleine Systeme mit nur wenigen Elektronen nicht mehr behandelbar.

Um dieses Problem zu lösen geht man zwei Wege: Einerseits werden die Methoden bezüglich ihrer Skalierung mit der Molekülgröße untersucht. Die Berechnung von Beiträgen, die für die Ergebnisse ohne praktische Relevanz sind, wird eliminiert. Im Ergebnis dieser Arbeiten werden lokale Korrelationsmethoden entwickelt. Neue theoretische Ansätze werden verwendet, die bei vergleichbarer Genauigkeit eine sehr viel günstigere Skalierung zeigen. Andererseits wird eine immer bessere Skalierung der Rechenzeit in Bezug auf die Anzahl der eingesetzten Prozessoren angestrebt.

Im Spektrum quantenchemischer Methoden nehmen die Hartree-Fock-Theorie [1, 2, 3] als Grundlage für korrelierte Methoden und die Dichtefunktionaltheorie [4, 5] eine zentrale Stellung ein. Beide Verfahren erfordern die Lösung eines verallgemeinerten Eigenwertproblems, das aus der Matrixdarstellung des Hamiltonoperators resultiert. Im Gegensatz zur Hartree-Fock-Theorie, in der die Eigenvektoren als Wellenfunktion eine direkte physikalische Bedeutung haben, werden im Rahmen der Dichtefunktionaltheorie die Eigenvektoren für die Berechnung der Elektronendichte benutzt. Beide Methoden untergliedern sich in zwei Bestandteile:

1. Berechnung der Fockmatrix, die das Ergebnis der Matrixdarstellung des Hamiltonoperators ist.
2. Diagonalisierung dieser Matrix durch Lösung eines Eigenwertproblems.

Die Berechnung der Fockmatrix im Rahmen der Dichtefunktionaltheorie skaliert annähernd linear mit der Molekülgröße, die Lösung des Eigenwertproblems [6, 7] dagegen kubisch. Der Aufwand für die Lösung des Eigenwertproblems wird daher schnell dominant. Im Allgemeinen zeigt die Berechnung der Fockmatrix eine bessere Skalierung mit zunehmender Prozessoranzahl als die Berechnung des Eigenwertproblems. Daraus leiten sich zwei Aufgabenstellungen ab:

1. Vergleich verschiedener Eigenwertlöser in Bezug auf Rechenzeitaufwand, Genauigkeit und Speicherplatzbedarf
2. Untersuchung der Skalierung der parallelen Verfahren

Dabei soll herausgefunden werden, inwiefern Eigenwertlöser, die auf einem Prozessor im Hinblick auf Rechenzeit und Speicherbedarf einen gewissen Mehraufwand aufweisen, wegen einer günstigeren parallelen Skalierung sequentiell hocheffizienten Eigenwertlösern doch vorzuziehen sind.

ScaLAPACK und LAPACK

In meinen Betrachtungen werden zwei verschiedene Methoden zur Lösung des verallgemeinerten Eigenwertproblems auf dem Linux-Cluster ZAMpano [8] untersucht. Für das Standardeigenwertproblem für vollbesetzte Matrizen werden für parallele Ausführung ScaLAPACK-Routinen, im sequentiellen Fall LAPACK-Routinen verwendet.

ScaLAPACK[9] (Scalable Linear Algebra PACKage) ist eine Programmbibliothek für Parallelrechner mit verteiltem Speicher, die hocheffiziente Routinen für Probleme aus der Linearen Algebra wie lineare Gleichungssysteme, Eigenwertprobleme und Singulärwertberechnungen bereitstellt.

ScaLAPACK stellt eine Weiterentwicklung der LAPACK-Bibliothek dar, die sequentielle Routinen für oben angesprochene Probleme beinhaltet. LAPACK (Linear Algebra PACKage) basiert auf der BLAS-Bibliothek (Basic Linear Algebra Subprograms), ScaLAPACK auf der parallelisierten Variante, der Parallel BLAS- bzw. PBLAS-Library.

Hinsichtlich der Kommunikation zwischen den Prozessoren basiert ScaLAPACK auf BLACS (Basic Linear Algebra Communication Subprograms), einer Programmbibliothek, die Message Passing speziell für Probleme aus der Linearen Algebra bereitstellt. In der Regel sind diese BLACS auf der Grundlage von MPI (Message Passing Interface) implementiert (CRAY BLACS beruhen auf shmem calls). Beim zugrunde liegenden Kommunikationsmodell handelt es sich um ein rechteckiges Prozessorgitter.

LAPACK- und ScaLAPACK-Routinen lassen sich in die folgenden drei Kategorien einteilen:

- Treiberroutinen:
  Darunter versteht man Routinen, die ein Problem, wie das Lösen eines linearen Gleichungssystems oder die Berechnung der Eigenwerte einer reellen symmetrischen Matrix, komplett lösen. Dabei werden von einer Treiberroutine eine Reihe von Routinen der nächsten Kategorie, sogenannte Rechenroutinen, aufgerufen.
- Rechenroutinen:
  Hierunter fallen Routinen, die einen Teilschritt wie z. B. eine LR-Zerlegung oder die Reduktion einer reellen symmetrischen Matrix auf Tridiagonalgestalt durchführen.

- Hilfsroutinen:
  Die Hilfsroutinen dienen zur Lösung einer Teilaufgabe wie z. B. der Berechnung einer Matrixnorm.

Die BLAS-Bibliothek stellt Routinen zur Lösung elementarer Probleme aus der Linearen Algebra bereit. Eine Einteilung hinsichtlich ihrer Komplexität erfolgt in Level-1- (Vektor-Vektor-Operationen), Level-2- (Matrix-Vektor-Operationen) und Level-3-BLAS-Routinen (Matrix-Matrix-Operationen). Analoges gilt für PBLAS.

Schematisch sei die Bibliothekshierarchie von ScaLAPACK und LAPACK in der folgenden Graphik (Abb. 1) veranschaulicht, wobei die parallelen Bibliotheken sich oberhalb, die lokal von einem Prozessor aufgerufenen Bibliotheken sich unterhalb der gestrichelten Linie befinden.

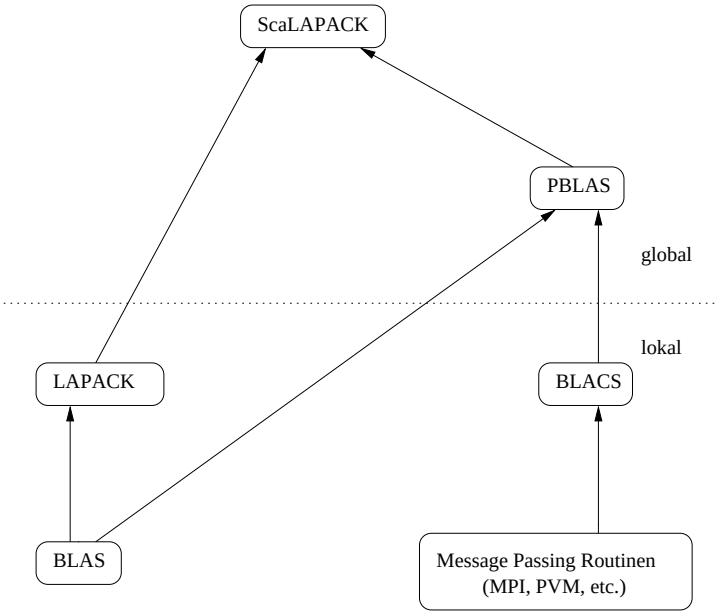


Abbildung 1: ScaLAPACK-Hierarchie

Methoden zur Lösung des verallgemeinerten Eigenwertproblems

Ausgangspunkt meiner Untersuchungen ist das verallgemeinerte Eigenwertproblem

$$HC = SCE, \tag{1}$$

$H, S, C, E \in \mathbb{R}^{n \times n}$, $H = H^t$, $S = S^t$, S positiv definit.

Gegeben sind dabei die Fockmatrix H und die Überlappungsmatrix S.
Gesucht sind die Eigenwerte bzw. die Diagonalmatrix E, deren Einträge auf der Diagonalen gerade die Eigenwerte darstellen, sowie die Matrix C.

Vergleichend werden zwei Methoden zur Reduktion des verallgemeinerten Eigenwertproblems auf ein Standardeigenwertproblem betrachtet:

Lösen des verallgemeinerten Eigenwertproblems mittels Cholesky-Faktorisierung

Für diese Methode wird die Cholesky-Faktorisierung der Matrix S berechnet und im Folgenden das verallgemeinerte Eigenwertproblem (1) auf ein Standardeigenwertproblem

$$\tilde{H}V = VE, \tag{2}$$

$\tilde{H} \in \mathbb{R}^{n \times n}$, $\tilde{H} = \tilde{H}^t$ symmetrisch, reduziert.

1. Zunächst ist die Cholesky-Faktorisierung der Matrix S durchzuführen (ScaLAPACK-Routine PDPOTRF):

$$S = LL^t \text{ bzw. } S = U^tU$$

2. Daraufhin wir das verallgemeinerte Eigenwertproblem (1) in ein Standardeigenwertproblem (2) transformiert (ScaLAPACK-Routine PDSYGST):

$$\begin{aligned} & HC = SCE \\ \iff & HC = LL^tCE \\ \iff & L^{-1}H(L^{-t}L^t)C = L^tCE, \text{ da } L \text{ regulär } (S \text{ positiv definit}) \\ \iff & \tilde{H}V = VE \text{ mit } \tilde{H} := L^{-1}HL^{-t}, V := L^tC \end{aligned}$$

(analog für $S = U^tU$)

3. Das Standardeigenwertproblem wird mit der ScaLAPACK-Routine PDSYEVX gelöst, die Eigenwerte stimmen dabei mit den Eigenwerten des Ausgangsproblems überein.

4. Zur Berechnung der Eigenvektoren des Ausgangsproblems, d. h. der Matrix C, wird die Dreiecksgestalt der Matrix L bzw. U beim Lösen des Gleichungssystems

$$L^tC = V \text{ bzw. } UC = V$$

ausgenutzt (ScaLAPACK-Routine PDTRTRS).

Die vier aufgeführten Rechenroutinen PDPOTRF, PDSYGST, PDSYEVX und PDTRTRS werden beim Aufruf der Treiberroutine PDSYGVX abgearbeitet; diese Routine wird in den weiteren Betrachtungen als Blackbox gehandhabt und mit der nachfolgend beschriebenen Methode verglichen.

Lösen des verallgemeinerten Eigenwertproblems mittels Löwdin-Orthogonalisierung

Für diese Methode wird die Löwdin-Zerlegung der Matrix S berechnet und wie bei der Methode mit Cholesky-Faktorisierung das Ausgangsproblem (1) auf ein Standardeigenwertproblem (2) reduziert.

1. Zunächst ist das Standardeigenwertproblem

$$SU_s = U_sE_s$$

für die Matrix S zu lösen.
Als Resultat erhält man die Matrix der Eigenwerte E_s und die Matrix der Eigenvektoren U_s mit $U_s^tSU_s = E_s$, U_s orthogonal.

2. Hieraus können die Matrizen

$$S^{-\frac{1}{2}} := U_sE_s^{-\frac{1}{2}}U_s^t \text{ und } S^{\frac{1}{2}} := U_sE_s^{\frac{1}{2}}U_s^t$$

berechnet werden, wobei für weitere Schritte nur $S^{-\frac{1}{2}}$ direkt benötigt wird.

$E_s^{-\frac{1}{2}}$ entsteht dabei aus der Diagonalmatrix $E_s = diag(e_{11}, \ldots, e_{nn})$, e_{ii} die Eigenwerte der Matrix S, durch Bildung der Diagonalmatrix $diag(e_{11}^{-\frac{1}{2}}, ..., e_{nn}^{-\frac{1}{2}})$, $E_s^{\frac{1}{2}}$ analog.
Dabei gilt:

$$S^{\frac{1}{2}}S^{\frac{1}{2}} = U_s E_s^{\frac{1}{2}} U_s^t U_s E_s^{\frac{1}{2}} U_s^t = U_s E_s^{\frac{1}{2}} E_s^{\frac{1}{2}} U_s^t = U_s E_s U_s^t = S$$
und
$$S^{-\frac{1}{2}}S^{\frac{1}{2}} = U_s E^{-\frac{1}{2}} U_s^t U_s E^{\frac{1}{2}} U_s^t = U_s E^{-\frac{1}{2}} E^{\frac{1}{2}} U_s^t = U_s U_s^t = I = S^{\frac{1}{2}}S^{-\frac{1}{2}}$$

3. Jetzt erfolgt die Reduktion des verallgemeinerten Eigenwertproblems (1) in ein Standardeigenwertproblem (2):

   $$\begin{aligned} & HC = SCE \\ \Longleftrightarrow\ & HC = S^{\frac{1}{2}}S^{\frac{1}{2}}CE \\ \Longleftrightarrow\ & S^{-\frac{1}{2}}H(S^{-\frac{1}{2}}S^{\frac{1}{2}})C = S^{\frac{1}{2}}CE, \text{ da } S^{\frac{1}{2}} \text{ regulär } (S \text{ positiv definit}) \\ \Longleftrightarrow\ & \tilde{H}V = VE \text{ mit } \tilde{H} := S^{-\frac{1}{2}}HS^{-\frac{1}{2}},\ V := S^{\frac{1}{2}}C \end{aligned}$$

4. Das Standardeigenwertproblem $\tilde{H}V = VE$ kann nun mit Hilfe eines Standardeigenwertlösers behandelt werden. Die hierbei berechneten Eigenwerte stimmen mit den Eigenwerten des Ausgangsproblems überein.

5. Um an die Eigenvektoren des verallgemeinerten Eigenwertproblems (1) zu gelangen, muss das Gleichungssystem

   $$S^{\frac{1}{2}}C = V$$

   gelöst werden. Da bereits unter 2. die Matrix $S^{-\frac{1}{2}}$ berechnet wurde, reicht es, hierzu die Matrixmultiplikation $S^{-\frac{1}{2}}V$ mit Resultat C, der Matrix der Eigenvektoren, durchzuführen.

Für die einzelnen Schritte der Löwdin-Methode werden ScaLAPACK- und PBLAS-Routinen verwendet. Für das Lösen der beiden Eigenwertprobleme in den Schritten 1. und 4. wird die ScaLAPACK-Routine PDSYEVX verwendet, zur Berechnung der Matrix $S^{-\frac{1}{2}}$ wird zunächst eine Skalierung der Eigenvektoren mit den inversen Wurzeln der Eigenwerte vorgenommen, danach mit Hilfe der PBLAS-Routine PDGEMM durch Matrixmultiplikation die Matrix $S^{-\frac{1}{2}}$ berechnet. Zur Berechnung der Matrix $\tilde{H}$ unter 3. bzw. zur Bestimmung der Eigenvektoren des Ausgangsproblems unter 5. kann die Symmetrie der Matrix $S^{-\frac{1}{2}}$ ausgenutzt werden, so dass für die benötigten Matrixmultiplikationen die PBLAS-Routine PDSYMM herangezogen wird.

Ergebnisse der parallelen Verfahren

Für die Untersuchungen stehen 3 Testmatrizenpaare H, S mit Matrixdimension $n = 959$, 2530 und 5060 zur Verfügung, die einem quantenchemischen Programm entnommen wurden.

Um optimale Berechnungszeiten zu erreichen, wird zunächst die Blockgröße nb für die blockzyklische Datenstruktur, wie sie unter ScaLAPACK verwendet wird (vgl. [9]), ermittelt. Für die größeren der verwendeten Prozessorgitter ist hierfür ein Wert um $nb = 24$ geeignet, für die kleineren Gitter kann dagegen ein höherer Wert, jedoch im Bereich von $nb = 20$ bis $nb = 40$, ratsam sein. In den weiteren Betrachtungen wird versucht, Zeitvergleiche zwischen den Prozessorgittern jeweils mit günstiger Wahl der Blockgröße durchzuführen.

Für beide Methoden werden die Berechnungszeiten für verschiedene Gitter miteinander verglichen. Dabei kann ein deutlicher Zeitvorteil für die Methode mit Cholesky-Faktorisierung festgestellt werden (vgl. Abb. 2, 3 und 4). Eine Erklärung hierfür wird später bei Betrachtung der Einzelschritte gegeben.

Zunächst sollen jedoch die Skalierungen hinsichtlich der Prozessoranzahl näher betrachtet werden (vgl. Abb. 5, 6 und 7). Dabei wird jeweils das Gitter zum Vergleich herangezogen, das bei gleicher Prozessoranzahl besser abgeschnitten hat.

Bei beiden Verfahren sind für die Testmatrizen der Größe $n = 959$ und $n = 2530$ keine besonders guten Speedup-Werte zu beobachten, was auf die doch recht aufwendige Kommunikation zwischen den einzelnen Prozessoren zurückzuführen ist. Dies spricht nicht sehr für eine Parallelisierung der Verfahren, solange das verallgemeinerte Eigenwertproblem noch auf einem Prozessor gelöst werden kann.

Wesentlich bessere Ergebnisse liefert dagegen die Testmatrix der Größe $n = 5060$, deren Behandlung auf einem Knoten wegen zu hoher Speicherplatzanforderungen nicht mehr möglich ist. Optimale Werte können jedoch wegen aufwendiger Kommunikation auch hier nicht erwartet werden.

Werden die Zeiten für die Einzelschritte der Löwdin-Methode jeweils separat und im Vergleich mit dem verallgemeinerten Eigenwertlöser PDSYGVX (vgl. Abb. 8 und 9) betrachtet, erhält man Aufschluss über den Zeitvorteil der Methode mit Cholesky-Faktorisierung.

Man sieht, dass der größte Anteil an der gesamten Berechnungszeit der Löwdin-Methode für die Lösung der beiden Eigenwertprobleme aufgewendet wird. Die darüber hinaus benötigten Matrixmultiplikationen sowie die Skalierung der Eigenvektoren im Schritt 2 der Methode fallen dagegen deutlich weniger ins Gewicht. Wird der Zeitaufwand zur Berechnung des standardisierten Eigenwertproblems des Weiteren dem Zeitaufwand der Routine PDSYGVX, die das allgemeine Eigenwertproblem löst, gegenübergestellt, so kann als Ergebnis festgehalten werden, dass auch die Cholesky-Faktorisierung sowie die Rücktransformation der Eigenvektoren zu keinem wesentlichen Mehraufwand beitragen.

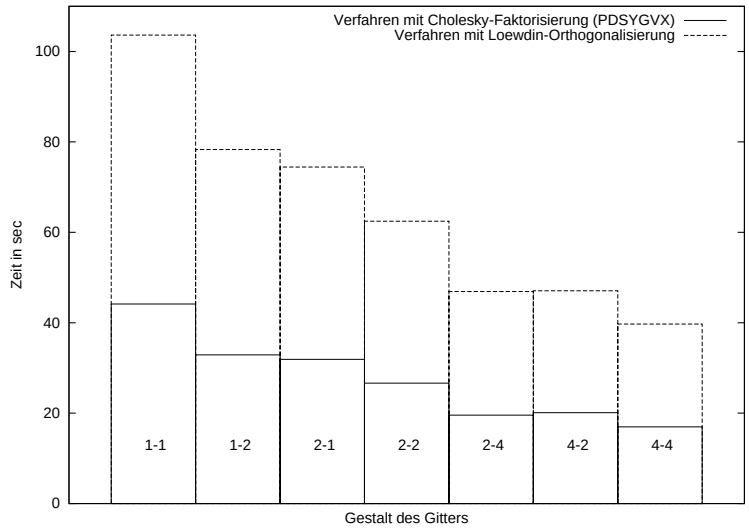


Abbildung 2: Zeitvergleich der beiden Methoden auf verschiedenen Prozessorgittern (n=959)

Weshalb die Löwdin-Methode bei der Matrix der Größe $n = 2530$ gegenüber der Methode mit Cholesky-Faktorisierung nicht ganz so schlecht abschneidet, kann durch Betrachtung der Struktur des Spektrums (vgl. Abb. 10) erklärt werden.

Aufgrund von starker Clusterbildung und deshalb durchgeführten Reorthogonalisierungen der Eigenvektoren innerhalb der Routine PDSYEVX konzentriert sich ein großer Teil der Berechnungszeit auf die Lösung des standardisierten Eigenwertproblems, so dass das erstere zur Berechnung der Matrix $S^{-\frac{1}{2}}$ weit weniger ins Gewicht fällt.

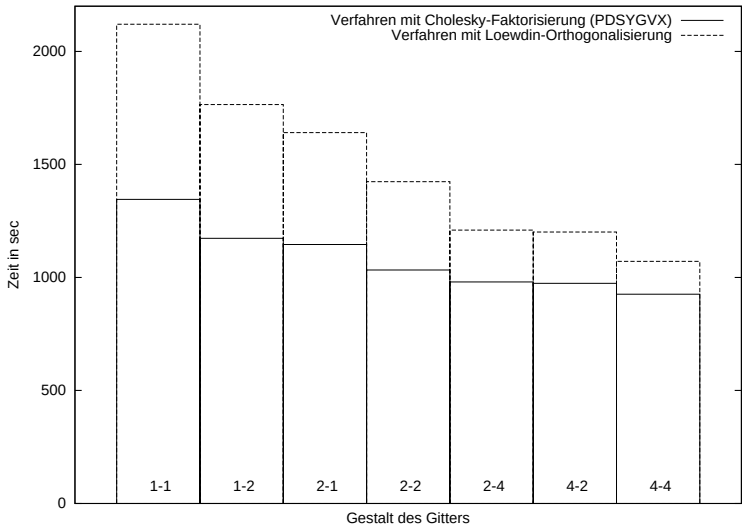


Abbildung 3: Zeitvergleich der beiden Methoden auf verschiedenen Prozessorgittern (n=2530)

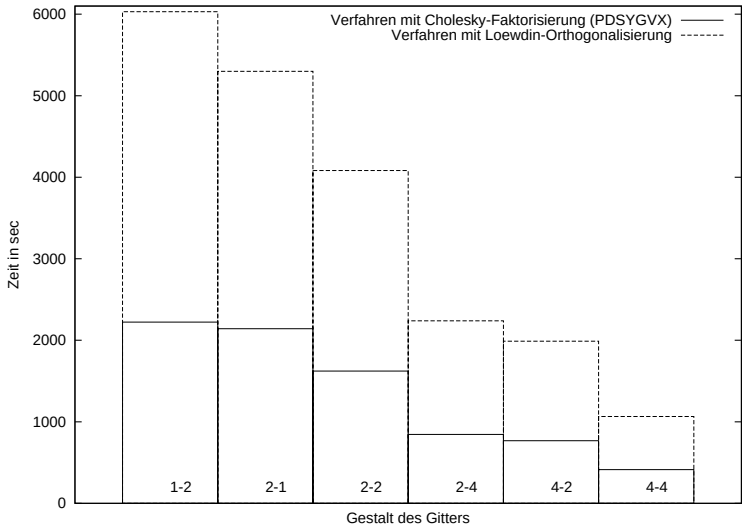


Abbildung 4: Zeitvergleich der beiden Methoden auf verschiedenen Prozessorgittern (n=5060)

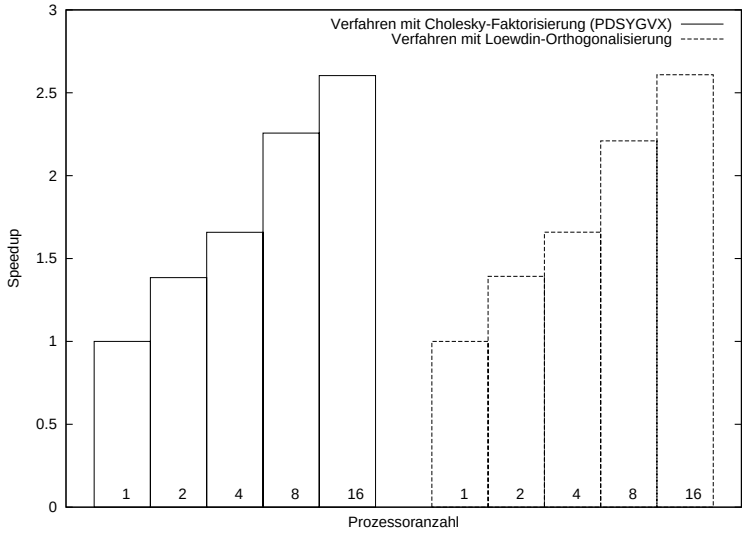


Abbildung 5: Speedup-Werte (n=959)

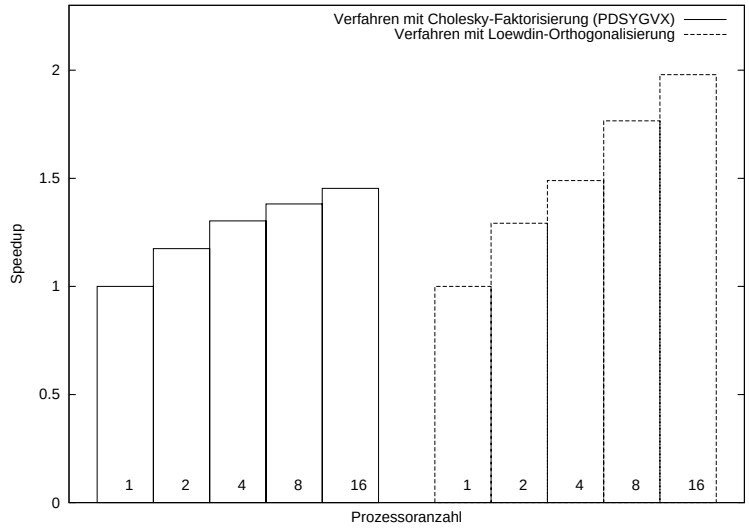


Abbildung 6: Speedup-Werte (n=2530)

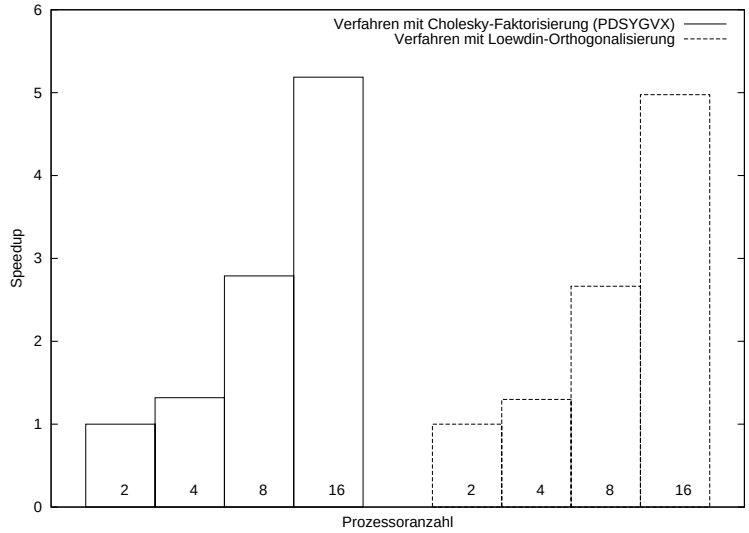


Abbildung 7: Speedup-Werte (n=5060)

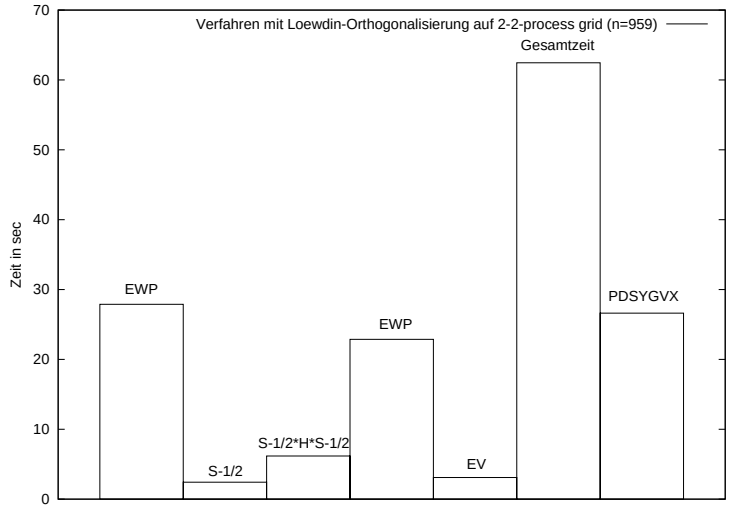


Abbildung 8: Einzelschritte bei Verfahren mit Löwdin-Orthogonalisierung (n=959) im Vergleich zur Gesamtzeit mit Cholesky-Faktorisierung

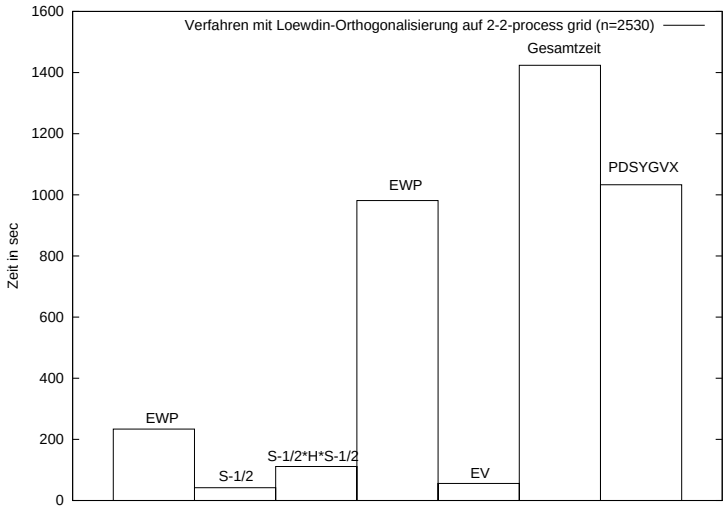


Abbildung 9: Einzelschritte bei Verfahren mit Löwdin-Orthogonalisierung (n=2530) im Vergleich zur Gesamtzeit mit Cholesky-Faktorisierung

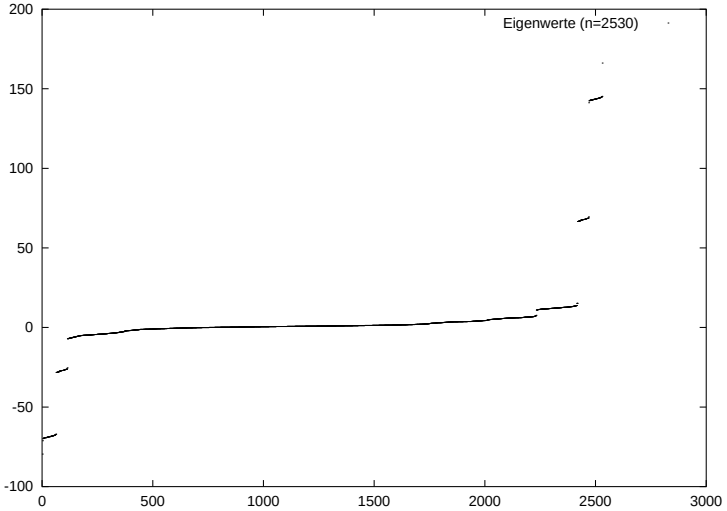


Abbildung 10: Spektrum der Testmatrix (n=2530)

Bei Skalierung der Rechnung mit n^3 dürfte der Zeitaufwand für die Berechnung der 2530er Matrix nur ca. $20\times$ so hoch sein wie für die 959er Matrix, er ist aber bei PDSYGVX mehr als $30\times$ so hoch!

Sequentielle Alternativen zur ScaLAPACK-Routine PDSYEVX

Vergleich verschiedener LAPACK-Routinen zur sequentiellen Lösung des verallgemeinerten Eigenwertproblems

Die beiden oben beschriebenen Methoden lassen sich unter Verwendung der genannten ScaLAPACK-Routinen als parallele Verfahren realisieren. Für kleine Matrizen, d. h. für Matrizen, die auf einem Prozessor verarbeitet werden können, sind alternativ zu den ScaLAPACK- die entsprechenden LAPACK-Routinen (DSYGVX, DSYEVX, DSYMM und DGEMM) einsetzbar.

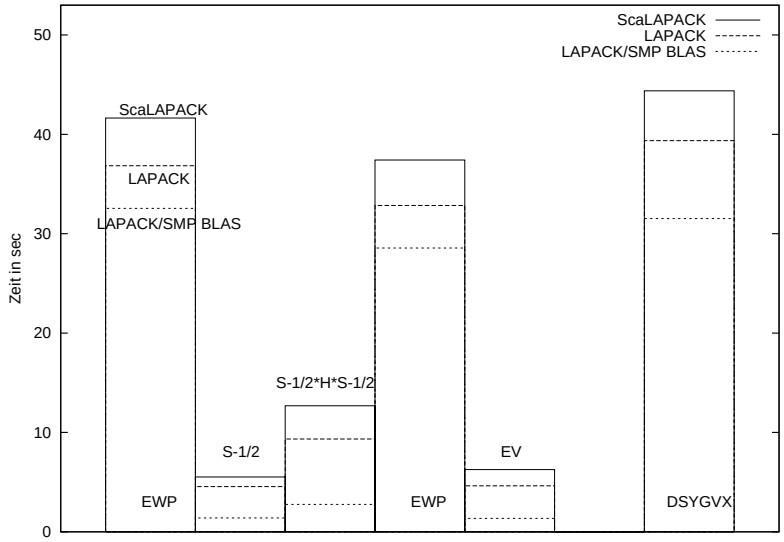


Abbildung 11: Vergleich von ScaLAPACK und LAPACK unter Verwendung der korrespondierenden Routinen (n=959)

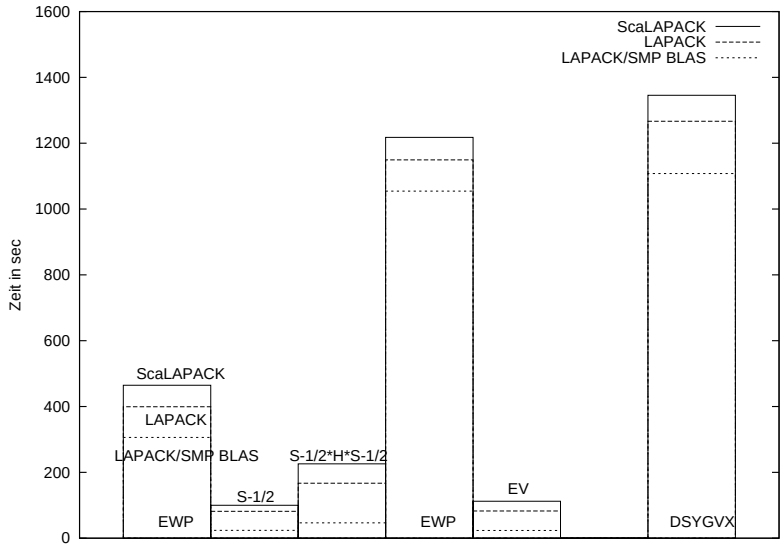


Abbildung 12: Vergleich von ScaLAPACK und LAPACK unter Verwendung der korrespondierenden Routinen (n=2530)

Des Weiteren wird die Shared Memory-parallelisierte BLAS-Bibliothek von ATLAS [10] angebunden;

dabei werden die BLAS auf den 4 Prozessoren eines Knotens parallel bearbeitet unter Verwendung des gemeinsamen Speichers. Für die BLAS 3-Routinen sollte sich hierdurch ein Zeitvorteil ergeben.

Anhand der Abb. 11 und 12 sind folgende Resultate zu erkennen: Sowohl durch die LAPACK-Routinen als auch durch den Einsatz von SMP BLAS kann jeweils eine Zeitverbesserung erzielt werden, so dass für sequentielle Verfahren die LAPACK- den ScaLAPACK-Routinen vorzuziehen sind sowie bei entsprechender Rechnerstruktur SMP BLAS eingesetzt werden sollten.

Man sieht, dass die Verwendung der SMP BLAS bei den BLAS-Operationen einen sehr großen Gewinn bringen, dagegen wirken sie sich bei der Lösung des Eigenwertproblems nur wenig aus. Hier ist evtl. durch eine explizite SMP-Parallelisierung mit OpenMP ein größerer Zeitgewinn zu erreichen.

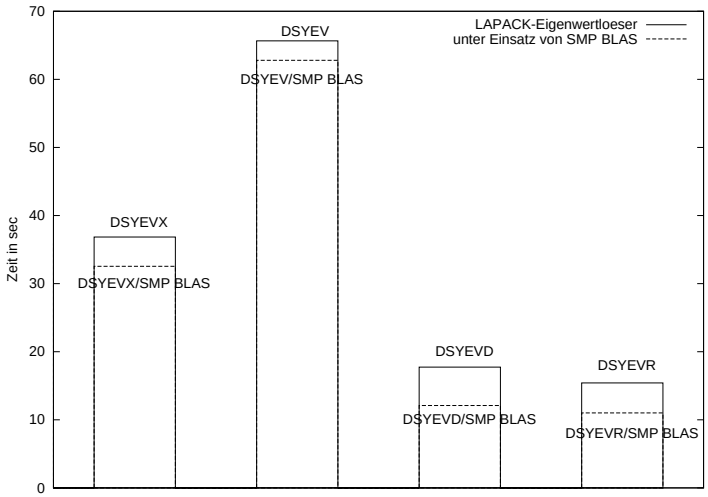


Abbildung 13: LAPACK-Eigenwertlöser im Vergleich (Berechnung von $S^{-\frac{1}{2}}$, n=959)

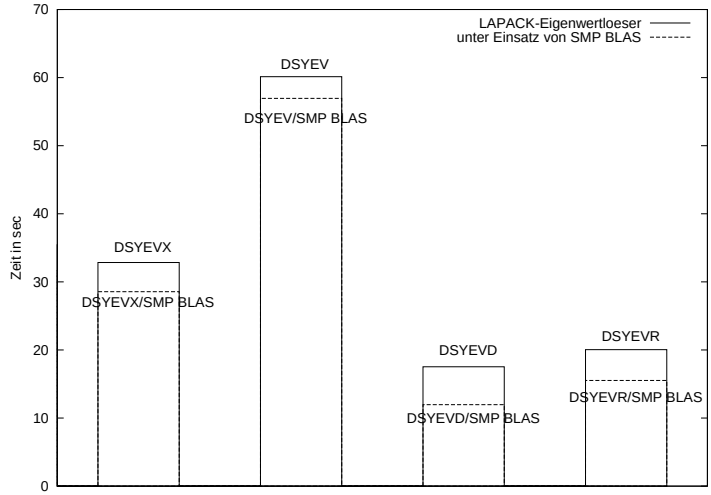


Abbildung 14: LAPACK-Eigenwertlöser im Vergleich (standardisiertes Eigenwertproblem, n=959)

Zur Lösung des Standardeigenwertproblems stellt die LAPACK-Bibliothek darüber hinaus weitere Routinen zur Verfügung [11]. Die im Weiteren vergleichend untersuchten Eigenwertlöser seien an dieser Stelle aufgeführt und kurz beschrieben:

- DSYEVX (expert driver):
  Bei den eingesetzten Verfahren handelt es sich hier um Bisektion und inverse Iteration.
- DSYEV (simple driver):

Die Routine basiert auf dem QR-Algorithmus.

- DSYEVD (divide-and-conquer driver):
  Zur Berechnung der Eigenvektoren wird ein divide and conquer-Algorithmus verwendet. Wegen Verwendung vieler BLAS 3-Routinen wird jedoch n^2 mehr Speicher als bei der Routine DSYEVX benötigt.
- DSYEVR (relatively robust representation (RRR) driver):
  Der Name RRR leitet sich aus dem zugrunde liegenden Algorithmus ab. Die Routine vermeidet weitgehend Gram-Schmidt-Orthogonalisierung, verwendet sonst wie die Routine DSYEVX Bisektion und inverse Iteration.

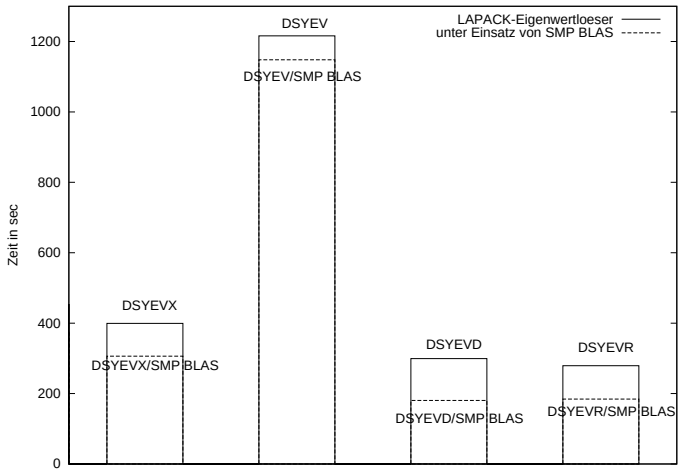


Abbildung 15: LAPACK-Eigenwertlöser im Vergleich (Berechnung von $S^{-\frac{1}{2}}$, n=2530)

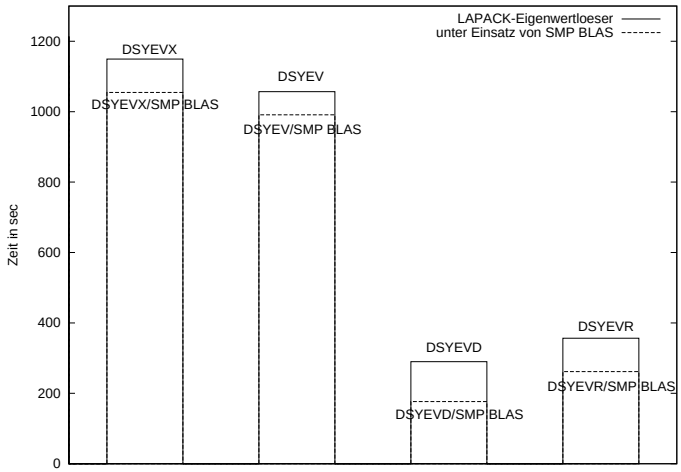


Abbildung 16: LAPACK-Eigenwertlöser im Vergleich (standardisiertes Eigenwertproblem, n=2530)

Wie in den Abb. 13, 14, 15 und 16 erkennbar ist, schneidet die Routine DSYEV in vielen Fällen schlechter ab als die Routine DSYEVX. Nur bei großer Cluster-Bildung, wie es beim Lösen des standardisierten Eigenwertproblems für $n = 2530$ der Fall ist, kann die Routine DSYEV bessere Ergebnisse erzielen. Deutlich schneller als diese beiden arbeiten jedoch die Routinen DSYEVD und DSYEVR.

Da der benötigte Speicherbedarf zur Bewertung der Güte einer Routine ebenfalls eine große Rolle spielt, wird hierauf in weiteren Betrachtungen eingegangen.

Zunächst soll anhand einer Auflistung des Mindestspeicherbedarfs angezeigt werden, welche Komponenten in welcher Größenordnung zum Gesamtspeicherbedarf beitragen:

	DSYEVX	DSYEV	DSYEVD	DSYEVR
Matrix	n^2	n^2	n^2	n^2
Eigenvektoren	n^2	-	-	n^2
Eigenwerte	n	n	n	n
WORK/IWORK	$8n/5n$	$3n-1$/ -	$1+6n+2n^2/3+5n$	$26n/10n$
IFAIL/ISUPPZ	n/ -	- / -	- / -	- / $2n$

Für optimale Berechnungszeiten bei den Routinen DSYEVX, DSYEV und DSYEVR wird jedoch mehr Arbeitsspeicher WORK benötigt. Dieser kann durch Speicherplatzanfragen ermittelt werden. Der tatsächlich benötigte Speicherbedarf ist in Abb. 17 graphisch dargestellt.

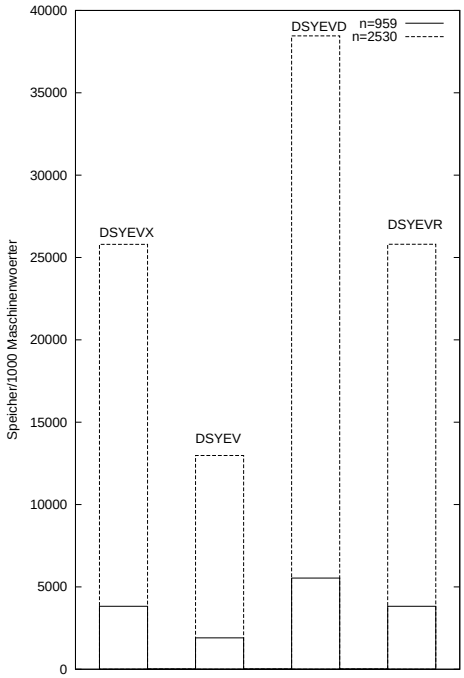


Abbildung 17: Speicherbedarf der LAPACK-Eigenwertlöser

Man erkennt, dass bei der Routine DSYEVD der Vorteil bei der Berechnungszeit durch einen deutlich größeren Speicherbedarf bezahlt werden muss, bei der Routine DSYEV sind ähnliche Beobachtungen in anderer Richtung zu machen.

Die Routine DSYEVR bietet dagegen eine sehr gute Alternative zu den anderen LAPACK-Eigenwertlösern, da sie sehr gute Ergebnisse hinsichtlich der Berechnungszeit mit akzeptablem Speicherbedarf verbindet. So steht DSYEVR zeitlich gesehen der divide and conquer-Routine DSYEVD in nichts nach, beansprucht aber gerade mal so viel Speicher wie die Routine DSYEVX.

Vergleichende Untersuchung der in Karlsruhe entwickelten Routine RDIAG

Die an der Universität Karlsruhe am Lehrstuhl für Theoretische Chemie entwickelte Routine RDIAG erwartet als Eingabe die Matrix des standardisierten Eigenwertproblems in gepackter Form sowie die Matrix, mit der die Standardisierung durchgeführt wurde, als $n \times n$-Matrix. Als Ausgabe werden Eigenwerte und Eigenvektoren des verallgemeinerten Eigenwertproblems geliefert. Zur vergleichenden Untersuchung mit den LAPACK-Routinen wird diese Routine mit folgender Ein- und Ausgabe verwendet:

Input: $\tilde{H} := S^{-\frac{1}{2}} H S^{-\frac{1}{2}}$ in gepackter Form,
$S^{-\frac{1}{2}}$ als $n \times n$-Matrix
Output: E, C (Eigenwerte und Eigenvektoren des Ausgangsproblems)

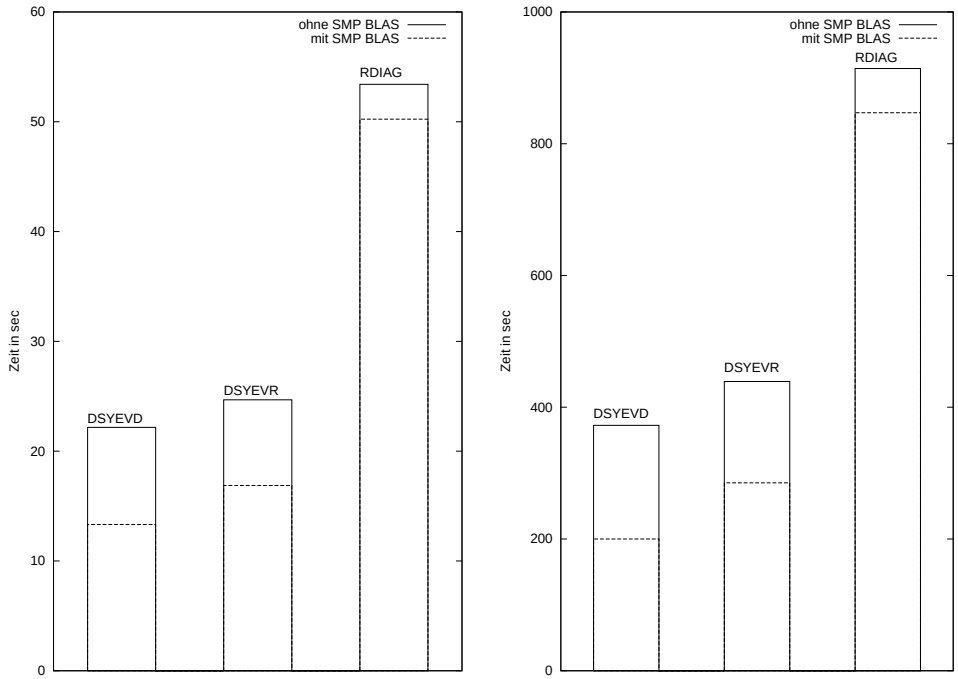


Abbildung 18: Zeitvergleich der Routine RDIAG mit den LAPACK-Eigenwertlösern DSYEVD und DSYEVR

Bei der Implementierung der Routine RDIAG wurde folgendes Verfahren zur Diagonalisierung verwendet:

1. Reduktion der symmetrischen Matrix $\tilde{H}$ mittels Householder-Transformationen auf Tridiagonalgestalt
2. Erzeugung der orthogonalen Transformationsmatrix
3. Berechnung der Eigenwerte und Eigenvektoren der Tridiagonalmatrix

Eine zusätzliche Matrixmultiplikation zur Berechnung der Eigenvektoren wird nicht benötigt; dies geschieht mittels Jacobi-Diagonalisierung mit Startmatrix $S^{-\frac{1}{2}}$ im Schritt 3 [12]. Dadurch kann die Routine RDIAG alternativ zur Lösung des standardisierten Eigenwertproblems sowie der Berechnung der Eigenvektoren des Ausgangsproblems eingesetzt werden.

In weiteren Untersuchungen wird die Routine RDIAG den LAPACK-Eigenwertlösern DSYEVD und DSYEVR gegenübergestellt, wobei beim Vergleich der Berechnungszeiten (vgl. Abb. 18) bei letzteren die zusätzlich benötigte Matrixmultiplikation berücksichtigt wird.

Klar zu sehen ist, dass die Routine RDIAG zeitlich gesehen mit den LAPACK-Routinen DSYEVD und DSYEVR nicht konkurrieren kann. Durch Verwendung der gepackten Form zur internen Repräsentation der Matrix können die BLAS 3-Routinen nicht verwendet werden, die jedoch entscheidend zur Geschwindigkeit beitragen.

Inwieweit tatsächlich Speicherplatz durch diese Darstellung eingespart werden kann, soll anhand folgender Aufstellung ermittelt werden:

	DSYEVD	DSYEVR	RDIAG
Matrix	n^2	n^2	$n(n+1)/2$
Eigenvektoren	-	n^2	n^2
Eigenwerte	n	n	n
WORK/IWORK	$1+6n+2n^2/3+5n$	$26n/10n$	$81n$/ -
IFAIL/ISUPPZ	- / -	- / $2n$	- / -

Graphisch ist der jeweilige Speicherbedarf der Routinen für die Matrixgrößen $n = 959$ und $n = 2530$ in Abb. 19 dargestellt.

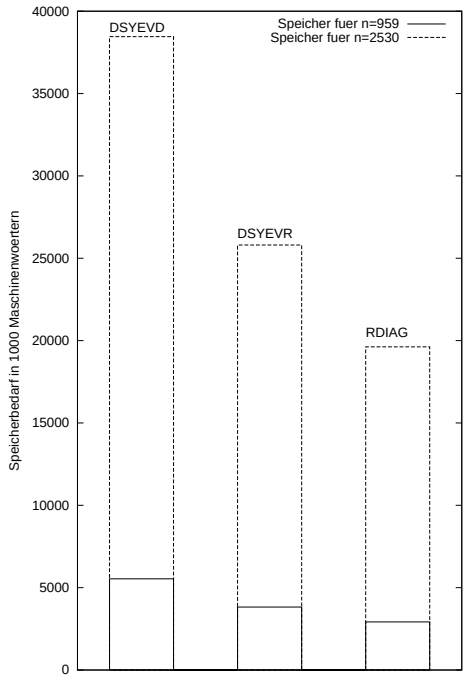


Abbildung 19: Vergleich des Speicherplatzes für die Routinen RDIAG, DSYEVD und DSYEVR

Unter Berücksichtigung der Ergebnisse hinsichtlich der Berechnungszeit ist der Speicherbedarf für RDIAG als nicht wesentlich geringer als der für die Routine DSYEVR einzuschätzen. Dass die Routine DSYEVD wesentlich schlechter abschneidet, war anhand vorheriger Ergebnisse abzusehen.

Zusammenfassung der Ergebnisse

Um die Resultate kurz zusammenzufassen, sei Folgendes gesagt:
Dadurch dass bei der Löwdin-Orthogonalisierung zwei Eigenwertprobleme auftreten, kann diese Herangehensweise zur Lösung des verallgemeinerten Eigenwertproblems nicht mit dem Verfahren konkurrieren, das Cholesky-Faktorisierung zur Standardisierung verwendet.

Darüber hinaus ist angesichts der guten Ergebnisse der Routine DSYEVR im sequentiellen Einsatz mit Bedauern festzuhalten, dass eine parallele Version noch nicht verfügbar ist. Für DSYEVD steht seit kurzem (31.8.2001) mit ScaLAPACK 1.7 eine parallele Version zur Verfügung, die jedoch auf ZAMpano noch nicht installiert ist.

Danksagung

An dieser Stelle möchte ich mich bei Herrn Prof. F. Hossfeld und Herrn Dr. R. Esser für die Organisation und Durchführung des Gaststudentenprogramms für Wissenschaftliches Rechnen am NIC/ZAM bedanken.

Für die Betreuung und Unterstützung während meines Aufenthaltes möchte ich mich recht herzlich bei meiner Betreuerin Frau I. Gutheil sowie Herrn Dr. J. Grotendorst bedanken, des Weiteren bei allen Mitarbeitern des Zentralinstituts für Angewandte Mathematik, die mir mit Rat und Tat zur Seite standen.

Literatur

1. D. R. Hartree; Proc. Cambridge Phil. Soc. 24, 89 (1928)
2. V. Fock, Z. Phys. 61, 126 (1930)
3. V. Fock, Z. Phys. 62, 795 (1930)
4. P. Hohenberg und W. Kohn, Phys. Rev. B 76, 6062 (1964)
5. W. Kohn und L. J. Sham, Phys. Rev. A 140, 1133 (1965)
6. B. N. Parlett, The Symmetric Eigenvalue Problem, Society for Industrial and Applied Mathematics, Philadelphia (1998)
7. J. H. Wilkinson and C. Reinsch, Handbook for Automatic Computation, Volume II, Lineare Algebra, (Springer Verlag 1971)
8. ZAMpano - ZAM Parallel Nodes http://zampano.zam.kfa-juelich.de
9. L. S. Blackford, J. Choi und andere, ScaLAPACK User's Guide, SIAM (1997)
10. ATLAS- Automatically Tuned Linear Algebra Software http://www.netlib.org/atlas
11. E. Anderson, Z. Bai und andere, LAPACK User's Guide, SIAM (1999)
12. R. Ahlrichs und K. Tsereteli, Efficient Linear Algebra Routines for Symmetric Matrices Stored in Packed Form (2001)

A Multiple-Time-Step-Algorithm for Molecular Dynamics Simulations

Andreas Nußbaumer

University of Leipzig

psy96cfc@studserv.uni-leipzig.de

Abstract: The critical factors that govern molecular dynamics simulations are time and memory consumption. For short range interactions a "cut off radius" may be used to speed up computation. Furthermore methods known as Multiple-Time-Step (MTS) methods are used which separate contributions to the force on different time scales. One of these is the method presented here. A Taylor expansion in the forces is carried out every n-th step for particles in a certain range. In the intermediate steps this expansion is used to predict positions and forces. Based on the work of Street and Tildesley and an example source code by Tildesley a module for the MD simulation program "DMMD" developed at the Central Institute for Applied Mathematics (ZAM) at the Forschungszentrum Jülich was implemented in Fortran 90. In order to test the correctness of the simulation, exhaustive tests were performed for a system of liquid argon on the ZAMpano cluster.

Introduction

Molecular Dynamics Simulations

Molecular dynamics (MD) simulations are ruled by the systems' Hamiltonian $\mathcal{H}$, which consists of the kinetic energy part, a potential energy part (most often approximated by a pair potential), and external contributions to the system energy. Based on this, the equations of motion

$$\dot{\vec{r}}_i = \frac{\partial \mathcal{H}}{\partial \vec{p}_i} \qquad \dot{\vec{p}}_i = -\frac{\partial \mathcal{H}}{\partial \vec{r}_i} \tag{1}$$

can be solved numerically.

In a MD simulation the particles are distributed in a certain geometrical region (often a cubic box) to which boundary conditions are applied, e.g. periodic or reflecting boundary conditions. The integrated Eqn. 1 gives positions and momenta of the particles as a function of time which results in a complete description of the physical system in a classical sense. With the help of statistical physics structural, dynamic, and thermodynamic properties can be calculated. Fig. 13 shows a scheme of a typical MD-simulation routine.

Ingredients

For a MD simulation there are three basic ingredients necessary: a model for the potential, an integrator and an ensemble.

- The **potential model** describes the interaction between atoms constituting the system. A typical example for a pairwise potential is the Lennard-Jones 12-6 effective pair potential (see Fig. 1) that may be used to describe a simple liquid, e.g. Argon:

$$\phi_{LJ}(r) = 4\epsilon \left(\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right) \tag{2}$$

The potential (see Fig. 1) has an attractive tail of the form $1/r^6$ representing dispersion energies, and a steeply rising repulsive part starting at $r \sim \sigma$ which models the hard core of the atom. Futhermore the potential exhibits a minimum at $r = 2^{1/6}\sigma$ of depth ϵ. Realistic parameters for a simulation of a liquid are e.g. in the case of Argon $\epsilon/k_B \approx 120$K and $\sigma \approx 0.34$ nm for Aragon which provide a reasonable agreement to the experimental data.

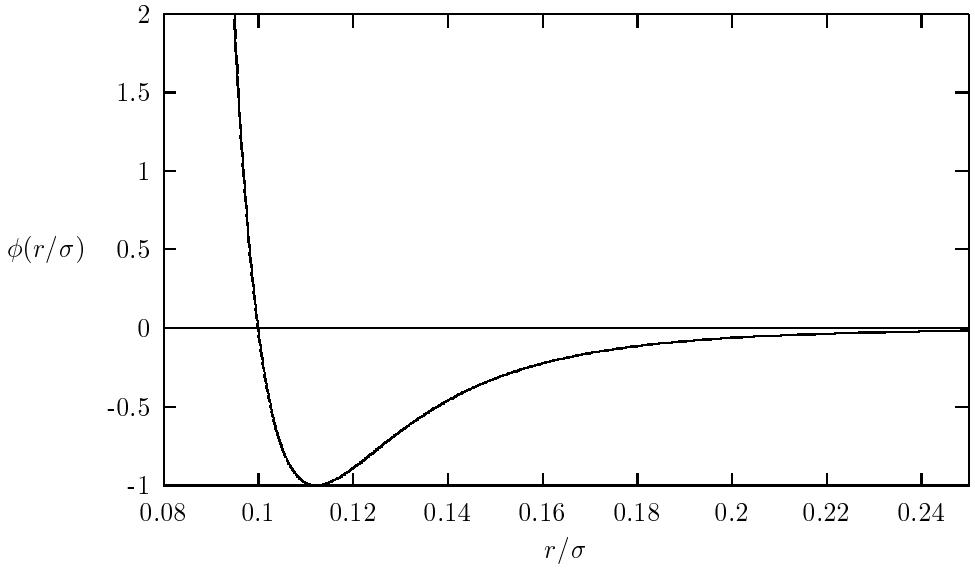


Figure 1: Lennard Jones potential in reduced units

- The **integrator** is a scheme to solve the equations of motion (1). A well known method is the *Verlet* Algorithm [3]. Thereby the Taylor expansion of a particle i at a position r_i at a time $t + h$ and $t - h$ is calculated:

$$\vec{r}_i(t+h) = \vec{r}_i(t) + \vec{v}_i(t)h + \frac{h^2}{2}\vec{a}_i(t) + \cdots \tag{3}$$

$$\vec{r}_i(t-h) = \vec{r}_i(t) - \vec{v}_i(t)h + \frac{h^2}{2}\vec{a}_i(t) - \cdots \tag{4}$$

where h is a small quantity which is chosen to ensure a stable integration. Adding both equations cancels the odd terms:

$$\vec{r}_i(t+h) + \vec{r}_i(t-h) = 2\vec{r}_i(t) + h^2\vec{a}_i(t) + \cdots \tag{5}$$

Solving for $\vec{r}_i(t+h)$ gives the Verlet-algorithm:

$$\vec{r}_i(t+h) = 2\vec{r}_i(t) - \vec{r}_i(t-h) + h^2\vec{a}_i(t) \tag{6}$$

Terms of the order of three and higher are neglected and the error is of the order of h^4. For $h \to 0$ Eqn. 6 converges and solves the equations of motion (1).

In Eqn. 6 there are no velocities, they are calculated by means of the central difference (error of the order of h^2):

$$\vec{v}_i(t) = \frac{\vec{r}_i(t+h) - \vec{r}_i(t-h)}{2h} \tag{7}$$

To calculate the velocity, the position at time $t+h$ is needed, therefore the position at time $t-h$ must be stored (see Eqn. 7). To come around that problem there is a slightly modified version of the Verlet-algorithm, the *velocity Verlet* algorithm.

$$\vec{r}_i(t+h) = \vec{r}_i(t) + h\vec{v}_i(t) + \frac{h^2}{2}\vec{a}_i(t) = \vec{r}_i(t) + h\vec{v}_i(t) + \frac{h^2}{2m_i}\vec{F}_i(t) \tag{8}$$

$$\vec{v}_i(t+h) = \vec{v}_i(t) + \frac{h}{2}\left(\vec{a}_i(t) + \vec{a}_i(t+h)\right) = \vec{v}_i(t) + \frac{h}{2m_i}\left(\vec{F}_i(t) + \vec{F}_i(t+h)\right) \tag{9}$$

Substituting v_i in the equations for $r_i(t+h)$ and $r_i(t-h)$ and adding both equations gives the Verlet-algorithm. Therefore Eqn. 8 and Eqn. 9 are equivalent to Eqn. 6 and Eqn. 7.

- For a closed physical system with no exchange of energy and matter the *micro canonical* **ensemble** or NVE-ensemble is chosen. In a MD simulation this situation is very easily realized: the number of particles doesn't change during the whole run and the Verlet algorithm guarantees the conservation of energy.

Theoretical Background

In the following the Lennard-Jones potential function is used as example. Moreover, the method described applies in the same way to any continuous, differentiable potential.

The force exerted by a molecule j on a molecule i is

$$\vec{F}_{ij} = -\nabla U(r_{ij}) = -\frac{\vec{r}_{ij}}{r_{ij}}\frac{d\phi(r_{ij})}{dr_{ij}} \tag{10}$$

where $\vec{r}_{ij} = \vec{r}_i - \vec{r}_j$ and $r_{ij} = |\vec{r}_{ij}|$. Therefore the total force on the molecule i exerted by all surrounding molecules $j \neq i$ is:

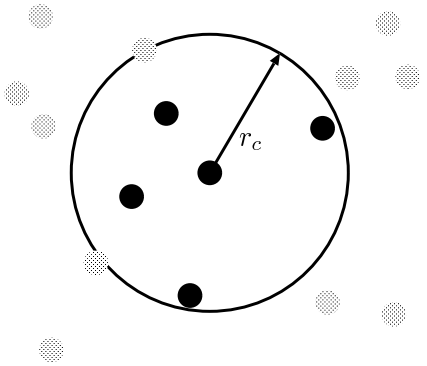
$$\vec{F}_i = \sum_{j \neq i} \vec{F}_{ij} = \sum_{j \neq i} -\frac{\vec{r}_{ij}}{r_{ij}}\frac{d\phi(r_{ij})}{dr_{ij}} \tag{11}$$

The introduction of a *cut-off radius* r_c gives:

$$\vec{F}_i = \sum_{j \neq i} \left(-\frac{\vec{r}_{ij}}{r_{ij}}\frac{d\phi(r_{ij})}{dr_{ij}}\right)_{r_{ij} < r_c} \tag{12}$$

The condition $r_{ij} < r_c$ means, that only particles j that are at least r_c away from the particle i contribute to the total force F_i (see Fig. 2). This is possible since only short range forces are taken into account that fall off rapidly.

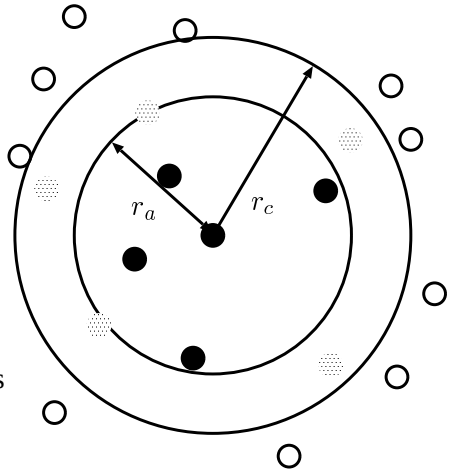
Figure 2: black: particles within the cut-off radius; grey: particles outside the cut-off radius



The MTS-method introduces a second sphere $r_a \leq r_c$ that divides the sphere r_c into a primary area with primary particles ($r_{ij} \leq r_a$), and a secondary area with secondary particles ($r_a \leq r_{ij} \leq r_c$). Then the total force on the particle i given by Eqn. 12, can be separated into a primary force $\vec{F}_{i_P}$ and a secondary force $\vec{F}_{i_S}$ produced by these two sets of particles:

$$\vec{F}_i = \vec{F}_{i_P} + \vec{F}_{i_S} = \sum_j \left(-\frac{\vec{r}_{ij}}{r_{ij}} \frac{d\phi(r_{ij})}{dr_{ij}} \right)_{r_{ij} \leq r_a} + \sum_k \left(-\frac{\vec{r}_{ik}}{r_{ik}} \frac{d\phi(r_{ik})}{dr_{ik}} \right)_{r_a \leq r_{ik} \leq r_c} \tag{13}$$

Figure 3: black: particles within the primary radius; grey: particles within the secondary radius; hollow: particles outside the cut-off radius



rag replacements

From the perception there is known that for short range forces holds:

$$\left\langle |\vec{F}_P| \right\rangle > \left\langle |\vec{F}_S| \right\rangle \tag{14}$$

$$\left\langle |\dot{\vec{F}}_P| \right\rangle > \left\langle |\dot{\vec{F}}_S| \right\rangle \tag{15}$$

Eqn. 14 means that on average the contributions to the forces form the secondary area are weaker than the ones from the primary area, while Eqn. 15 means that fluctuations in these contributions are weaker in the secondary area than in the primary one.

Due to the magnitude and the curvature of the Lennard-Jones potential these two properties seem to be fullfilled, i.e. the secondary forces are smaller and varying slower in time and distance. In Sec. and Sec.) this assumption will be prooven numerically.

These differences make it possible to distinct between two update mechanisms. Like in conventional MD simulations the primary forces are updated completely every time step while the secondary forces are only updated completely every n-th time step. In the $n-1$ remaining steps a $Taylor-update$ is performed.

At time t the primary and secondary forces on molecule i, $\vec{F}_{i_P}(t)$, $\vec{F}_{i_S}(t)$ are calculated from Eqn. 13 and a list of the corresponding particles is compiled. The time derivatives of the secondary forces $\dot{\vec{F}}_{i_S}(t)$, $\ddot{\vec{F}}_{i_S}(t)$, ... are calculated as well. This is done in the following way: with $A = -r^{-1}(d\phi/dr)$ Eqn. 10 gives

$$\vec{F} = -\frac{\vec{r}}{r}\frac{d\phi}{dr} = A\vec{r} \tag{16}$$

where $\vec{r} = \vec{r}_i(t) - \vec{r}_j$(t), $\dot{\vec{r}} = \dot{\vec{r}}_i(t) - \dot{\vec{r}}_j(t)$, ... Every term in 11 can be differentiated individually, and therefore the indices are omitted. Differentiating Eqn. 16 with respect to time gives:

$$\dot{\vec{F}} = A\dot{\vec{r}} + B(\vec{r}\cdot\dot{\vec{r}})\vec{r} \tag{17}$$

where $B = r^{-1}(dA/dr)$. The second derivative is:

$$\ddot{\vec{F}} = \left[B(\vec{r}\cdot\ddot{\vec{r}} + \dot{\vec{r}}\cdot\dot{\vec{r}}) + C(\vec{r}\cdot\dot{\vec{r}})^2\right]\vec{r} + 2B(\vec{r}\cdot\dot{\vec{r}})\dot{\vec{r}} + A\ddot{\vec{r}} \tag{18}$$

where $C = r^{-1}(dB/dr)$. If the Lennard-Jones potential (see Eqn. 2) is used, the expressions for A, B and C are:

$$A = -\frac{1}{r}\frac{d\phi}{dr} = 48\epsilon\left(\frac{\sigma^{12}}{r^{14}} - 0.5\frac{\sigma^6}{r^8}\right) \tag{19}$$

$$B = \frac{1}{r}\frac{dA}{dr} = 192\epsilon\left(-3.5\frac{\sigma^{12}}{r^{16}} + \frac{\sigma^6}{r^{10}}\right) \tag{20}$$

$$C = \frac{1}{r}\frac{dB}{dr} = 1920\epsilon\left(5.6\frac{\sigma^{12}}{r^{18}} - \frac{\sigma^6}{r^{12}}\right) \tag{21}$$

During the following time steps ($t+h_1$, $t+h_2$, ...) the index list (compiled in the first step) is used. Only the primary forces are recalculated. For the secondary forces a Taylor expansion in the forces F_{i_S} is used:

$$\vec{F}_S(t+h) = F_S(t) + \dot{\vec{F}}_S(t)\frac{h}{1!} + \cdots + \frac{d^m}{dt^m}\vec{F}_S(t)\frac{h^m}{m!} \tag{22}$$

Plugging Eqn. 16 to 18 into Eqn. 22 gives an expression in terms of A, B, C:

$$\begin{aligned}\vec{F}_S(t+h) &= A\vec{r} + \left[A\dot{\vec{r}} + B(\vec{r}\cdot\dot{\vec{r}})\vec{r}\right]h + \\ &\quad + \left[\left(B(\vec{r}\cdot\ddot{\vec{r}} + \dot{\vec{r}}\cdot\dot{\vec{r}}) + C(\vec{r}\cdot\dot{\vec{r}})^2\right)\vec{r} + 2B(\vec{r}\cdot\dot{\vec{r}})\dot{\vec{r}} + A\ddot{\vec{r}}\right]\frac{h^2}{2}\end{aligned} \tag{23}$$

After $n-1$ steps both the primary and secondary forces and Taylor-coefficients are recalculated. Due to the motion of particles across the primary and secondary radii the boundary between the two regions gets gradually distorted from its original shape. The average number of steps until a secondary particle intrudes into the primary region defines the maximum number of *Taylor steps.*

To calculate equilibrium properties like the average internal energy U and the average virial $\mathcal{W}$ the expressions must be separated into primary and secondary contributions. For the average of the internal energy and the virial (without MTS but with a cut-off-radius) holds:

$$U = \left\langle \sum_i \sum_{j>i} \phi(r_{ij}) \right\rangle_{r_{ij} \le r_c} \tag{24}$$

and

$$\mathcal{W} = \frac{1}{3} \left\langle \sum_i \sum_{j>i} r_{ij} \frac{d\phi(r_{ij})}{r_{ij}} \right\rangle \tag{25}$$

There are two possibilities to get an equivalent expression for the MTS-method: The properties are calculated only every n-th step when a full update is performed, or a Taylor expansion similar to Eqn. 22 is performed. The latter alternative requires expressions for the time derivatives in Eqn. 24 and Eqn. 25. They are:

$$\dot{\phi} = -A(\vec{r} \cdot \dot{\vec{r}}) \tag{26}$$

$$\ddot{\phi} = -B(\vec{r} \cdot \dot{\vec{r}})^2 - A(\vec{r} \cdot \ddot{\vec{r}} + \dot{\vec{r}} \cdot \dot{\vec{r}}) \tag{27}$$

$$\dddot{\phi} = -C(\vec{r} \cdot \dot{\vec{r}})^3 - 3B\left[(\vec{r} \cdot \dot{\vec{r}})(\vec{r} \cdot \ddot{\vec{r}} + \dot{\vec{r}} \cdot \dot{\vec{r}})\right] - A(\vec{r} \cdot \dddot{\vec{r}} + 3\dot{\vec{r}} \cdot \ddot{\vec{r}}) \tag{28}$$

and

$$\frac{d}{dt}\left(r\frac{d\phi}{dr}\right) = -X(\vec{r} \cdot \dot{\vec{r}}) \tag{29}$$

$$\frac{d^2}{dt^2}\left(r\frac{d\phi}{dr}\right) = -Y(\vec{r} \cdot \dot{\vec{r}})^2 - X(\vec{r} \cdot \ddot{\vec{r}} + \dot{\vec{r}} \cdot \dot{\vec{r}}) \tag{30}$$

$$\frac{d^3}{dt^3}\left(r\frac{d\phi}{dr}\right) = -Z(\vec{r} \cdot \dot{\vec{r}})^3 - 3Y\left[(\vec{r} \cdot \dot{\vec{r}})(\vec{r} \cdot \ddot{\vec{r}} + \dot{\vec{r}} \cdot \dot{\vec{r}})\right] - X(\vec{r} \cdot \dddot{\vec{r}} + 3\dot{\vec{r}} \cdot \ddot{\vec{r}}) \tag{31}$$

where $X = -r^{-1}(d\phi/dr)$, $Y = r^{-1}(dX/dr)$ and $Z = r^{-1}(dY/dr)$

Implementation

The presented MTS algorithm was implemented in the framework of the parallel MD program DMMD (see [1]). DMMD uses the velocity-Verlet algorithm as integrator (see Sec.). Fig. 14 shows the correspondence between the physical equations and the program functions. The main loop consists essentially of the routine calls to `motion(1)`, `interaction()` and `motion(2)`. Here `interaction()` is the force routine based on a *systolic loop*-algorithm that sends positions to the next processor and gets

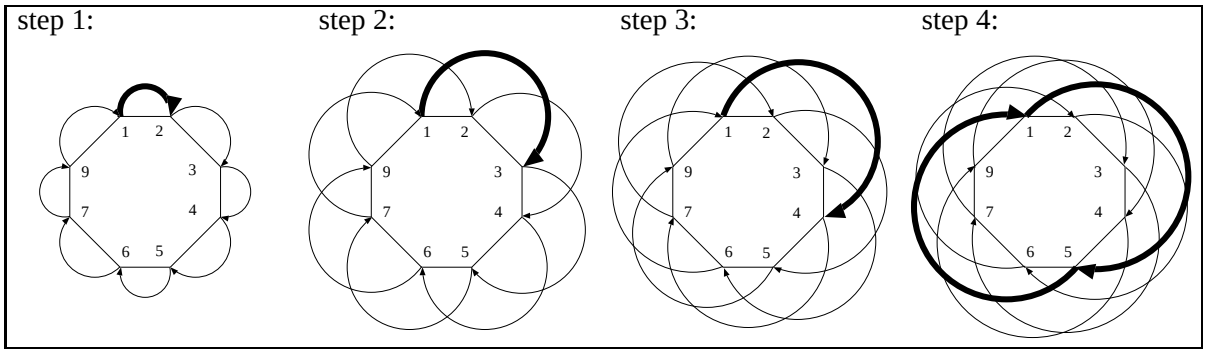


Figure 4: Schematic diagram of the positions sent in a systolic loop (forces are sent in the other direction)

forces in return (see Fig. 4). `Motion` is the integrator routine consisting of the two parts of the velocity-Verlet algorithm.

The main part of the MTS-algorithm is located in the module `mts_module` in the file `mts.F90`. A few global variables must be introduced, most of them in module `mts_parameters` in the file `variables.F90`.

The most prominent change in the existing source code is the introduction of a `switch`-statement in the subroutine `interaction(...)` in `interaction.F90`. A call to `interaction(1)` calculates the primary and the secondary forces, a call to `interaction(2)` the Taylor coefficients. Fig. 15 shows the MTS version of Fig. 14. The secondary forces and Taylor coefficients are recalculated only every n-th step.

The implementation and testing of the algorithm were done on the ZAMpano SMP cluster (see [6]) located at the NIC/ZAM. ZAMpano is a Linux based PC-cluster with 32 Pentium processors.

Results

Free Parameters

All in all there are four free parameters in the MTS method:

1. the cut-off radius r_c
2. the Taylor radius r_a
3. the number of Taylor steps n
4. the order of the Taylor expansion $O(F_{i_S})$

Besides item 4 that is essentially fixed in the source code there are still three parameters to determine.

Primary and Secondary Forces

One of the main assumptions of the MTS-method is the lower magnitude of the secondary forces (see Eqn. 14). In Fig. 5 these two contributions are compared for the case $r_a = 0.7 \cdot r_c$ and it is found that the magnitude of the secondary forces is about an orders of magnitude smaller than the primary forces (note that the scales are shifted for clarification).

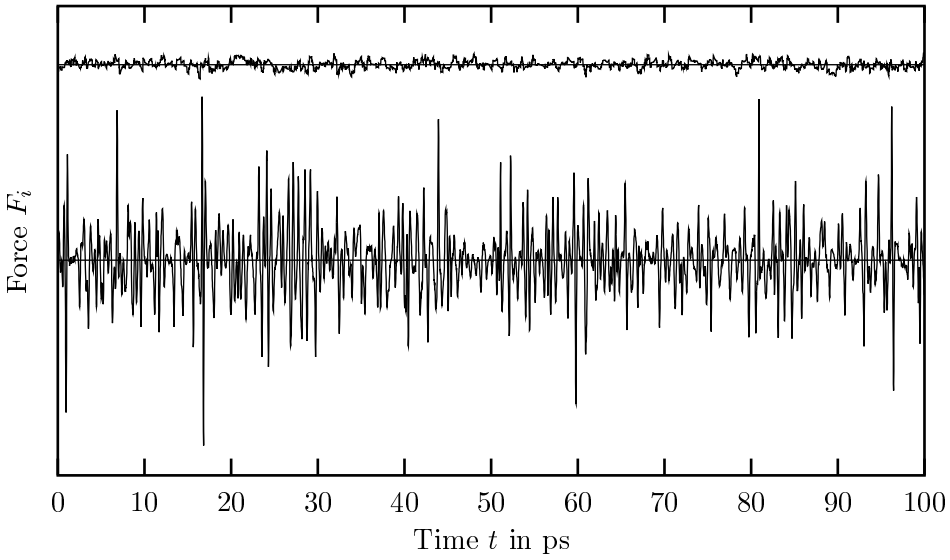


Figure 5: Comparison of primary forces (lower plot) with secondary forces (upper shifted plot). $r_a = 0.7 \cdot r_c$

If r_a gets bigger the magnitude of the secondary forces should get smaller. Fig. 6 shows a comparison of the secondary forces for three different radii. It is obvious that the Lennard-Jones potential has this property.

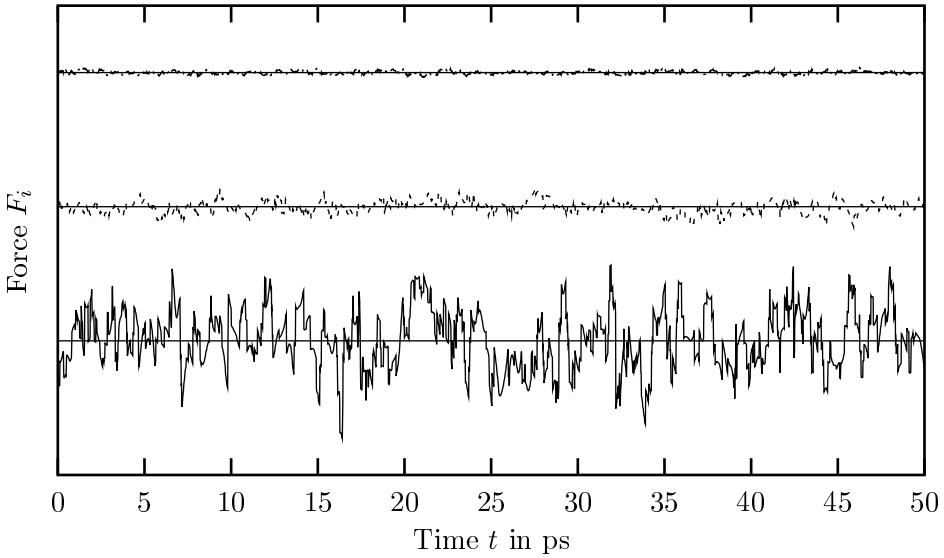


Figure 6: Comparison of secondary forces for different radii r_a. Upper (shifted) plot: $r_a = 0.9 \cdot r_c$, plot in the middle (shifted): $r_a = 0.7 \cdot r_c$, lower plot: $r_a = 0.5 \cdot r_c$

Spectral Properties

The second assumption made in Eqn. 15 was about the lower fluctuations in the secondary radius. Fig. 7 compares the fluctuation of primary and secondary forces. The primary forces have a maximum fluctuation frequency at about $10\,\mathrm{ps}^{-1}$ while the secondary forces have their maximum (as expected) at the low end of the spectrum.

In Fig. 8 the power spectrum of the secondary forces is plotted for different radii. With increasing radius r_a the maximum is shifted to the low end of the spectrum.

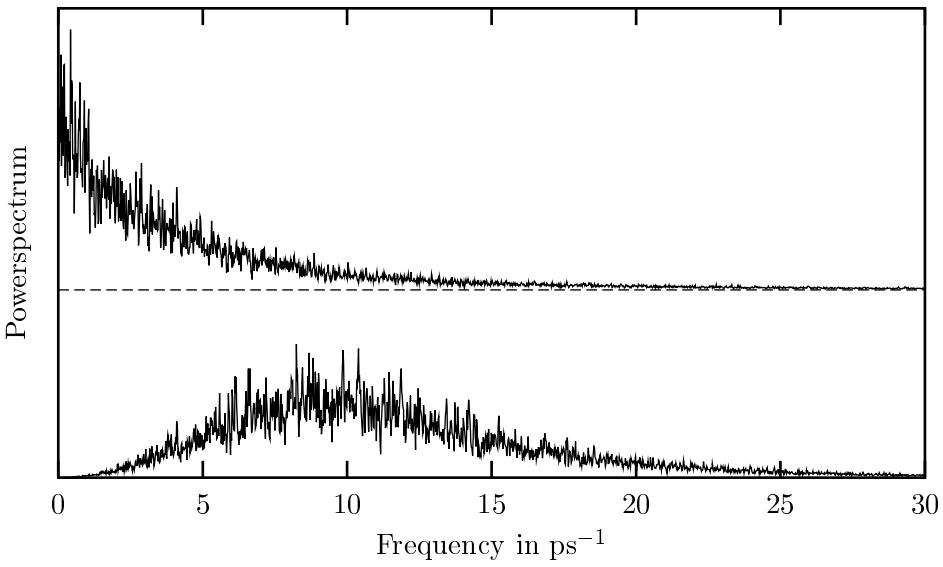


Figure 7: Comparison of the power spectrum of the primary force (lower plot) with the secondary force (upper shifted plot). $r_a = 0.7 \cdot r_c$

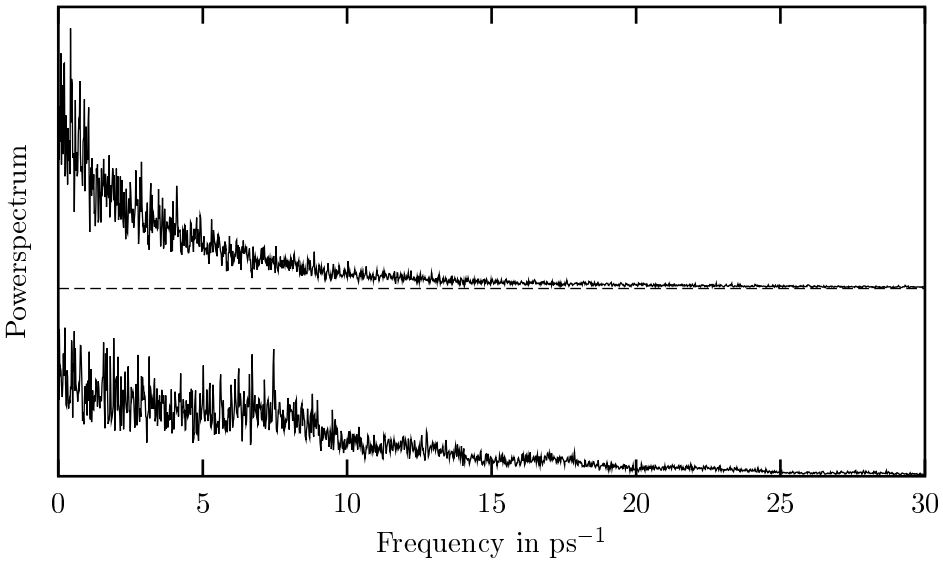


Figure 8: Comparison of the power spectrum of the secondary force for different radii. Lower plot: $r_a = 0.5 \cdot r_c$, upper (shifted) plot: $r_a = 0.7 \cdot r_c$

Energy Loss

The integrator solves the equation of motion. In an NVE-ensemble, energy is conserved and solutions of the differential equation should conserve energy as well. This leads to the requirement that an MTS-algorithm must conserve energy to be useful. However a numerical integration of the equation of motion shows that the present form of the MTS-algorithm is not energy conserving, i. e. a thermostat has to be used in order to compensate an energy drift. Fig. 9 shows three different scenarios: first without any heating, second a Berendsen weak coupling thermostat is applied with coupling constant $4.0 \cdot 10^{-4}$, third velocity rescaling is used to keep the temperature constant.

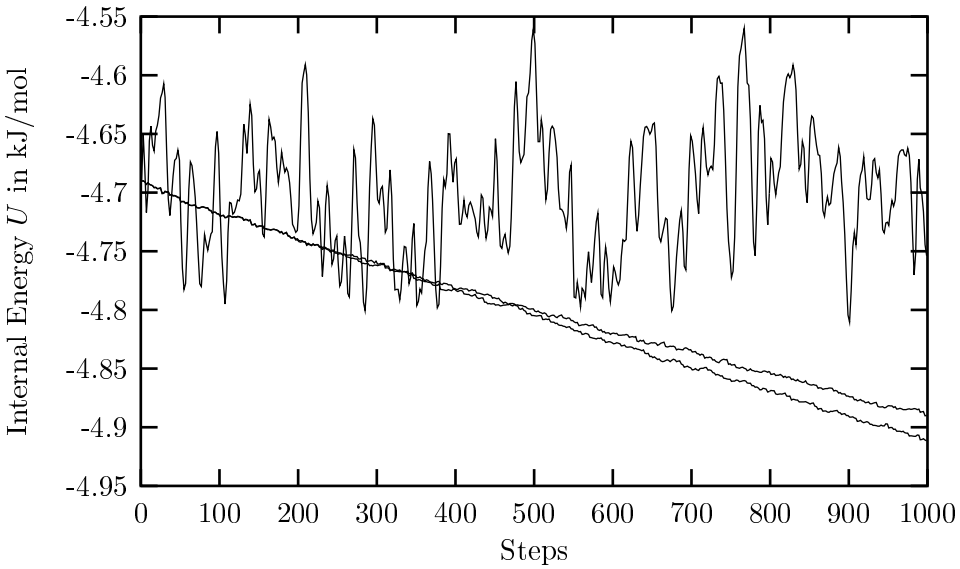


Figure 9: Comparison of different heating strategies. Lower straight line: no heating, upper straight line: weak heating, zigzag line: strong heating.

There is at least a weak heating necessary to stop the system from freezing out. Fig 10 shows the use of weak heating with different coupling coefficients. The higher the coefficient, the lower is the influence on the system. For all three coefficients the system finds a new equilibrium energy that is little lower than the original one.

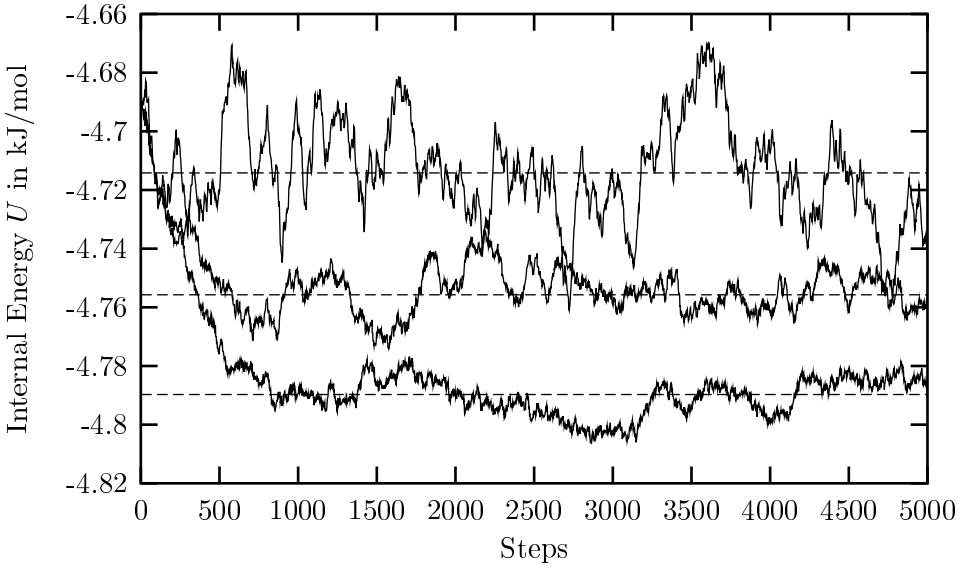


Figure 10: Comparison of different Berendsen weak coupling coefficients. From top to bottom: $2 \cdot 10^{-2}$, $1 \cdot 10^{-2}$, $0.67 \cdot 10^{-2}$.

A possible explanation for this behaviour is schematically described in Fig. 11. Considering a particle i that moves during the Taylor-expansion steps from the secondary into the primary area, its force contribution will be approximated by the secondary force and the Taylor-coefficients. As found in numerical studies this approximation is sometimes a crude one leading to a difference between the extrapolated force and a *potentially calculated* real one. This difference in force, going in line with a difference in energy results in an energy loss at a complete Taylor step (when all forces are calculated explicitly) and thus leading to the observed drift in energy.

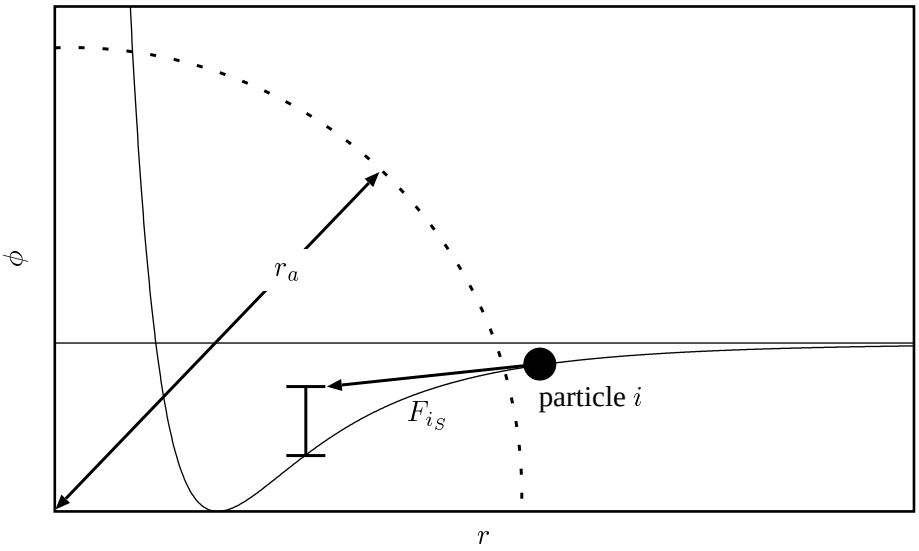


Figure 11: Lennard-Jones potential with Taylor radius r_a (dashed line). A particle i with the secondary force F_{i_S} is moving form the secondary area into the primary area.

Conclusion and Prospects

The MTS-method in the existing from is not energy conserving. Therefore the constant influence of a (weak) heating thermostat is necessary. Nevertheless is it possible to sample the configuration space faster with a thermostat which may be useful for equilibration runs.

A treatment of the energy problem is the introduction of a weighting function, leading to a smooth crossover of force contributions from particles which move from the secondary to the primary area. The idea is to weight the forces and Taylor-coefficients with the follow function:

$$w(r_{ij}) = \begin{cases} 1 & : \quad r_{ij} \leq r_a \\ 1 - \dfrac{(r_{ij} - r_a)^2 (3r_c - r_a - 2r_{ij})}{(r_c - r_a)^3} & : \quad r_a \leq r_{ij} \leq r_c \\ 0 & : \quad r_{ij} \geq r_c \end{cases} \tag{32}$$

The plot of Eqn. 32 and of the inverse equation (dashed) is shown in Fig. 12. To make a smooth crossover the primary forces have to be multiplied with $w(x)$ and the secondary forces (including the Taylor forces) with $1 - w(x)$. This only needs to be done in the area where $w(x) \neq 1$.

The introduction of this type of weighting function should result in a smaller energy gap and therefore better energy conservation.

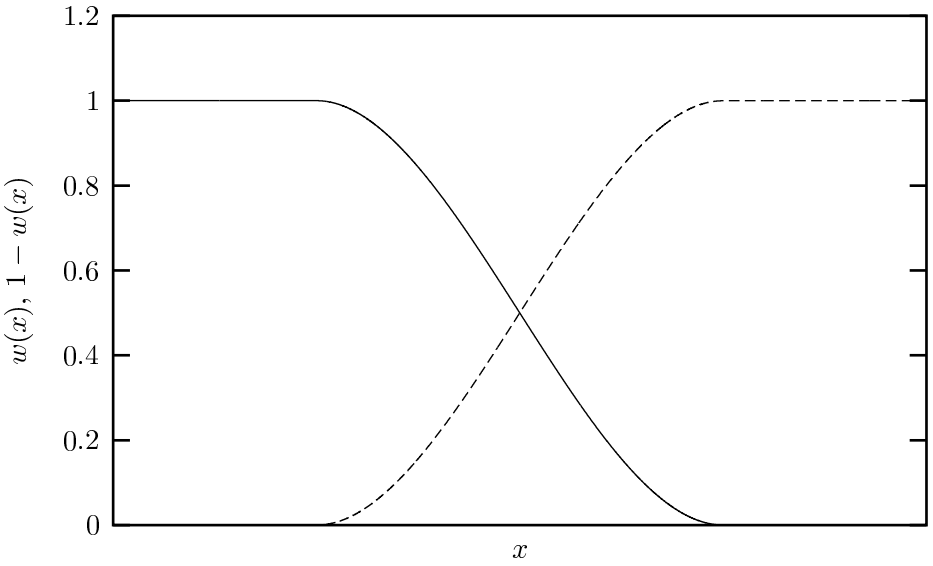


Figure 12: Plot of the weighting function $w(x)$ (Eqn. 32) and $1 - w(x)$ (dashed).

Acknowledgments

The work presented here was carried out during the ten week long "Guest Student Program" at the NIC/ZAM. I would like to thank Prof. Dr. Hoßfeld and Dr. Esser for inviting me to the program and the CCP 2001 and for the financial support. Furthermore I want to thank all the staff of the ZAM for providing excellent working conditions. Finally I'm grateful to my supervisor Dr. G. Sutmann. The discussions with him helped me to understand the physics behind MD and MTS. Without his active part in the debugging process the MTS code would still be a fragment.

References

1. N. Attig, M. Lewerenz, G. Sutmann, R. Vogelsang. Molecular Dynamics Algorithms for Massively Parallel Computers. 1999
2. W. B. Street, D. J. Tildesley. Multiple time-step methods in molecular dynamics.Mol. Phys. **35**, 1978
3. L. Verlet. Computer 'experiments' on classical fluids. Phys. Rev. **165**, 1968
4. R. Haberland, S. Fritzsche, G. Peinel, K. Heinzinger. Molekulardynamik - Grundlagen und Anwendungen. Vieweg 1995
5. O. Steinhauser. Reaction field simulation of water. Mol. Phys. **45**, 1981
6. http://zampano.zam.kfa-juelich.de/

Nassi-Shneiderman Diagrams

rag replacements

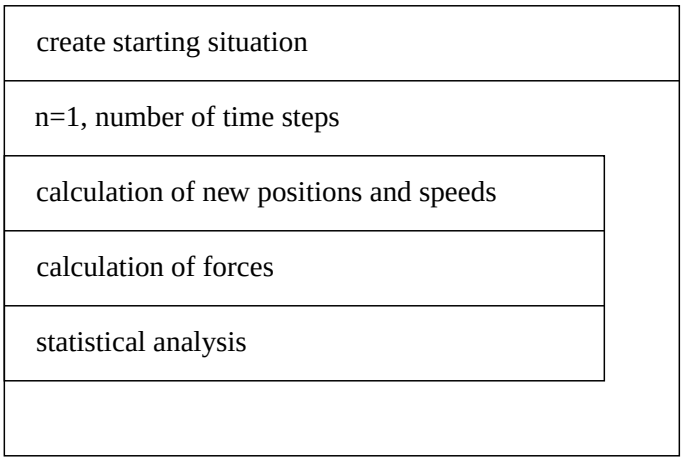


Figure 13: Nassi-Shneiderman diagram of an MD simulation run

rag replacements

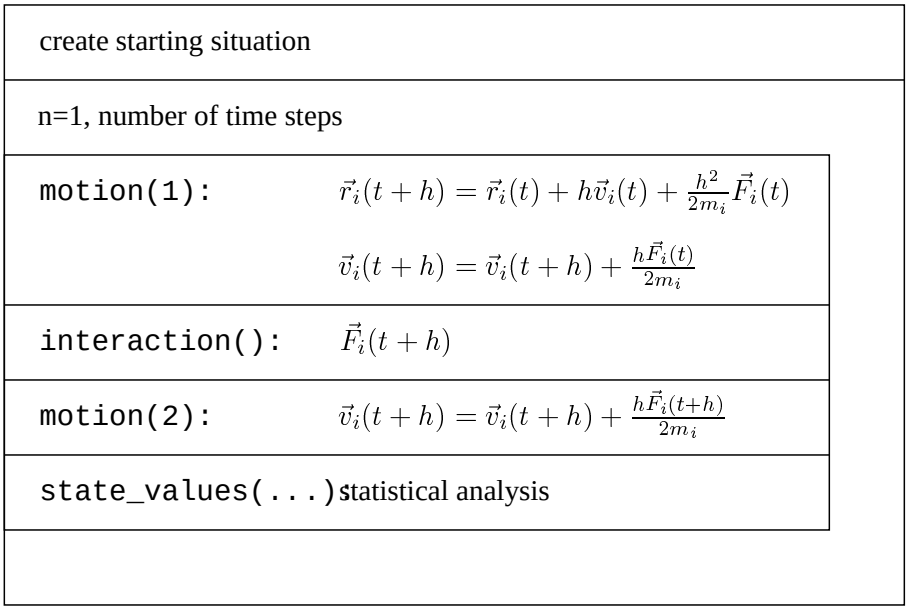


Figure 14: Nassi-Shneiderman diagram of an DMMD simulation run

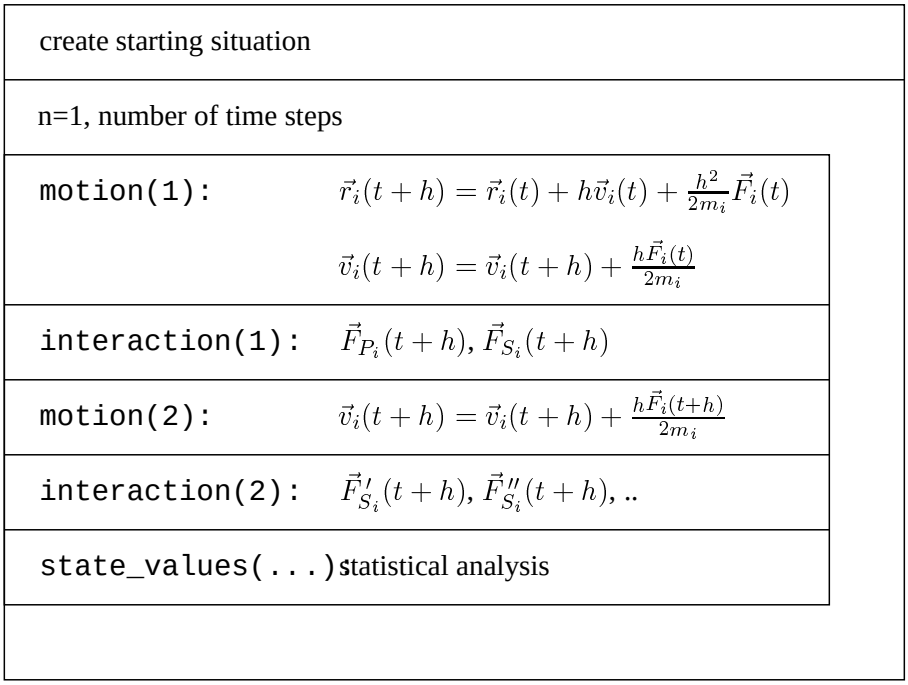


Figure 15: Nassi-Shneiderman diagram of an MTS-MD simulation run

Bestimmung von Eigenwerten im Inneren des Spektrums mit Hilfe von Teilraumverfahren

Gerald Schubert

Physikalisches Institut, Theoretische Physik I
Universität Bayreuth

E-mail: gerald.schubert@stud.uni-bayreuth.de

Zusammenfassung: Bei der numerischen Berechnung von Eigenwerten großer, dünnbesetzter symmetrischer Matrizen gibt es eine Vielzahl von Algorithmen, die dafür geeignet sind, Eigenwerte an den Rändern des Spektrums zu berechnen. Interessiert man sich dagegen für Eigenwerte im Inneren des Spektrums, so sind die zur Verfügung stehenden Methoden rar. Die Entwicklung des Jacobi-Davidson-Verfahrens mit Residuumsminimierung, sowohl mit harmonischen als auch normalen Ritz-Werten, war ein großer Fortschritt auf diesem Gebiet. Für die spezielle Klasse der in der theoretischen Festkörperphysik vorkommenden Anderson-Matrizen zeigt sich, daß aufgrund der spezifischen Eigenschaften des Spektrums die meisten Algorithmen zur Berechnung innerer Eigenwerte versagen oder nur nicht zufriedenstellende Ergebnisse liefern. Die Implementierung mit harmonischen Ritz-Werten scheint vielversprechend zu sein, obwohl auch hier Probleme auftreten, falls die inneren Eigenwerte nicht verteilt sind, sondern in Form von Clustern auftreten. Die unmittelbare Nähe der Eigenwerte innerhalb dieser Cluster zueinander führt dazu, daß man in irgendeiner Iteration zufällig als Shift genau einen Eigenwert trifft, was das harmonische Jacobi-Davidson-Verfahren zum Zusammenbrechen veranlaßt. In dieser Arbeit sollen unter anderem Möglichkeiten aufgezeigt werden, wie dies verhindert werden kann.

Einleitung

In der theoretischen Festkörperphysik treten bei der Beschreibung von komplexen Systemen mit vielen Freiheitsgraden, genauso wie auch in der Quantenchemie, Hilberträume auf, die sehr hochdimensional, in vielen Fällen sogar unendlichdimensional sind. Will man nun innerhalb dieser Hilberträume numerische Simulationen durchführen, so muß man die auftretenden Operatoren geeignet endlichdimensional approximieren. Da die darstellenden Matrizen dieser Operatoren glücklicherweise meist dünn besetzt sind, ist der Rechenaufwand in späteren Operationen gegenüber voll besetzen Matrizen erheblich reduziert. Daher können je nach verfügbarer Rechnerkapazität Matrizen mit Dimension von $10^5 \times 10^5$ bis $10^6 \times 10^6$, in Spezielfällen bis $10^8 \times 10^8$ behandelt werden, was für den größten Teil der Systeme eine sehr gute Näherung darstellt.
Trotz der sehr hohen Rechenleistung moderner Supercomputer ist es nicht möglich, alle Eigenwerte dieser Matrizen zu berechnen, was allerdings auch nicht nötig ist. Man interessiert sich vielmehr für eine geringe Anzahl (5 bis 20) von Eigenwerten, entweder an den Rändern des Spektrums, oder nun vermehrt auch innerhalb eines gewissen Intervalls im Inneren des Spektrums. Zur näherungsweisen Berechnung der gesuchten Eigenwerte und Eigenvektoren bieten sich aufgrund der großen Dimensionen der betrachteten Matrizen Teilraumverfahren an, bei denen man nicht den gesamten hochdimensionalen Raum, sondern nur einen niedrigdimensionalen Unterraum betrachtet und in diesem das Eigenwertproblem löst.

Durch geschickte Wahl dieser Unterräume kann man eine gute Konvergenz gegen die Eigenpaare der ursprünglichen Matrix erreichen. Während es für die extremen Eigenwerte eine ganze Anzahl verschiedener Algorithmen zur Berechnung gibt, treten im Inneren des Spektrums einige Probleme auf.
Im Folgenden wird dargelegt, wie man mit Hilfe einer Kombination verschiedener Verfahren die Eigenwerte einer sogenannten Anderson-Matrix berechnet. Die besondere Schwierigkeit bei dieser Matrix liegt in ihrer Indefinitheit und in der je nach gewähltem Parameter auftretenden Clusterbildung der Eigenwerte. Elsner et al. [3] haben in ihrem Artikel gezeigt, daß die bisher verwendeten Lösungsmethoden wie Arnoldi-Methode oder Lanczos-Algorithmus für diese Art von Problemen aufgrund hoher Rechenzeit oder Speicherbedarf ungeeignet sind, wobei eine Modifikation der Implementierung des Lanczos-Algorithmus von Cullum und Willoughby [11] noch die besten Ergebnisse liefert. Hier soll nun anhand eines Programms von A. Göke [1] geprüft werden, ob die Verwendung anderer Lösungsmethoden, insbesondere des harmonischen Jacobi-Davidson-Verfahrens zu einem besseren und schnelleren Konvergenzverhalten führt.

Theoretische Grundlagen

Jacobi-Davidson-Verfahren

Das Jacobi-Davidson-Verfahren ist ein iteratives Teilraumverfahren, das sich besonders gut zur Eigenwert- und Eigenvektorberechnung von großen, dünnbesetzten Matrizen eignet. In der ursprünglichen Konzeption dient dieses Verfahren allerdings nur zur Berechnung der extremen Eigenwerte. Es gliedert sich in zwei Teilabschnitte:
Im ersten Schritt werden innerhalb eines niedrigdimensionalen Unterraums Näherungen an die Eigenwerte und Eigenvektoren bestimmt, und danach wird in einem zweiten Schritt der Unterraum geeignet erweitert.
Man sucht also zur reell symmetrischen Matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ die s größten Eigenwerte λ_i und die zugehörigen Eigenvektoren $\mathbf{x}_i \in \mathbb{R}^n$, so daß

$$\mathbf{A}\mathbf{x}_i = \lambda_i \mathbf{x}_i \qquad \forall \mathrm{i} = 1, ..., \mathrm{s} \tag{1}$$

indem man einen niedrigdimensionalen Unterraum $\mathcal{V}^m \subset \mathbb{R}^n$ betrachtet, und in diesem möglichst gute Paare $(\theta_i, \mathbf{u}_i) \in (\mathbb{R} \times \mathcal{V}^m)$ mit

$$\mathbf{A}\mathbf{u}_i - \theta\mathbf{u}_i \perp \mathcal{V}^m \tag{2}$$

bestimmt. Dies geschieht durch orthogonale Projektion der Matrix $\mathbf{A}$ auf den Unterraum $\mathcal{V}^m$ mittels der orthonormalen Matrix $\mathbf{V}^m \in \mathbb{R}^{n\times m}$, deren Spalten die orthonormierten Basisvektoren $(\mathbf{v}_1, ..., \mathbf{v}_m)$ von $\mathcal{V}^m$ sind. Von der sich so ergebenden Matix

$$\mathbf{H} = (\mathbf{V}^m)^T \mathbf{A}\mathbf{V}^m \tag{3}$$

berechnet man dann die Eigenwerte, Ritz-Werte genannt, die Approximationen an die Eigenwerte von $\mathbf{A}$ sind. Multipliziert man die orthonormierten Eigenvektoren $\mathbf{y}_i \in \mathbb{R}^m$ der Matrix $\mathbf{H}$ mit $\mathbf{V}^m$, so erhält man die Ritz-Vektoren $\mathbf{u}_i := \mathbf{V}^m\mathbf{y}_i$, die eine Näherung an die Eigenvektoren von $\mathbf{A}$ sind. Die Approximationen sind umso besser, je kleiner der Winkel zwischen dem exakten Eigenvektor und dem Unterraum $\mathcal{V}^m$ ist [2]. Mit der Definition des Residuums $\mathbf{r}_i := \mathbf{A}\mathbf{u}_i - \theta_i\mathbf{u}_i$ für $\mathbf{u}_i \in \mathcal{V}^m$ ergibt sich als charakteristische Eigenschaft für die Ritz-Vektoren:

$$\mathbf{r}_i \perp \mathbf{u}_j \quad \text{für} \quad 1 \le i, j \le m \tag{4}$$

$$\mathbf{u}_i \perp \mathbf{u}_j \quad \text{für} \quad 1 \le i, j \le m, i \ne j \tag{5}$$

$$\tag{6}$$

Zur Erweiterung des Suchraums verwendet man die Korrekturvektoren $\mathbf{t}_i$, die sich als Lösung der von Sleijpen und van der Vorst vorgeschlagenen Korrekturgleichung $\mathbf{Q}(\mathbf{A} - \theta_i \mathbf{I})\mathbf{Q}\mathbf{t}_i = -\mathbf{r}_i$ mit

$$\mathbf{Q} := \mathbf{I} - \sum_{j=1}^{s} \mathbf{u}_j \mathbf{u}_j{}^T$$

ergeben [4].
Prinzipiell läßt sich dieses Verfahren (Ritz-Verfahren) auch auf innere Eigenwerte anwenden. Allerdings tritt hier die Komplikation auf, daß Ritz-Werte auftreten können, die scheinbar einen Eigenwert approximieren, aber deren zugehörige Ritz-Vektoren nur sehr kleine Anteile in Richtung der gesuchten Eigenvektoren haben. In solchen Fällen spricht man von "Phantomwerten" oder "ghostvalues". Dieser Effekt ist darauf zurückzuführen, daß die Ritz-Werte Konvexkombinationen der Eigenwerte sind, und tritt demnach nur für innere Eigenwerte auf [5]. Eine weitere mögliche Komplikation tritt auf, wenn zwei Ritz-Werte den gleichen Eigenwert approximieren, da dann die zugehörigen Ritz-Vektoren zueinander einen kleinen Winkel einschließen müßten, was im Widerspruch zu (5) steht.

Harmonische Ritz-Werte

Anders als im gewöhnlichen Ritz-Verfahren betrachtet man im harmonischen Jacobi-Davison-Verfahren nicht die Matrix $\mathbf{A}$, sondern ihre Inverse $\mathbf{A}^{-1}$ bzw $(\mathbf{A} - \mu\mathbf{I})^{-1}$. Man nennt dann $\tilde{\theta} \in \mathbb{R}$ einen harmonischen Ritz-Wert von $\mathbf{A}$, wenn $\tilde{\theta}^{-1}$ Ritz-Wert von $\mathbf{A}^{-1}$ ist. Durch diese Betrachtungsweise vertauschen $\mathcal{V}^m$ und $\mathcal{W}^m = \mathbf{A}\mathcal{V}^m$ die Rollen. Da die Berechnung der Inversen viel Rechenaufwand erfordert, will man dieses gerne vermeiden, und verwendet deshalb $\mathbf{A}^{-1}$ nur implizit. Dies ist möglich, setzt allerdings voraus, daß man für $\mathcal{W}^m$ eine Orthonormalbasis berechnet hat. Weiterhin gilt jetzt die Orthogonalität nicht mehr für die harmonischen Ritzvektoren $\tilde{\mathbf{u}} \in \mathcal{V}^m$, sondern für $\hat{\mathbf{u}} := \mathbf{A}\tilde{\mathbf{u}} \in \mathcal{W}^m$

Residuumsminimierung

Wenn man Eigenwerte λ_i im Inneren des Spektrum innerhalb eines vorgegebenen Intervalls $[\mu_l, \mu_r]$ sucht, kann man analog zum Ritz-Verfahren vorgehen, nur daß man nicht die s extremalen Ritz-Werte auswählt, sondern die s Werte, die am nächsten am Shift $\mu := \frac{1}{2}(\mu_l + \mu_r)$ liegen. Aufgund der Eigenschaft, daß die Ritz-Werte Konvexkombinationen der Eigenwerte sind, ist es jedoch jetzt möglich, daß Phantomwerte auftreten oder daß zwei Ritz-Vektoren denselben Eigenwert approximieren.
Um diese Probleme zu vermeiden gibt es zwei verschiedene Strategien:
Im Fall der Phantomwerte kann man an den Ritz-Werten nicht erkennen, ob es sich um einen "normalen" Ritz-Wert oder um einen Phantomwert handelt. Allerdings hat der Ritz-Vektor, der zu einem Phantomwert gehört, eine wesentlich größere Residuumsnorm. Eine mögliche Vorgehensweise, um s Ritzwerte zu erhalten, unter denen sich mit hoher Wahrscheinlichkeit kein Phantomwert befindet, kann wie folgt aussehen [1]:

- Sortiere die m zur Verfügung stehenden Ritz-Werte gemäß $|\theta_1 - \mu| \leq |\theta_2 - \mu| \leq ... \leq |\theta_m - \mu|$
- Vorauswahl: Wähle τ Ritz-Werte in der Umgebung von μ
- Verwerfe aus diesen die $\tau - s$ Ritz-Werte mit der größten Residuumsnorm

Auch im Fall der sich gegenseitig störenden Ritz-Werte ist die Residuumsnorm ein guter Indikator für Störungen. So kann die Residuumsnorm, falls zwei Ritzwerte den gleichen Eigenwert approximieren, von einem Iterationsschritt zum nächsten signifikant größer werden. In diesem Fall ist eine geeignete Methode, zu einer besseren Approximation zu gelangen, die Residuumsminimierung. Man erhält dann

für den $(k+1)$-ten Iterationsschritt das neue genäherte Eigenpaar $(\sigma^{k+1}, \mathbf{z}^{k+1})$, indem man das Residuum $\mathbf{r}$ für Ritzwerte σ unter folgenden Nebenbedingungen und mit möglichst geringem Rechenaufwand minimiert [1]:

- $\mathbf{z}^{k+1}$ liegt im erweiterten Suchraum und ist normiert
- $\|\mathbf{A}\mathbf{z}^{k+1} - \sigma^{k+1}\mathbf{z}^{k+1}\|_2 \leq \|\mathbf{A}\mathbf{u}^k - \theta^k\mathbf{u}^k\|_2$
- $\mathbf{z}^{k+1}$ approximiert $\mathbf{u}^k$ und σ^{k+1} approximiert θ^k

Implementierung

Das vorliegende Programm gliedert sich im wesentlichen in drei Teile:
Zuerst werden die benötigten Parameter und die Matrix aus Dateien eingelesen. Die im CRS-Format [2] abgespeicherte Matrix liegt aufgrund des geringeren Speicherbedarfs als Binärfile vor.
In einem zweiten Schritt werden die Matrixeinträge zeilenweise so auf die Prozessoren verteilt, daß jeder Prozessor in etwa gleich viele Einträge erhält. Diese Matrixeinträge werden dann auf jedem Prozessor in einen lokalen und einen nichtlokalen Block umsortiert. Für die nichtlokalen Elemente wird daraufhin ein Kommunikationsschema errechnet, indem ermittelt wird, welche Matrixeinträge von welchen anderen Prozessoren benötigt werden, und an welcher Stelle diese liegen. Durch diese Anordnung ist es möglich, die Matrix-Vektor-Multiplikationen im hochdimensionalen Raum in eine lokale und eine nichtlokale Subroutine auszulagern, die parallel auf mehreren Prozessoren laufen können und nur ein minimal nötiges Maß an Kommunikation erfordern.
Die Berechnungen innerhalb der niedrigdimensionalen Teilräume werden nicht parallelisiert, um dadurch zu vermeiden, daß eine weitere größere Anzahl an Synchronisationspunkten erforderlich würde. Dadurch, daß die Dimension dieser Teilräume in der vorliegenden Implementierung auf maximal 81 beschränkt ist, ist eine Parallelisierung auch nicht erfordelich.
Der Hauptteil des Programms besteht aus dem eigentlichen Jacobi-Davidson-Algorithmus. Der Vorteil der vorliegenden Version besteht darin, daß sowohl das gewöhnliche Ritz-Verfahren mit Residuumsminimierung wie auch das harmonische Verfahren nebeneinander angewendet werden, je nachdem, ob der "rank" des Prozessors gerade oder ungerade ist. Nachdem beide Varianten parallel ausgeführt wurden, vergleicht man die mittleren Residuumsnormen, die sich jeweils ergeben haben, miteinander. Man wählt nun die Methode, die zu einem günstigeren Ergebnis führt, als die für diesen Iterationsschritt geeignete. Die Prozessoren, auf denen die Ergebnisse des gewählten Verfahrens vorliegen, senden diese an die anderen, so daß wieder auf allen Prozessoren die gleichen Daten vorliegen.
Nach einem Test auf Konvergenz der ermittelten Ritz-Werte wird die Erweiterung des Suchraums durchgeführt. Nachdem innerhalb der Suchraumerweiterung die Korrekturvektoren bestimmt wurden, führt man zur Orthonormalisierung der neuen Basisvektoren eine implizite Cholesky-Zerlegung [6] durch. Diese hat gegenüber dem Gram-Schmidt-Verfahren, in dem eine relativ große Anzahl von Skalarprodukten berechnet werden muß, den Vorteil, daß sie nur einen Synchronisationspunkt benötigt.

Anwendung auf Testmatrizen

Mit Hilfe des vorliegenden Programms sollten jetzt Eigenwertberechnungen an Matrizen durchgeführt werden, die in der theoretischen Festkörperphysik auftreten.

Anderson-Modell

Das Anderson-Modell [7] gilt als geeignetes Modell zur Beschreibung elektronischer Eigenschaften in ungeordneten Systemen. Man kann dadurch quantenmechanische Effekte beschreiben, die im Zusam-

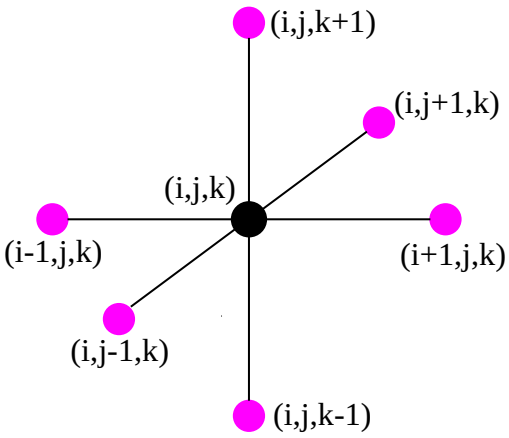
menhang mit der Unordnug stehen, wie beispielsweise die zunehmende räumliche Lokalisierung der elektronischen Wellenfunktionen bei steigender Unordnung. Auf diese Weise kann man den Übergang von einem Metall zu einem Nichtleiter theoretisch modellieren.
Der dieses Modell beschreibende Hamilton-Operator läßt sich als reell symmetrische Matrix $\mathbf{A}$ darstellen, deren Eigenvektoren den quantenmechanischen Wellenfunktionen entsprechen. Das Problem reduziert sich daher mathematisch auf die Lösung der Eigenwertgleichung

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \tag{7}$$

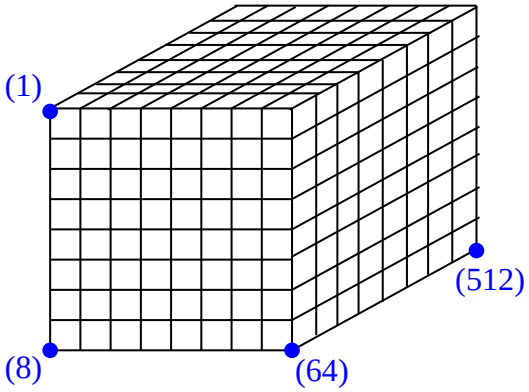
wobei die Vektoren $\mathbf{x}$ für ein kubisches Gitter mit Kantenlänge N von der Dimension N^3 sind. In der Gitterplatz-Darstellung stellt sich dieses Eigenwertproblem dar als:

$$x_{i-1,j,k} + x_{i+1,j,k} + x_{i,j-1,k} + x_{i,j+1,k} + x_{i,j,k-1} + x_{i,j,k+1} + \epsilon_{i,j,k}x_{i,j,k} = \lambda x_{i,j,k} \tag{8}$$

Hierbei bezeichnet i, j, k die kartesischen Koordinaten des Gitterplatzes. Die Nichtdiagonalterme entsprechen den Hüpfwahrscheinlichkeiten und seien zur Vereinfachung alle gleich groß gewählt, was einem isotropen Hüpfmodell entspricht. Die Diagonalelemente beinhalten zufallsverteilte Potentiale $\epsilon_{i,j,k}$ innerhalb des Energieintervalls $[-\frac{\omega}{2}, +\frac{\omega}{2}]$, wobei $\hbar = 1$ gesetzt ist. Durch diese Zufallsverteilung der Potentiale wird der Unordnung im betrachteten System Rechnung getragen. Eine Verallgemeinerung des betrachteten Modells wäre eine anisotrope Hüpfwahrscheinlichkeit, die eventuell auch noch zufallsverteilt sein könnte. Da sich dadurch aber an der Struktur der Matrix nichts ändert, bleibt die Betrachtung weiterhin auf den Spezialfall des isotropen Hüpfmodells beschränkt. In der Regel betrachtet man dieses Modell mit periodischen Randbedingungen, wie es auch hier im Folgenden angenommen wird. Es sind jedoch auch andere Randbedingungen möglich.



Um von der Gitterplatzdarstellung zu einer Darstellung entsprechend Gleichung (7) zu gelangen, faßt man die gesamten Gitterplätze zu einem Vektor zusammen, d.h. man numeriert die einzelnen Gitterplätze nun also nicht mehr mit (i, j, k), von $(1, 1, 1)$ bis (N, N, N) durch, sondern mit $\hat{\imath}$ von 1 bis N^3. Die Spalte $\hat{\imath}$ der Matrix $\mathbf{A}$ entspricht dem "Stern" von (8) zentriert auf die Position, welcher der Index $\hat{\imath}$ zugeordnet ist. Die gesamte Matrix $\mathbf{A}$ erhält man, indem man mit dem "Stern" alle Positionen des Gitters, also die gesamten Indizes $\hat{\imath}$ durchläuft. Damit ergibt sich für die Matrix $\mathbf{A}$ folgende Struktur (Abbildung 1):



Physikalische Eigenschaften

Es zeigt sich, daß je nach Wahl des einzig freien Parameters ω die Eigenwerte und Eigenvektoren der Matrix $\mathbf{A}$ völlig verschiedene Eigenschaften aufweisen [3].
Für kleine Werte von ω, etwa $\omega \ll 16.5$, besitzt $\mathbf{A}$ "ausgedehnte Eigenvektoren", d.h. alle Komponenten des Vektors haben einen in etwa gleichgroßen, nichtverschwindenden Betrag. In diesem "ausgedehnten

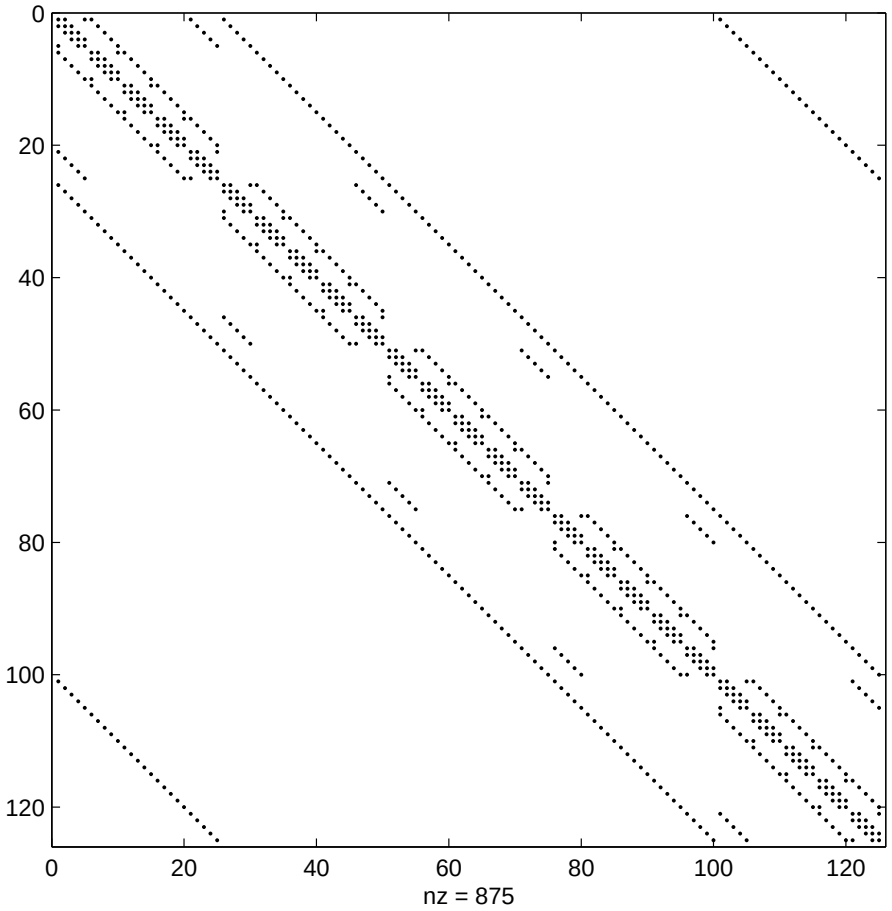


Abbildung 1: Struktur einer Anderson-Matrix (dim$= 125 \times 125$)

Bereich" fluktuieren die $x_{i,j,k}$ über die verschiedenen Gitterplätze und $|x|$ ist durch eine nichtverschwindende Konstante beschränkt.
Ist andererseits $\omega \gg 16.5$, so dominiert die Diagonale die Matrix und ihre Eigenschaften, und die Eigenvektoren sind sehr stark lokalisiert. Für den n-ten Eigenzustand gilt die Abschätzung:

$$|x_n| \approx \exp\left(-\frac{|\vec{r} - \vec{r_n}|}{l_n(\omega)}\right) \text{ mit } \vec{r} = (i,j,k)^T \text{ und Lokalisationslänge } l_n(\omega)$$

Da, im Gegensatz zu lokalisierten Zuständen, nur ausgedehnte Zustände zum Elektronentransport beitragen können, kann man mit Hilfe des Anderson-Modells die Unterschiede und den Übergang von einem Metall zu einem Nichtleiter beschreiben.
Ebenso wie auf die Eigenvektoren wirkt sich die Wahl von ω auch auf die Eigenwerte aus. So besteht das Spektrum für große ω aus gleichverteilten Eigenwerten mit kleinen Schwankungen aufgrund der Zufallsverteilung der $\epsilon_{i,j,k}$, wohingegen für kleine ω eine Clusterung der Eigenwerte auftritt.

Schwierigkeiten und Konzepte zur Beseitigung

Aufgrund der speziellen Struktur der Anderson-Matrizen ergeben sich für den vorliegenden Algorithmus einige Schwierigkeiten. Im Folgenden muß also sowohl der Indefinitheit, wie auch dem engen Zusammenfallen der Eigenwerte besonders Rechnung getragen werden. Dadurch, daß das vorliegende Programm bisher noch nicht auf Matrizen angewendet wurde, bei denen die Eigenwerte innerhalb der Cluster so dicht benachbart sind wie in der Anderson-Matrix, traten die Schwierigkeiten, mit denen man es jetzt zu tun hat, noch nicht auf.

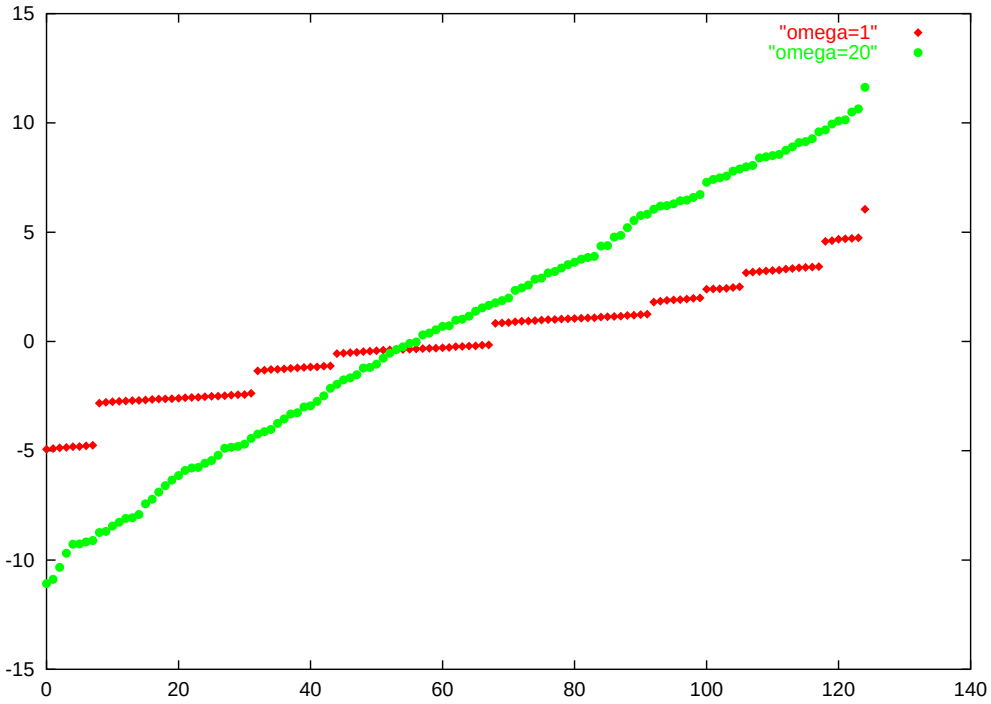


Abbildung 2: Spektrum zweier Anderson-Matrizen für verschiedenes ω (dim$= 125 \times 125$)

Lucky Breakdown

Das harmonische Jacobi-Davidson-Verfahren bricht zusammen, wenn man als verwendeten Shift genau einen Eigenwert trifft. Denn falls der Shift μ gleich dem Eigenwert $\tilde{\lambda}$ ist, so hat $\mathbf{A} - \mu\mathbf{I}$ einen Nulleigenwert und ist demzufolge nicht invertierbar. Da für die Anderson-Matrix die Eigenwerte nach dem Gersgorin circle theorem [8] innerhalb des Intervalls $[-\frac{\omega}{2} - 6, +\frac{\omega}{2} + 6]$ liegen, ist bei genügend großer Matrixdimension und hinreichend kleinem ω die Wahrscheinlichkeit groß, daß in irgendeinem Iterationsschritt einmal ein Eigenwert (innerhalb der Rechnergenauigkeit) direkt getroffen wird. Innerhalb der Testläufe trat dieses auch vermehrt auf, wobei das verwendete Programm andererseits in manchen Fällen auch problemlos funktionierte. Um dieses unvorhersagbare Verhalten in den Griff zu bekommen, ist es nötig, neue Strategien zu entwickeln.

- Wenn man durch Zufall mit dem Shift genau einen Eigenwert trifft, und dieses bemerkt, so hat man schon einen konvergierten Eigenwert gefunden (Lucky Breakdown). In diesem Fall ist es dann nötig, den Iterationsschritt nicht einfach weiterzuführen. Da das direkte Treffen des Eigenwerts das harmonische Verfahren zum Abbrechen bringt, ist ein gezieltes Abfangen des Lucky Breakdown erforderlich. Wenn die Spalten von $\mathbf{V}$ eine Basis von $\mathcal{V}^m$ bilden, so hat $\mathbf{V}$ noch vollen Rang, wohingegen $\mathbf{W} = (\mathbf{A} - \mu\mathbf{I})\mathbf{V}$ aufgrund des Nulleigenwertes von $\hat{\mathbf{A}} := \mathbf{A} - \mu\mathbf{I}$ rangdefekt ist. Also hat $\mathbf{W}$ einen nichttrivialen Kern, und man muß den Vektor aus $\mathbf{V}$, der auf $\mathbf{0}$ abgebildet wird, aus dem Verfahren herausnehmen. Dadurch wird die Dimension des Suchraums um eins erniedrigt. Dafür hat aber $\mathbf{W}$ wieder vollen Rang und man hat schon ein konvergiertes Eigenpaar gefunden. Mit den auf diese Weise reduzierten Matrizen $\mathbf{V}$ und $\mathbf{W}$ kann man nun wie gewohnt weiterverfahren.
- Eine weitere Möglichkeit, den Nulleigenwert von $\hat{\mathbf{A}}$ abzufangen, besteht darin, anstatt des "normalen" Eigenwertproblems $\hat{\mathbf{A}}^{-1}\mathbf{u} = \tilde{\theta}^{-1}\mathbf{u}$, das verallgemeinerte Eigenwertproblem

$$\mathbf{V}^T\mathbf{W}\mathbf{u} = \vartheta\mathbf{W}^T\mathbf{W}\mathbf{u} \tag{9}$$

zu betrachten. Dieses ist dadurch, daß $\mathbf{W}^\mathbf{T}\mathbf{W}$ nur positiv-*semi*definit und somit nicht invertierbar ist, ein "wirkliches" verallgmeinertes Eigenwertproblem und kann nicht durch Umformungen auf ein "normales" zurückgeführt werden.

- Außerdem hat man noch die Option des Einwirkens auf $\mathbf{V}$ bzw. $\mathbf{W}$ innerhalb der Routine zur Cholesky-Zerlegung. In der bisherigen Implementierung verwirft man, sobald ein Pivot-Element verschwindet, zur Reduktion der Dimension denjenigen Vektor, der gerade zufällig der in der Suchraumerweiterung zuletzt hinzugefügte ist. Hier könnte eine gezielte Auswahl desjenigen Vektors, der den maximalen Beitrag in Richtung des auf $\mathbf{0}$ abgebildeten Vektors aufweist, zum Erfolg führen.

Näherungsinverse

Bei der weiteren Verbesserung der Konvergenzeigenschaften, was sowohl Stabilität als auch Konvergenzgeschwindigkeit betrifft, wird es von besonderer Bedeutung sein, innerhalb des Preconditioners eine gute Näherungsinverse für die Anderson-Matrix zu finden [9]. Aufgrund der zufallsverteilten Diagonale erweist sich dieses jedoch als äußerst heikles Problem. Eine Näherung, wie die von Davidson [10] vorgeschlagene, in der einfach die Diagonale der Matrix verwendet und invertiert wird, ist hier denkbar ungeeignet. Da die Diagonale in diesem Fall, insbesondere für kleine ω keinerlei geeignete Information über die Matrix liefert, muß man hier auf andere Möglichkeiten ausweichen. Eine sinnvolle Näherungsinverse müßte der Eigenschaft der besonderen Struktur der Matrix (vgl. Abbildung 1) Rechnung tragen, ohne daß sie jedoch von der Zufallsverteilung der Elemente auf der Diagonalen zu stark beeinträchtigt wird.

Literatur

1. A. GÖKE, *Parallele Teilraumverfahren zur Bestimmung von Eigenwerten im Inneren des Spektrums*, Berichte des Forschungszentrums Jülich Jül-3794, Forschungszentrum Jülich GmbH, Jülich, Germany, 2000
2. Y. SAAD, *Numerical Methods for Large Eigenvalue Problems*, Manchester University Press, 1992
3. U. ELSNER, V. MEHRMANN, F. MILDE, R. A. RÖMER, M. SCHREIBER, *The Anderson Model of localisation: a challenge for modern eigenvalue methods*, `http://arXiv.org/abs/physics/9802009`
4. G. L. G. SLEIJPEN, H. A. VAN DER VORST, *A Jacobi-Davidson Iterative Method for Linear Eigenvalue Problems*, SIAM Journal on Matrix Analysis and Applications, Volume 17, Number 2, 1996, pages 401-425
5. D. S. SCOTT, *The Advantages of Inverted Operators in Rayleigh-Ritz Approximations*, SIAM Journal on Scientific and Statistical Computing, 3(1), 1982, pp 68-75
6. A. BASERMANN, B. STEFFEN, *New Preconditioned Solvers for Large Sparse Eigenvalue Problems on Massively Parallel Computers*, Interner Bericht FZJ-ZAM-IB-9701, Forschungszentrum Jülich GmbH, Jülich, Germany, 1997
7. P. W. ANDERSON, *Absence of diffusion in certain random lattices*, Phys. Rev., **109**, 1958, pp 1492-1505
8. G. H. GOLUB, C. F. V. LOAN, *Matrix Computations*, Johns Hopkins University Press, Baltimore and London, 3rd edition, 1996
9. B. STEFFEN, *Subspace Methods for Large Sparse Interior Eigenvalue Problems*, International Journal of Differential Equations and Applications, Volume 3,**3**, 2001, pp. 339 - 351
10. E. R. DAVIDSON, *The Iterative Calculation of a Few of the Lowest Eigenvalues and Corresponding Eigenvectors of Large Real-Symmetric Matrices*, Journal of Computational Physics, **17**, 1975, pp 87-94
11. J. CULLUM, R. A. WILLOUGHBY, *Lanczos Algorithms for Large Symmetric Eigenvalue Computations*, Birkhäuser, Boston, 1985

Einsatz von Virtual Reality Techniken zur Visualisierung geophysikalischer Daten

Sven Trentmann

Westfälische Wilhelms-Universität Münster

s.trentmann@uni-muenster.de

Zusammenfassung: Es wurde ein Datensatz, der als Ergebnis einer Finite-Element-Simulation der zeitlichen Entstehung eines Sedimentbeckens erstellt wurde, unter Einsatz von Virtual Reality Techniken visualisiert, wobei die Holobench inklusive dreidimensionaler Eingabegeräte zum Einsatz kam.

Datengrundlage

Als Datengrundlage dient das Ergebnis einer Simulation, die von der Firma Integrated Exploration Systems (IES) [1] durchgeführt wurde. Der verwendete Simulator ist Teil eines kommerziellen Softwarepaketes, das für die Erdöl- und Erdgasexploration entwickelt wurde. Mit Hilfe einer Finite-Element-Methode wird die Entstehungsgeschichte eines Sedimentbeckens simuliert, wobei Simulationszeiträume von größenordnungsmäßig 50 Millionen Jahren benutzt werden. Beginnend mit einer Basisschicht, dem sogenannten *Basement*, werden im Laufe der Zeit immer neue Schichten (Fazies) abgelagert, die bestimmte, vorgegebene Eigenschaften wie Gesteinsart, Dichte und Anteil an organischem Material haben. Zusätzlich kann die Form der Schicht (Dickenverteilung) vorgegeben werden. Als wichtige Komponenten gehen außerdem die Temperatur-Randwerte (z. B. Temperaturfluß von unten) in die Simulation ein, woraus dann z. B. Temperatur-, Porositäts-, und Druckverteilungen berechnet werden können. Aufbauend auf diesen Daten lassen sich dann Antworten auf drei grundlegende Fragen finden:

1. Wo kann sich Erdöl/Erdgas bilden (Entstehung)?
2. Welche Wege nimmt das entstandene Erdöl/Erdgas (Migration)?
3. Wo sammelt sich Erdöl/Erdgas an (Bildung von Reservoiren)?

Entscheidende Größen zur Bildung von Erdöl/Erdgas sind die Temperatur, die Druckverteilung und der Anteil an organischem Material, für die Migration sind der Druck und die Porosität entscheidend, ebenso für die Ansammlung (wann reicht der Druck nicht mehr aus, um das Öl durch das Gestein zu befördern?). Im englischen ist diese Art der Simulation bekannt unter dem Begriff *basin modelling.*

Der Ausgabedatensatz enthält Daten, die zu diskreten, vorgegebenen Zeitpunkten, den sogenannten *events*, herausgeschrieben wurden und das Modell mit allen Parametern zu diesen Zeitpunkten beschreiben. Die eigentliche Schrittweite der Simulationsschritte, die der Simulator intern benutzt, kann von den Abständen der Events abweichen. Das erste Event stellt das Basement dar, das letzte Event beschreibt typischerweise die (simulierten) Verhältnisse der Gegenwart (*present day*). Im benutzten Datensatz waren 37 solcher Events vorhanden.

Verwendete Programmpakete

Als hauptsächliche Komponenten wurden die beiden Visualisierungs-Bibliotheken svt - Scientific Visualization Toolkit und vtk - The Visualization Toolkit verwendet. Die grundlegenden Prinzipien und die Schnittstelle zwischen beiden Systemen sollen im folgenden beschrieben werden.

svt - Scientific Visualization Toolkit

Bei svt handelt es sich um eine Entwicklung des ZAM, die von Stefan Birmanns, Herwig Zilken und Frank Delonge programmiert wurde, um 3-dimensionale Szenarien auf verschiedenen Ausgabemedien darzustellen. Hierbei wurde Wert auf Flexibilität gelegt, die Software paßt sich gegebenen Hardwarevoraussetzungen an: es ist sowohl möglich, die Szene auf dem Holobench-System des ZAM darzustellen als auch in einem Fenster auf einem normalen Bildschirm. Auch die Benutzung von VR-Systemen mit anderen Bildschirmanordnungen, z. B. Caves, ist vorgesehen.

Mit Hilfe der Holobench kann eine virtuelle, 3-dimensionale Szene erzeugt werden, wobei die räumliche Darstellung dadurch erzielt wird, daß getrennte Bilder für das linke und rechte Auge abwechselnd angezeigt werden. Mit Hilfe einer speziellen Brille, der *Shutter-Brille*, die synchron dazu abwechselnd linkes und rechtes Auge des Betrachters verdunkelt, wird schließlich der stereoskopische (räumliche) Effekt erzielt. Die Synchronisation der Brille mit dem Bildschirm erfolgt hierbei hardware-gesteuert, das svt ist für die unterschiedlichen Darstellungen für das linke und rechte Auge verantworlich.

Die Darstellung geschieht in Abhängigkeit von der räumlichen Position des Kopfes vor der Holobench, um dem Betrachter den Eindruck zu vermitteln, das Objekt von allen Seiten betrachten zu können. Das dazu benötigte Kopf-Tracking wird genauso von svt übernommen wie die Steuerung von Eingabegeräten wie einem dreidimensionalen Eingabestift oder Force-Feedback-Geräten (nähere Informationen zur Holobench, siehe [2]).

Svt arbeit scenegraph-basiert, was bedeutet, daß die darzustellende Szene aus Objekten aufgebaut ist, die hierarchisch in Form einer Baumstruktur angeordnet werden. Objekte können in diesem Zusammenhang Kameras, Lichtquellen, Transformationen (alle Objekte unterhalb dieses Knotens werden transformiert), geometrische Objekte (z. B. Kugeln, Zylinder), oder **vtk-Objekte**, sogenannte Aktoren, sein. Der Begriff vtk-Objekt wird im nächsten Abschnitt näher erklärt, hier soll nur schon einmal erwähnt sein, daß dies die Schnittstelle der beiden Visualisierungspakete darstellt, durch die eine Nutzung der vtk-Funktionalitäten auch für svt ermöglicht wird.

vtk - The Visualization Toolkit

Bei vtk handelt es sich um eine Visualisierungsbibliothek auf Basis von OpenGL, die, genau wie svt, als C++-Klassenbibliothek implementiert ist. Es handelt sich um ein Open Source Projekt, die Nutzung ist also kostenlos, sämtlicher Code ist im Quellcode verfügbar (siehe auch [5]). vtk zeichnet sich durch eine Vielzahl von Datenstrukturen und durch effiziente Algorithmen auf diesen Datenstrukturen aus. So ist z. B. eine Gitterstruktur als Objekt implementiert, das zur Darstellung der Daten verwendet werden konnte. Genutzt werden neben einfacheren Algorithmen z. B. zur Darstellung des Modellumrisses oder eines Drahtmodells auch komplexere Operationen wie die Berechnung von Schnitten durch das Modell.

Zur Visualisierung nutzt vtk ein Pipeline-Verfahren, was bedeutet, daß die Schritte vom reinen Datensatz bis zur Darstellung auf dem Bildschirm im Schritt-für-Schritt-Verfahren mit Hilfe mehrerer Filter umgesetzt werden.

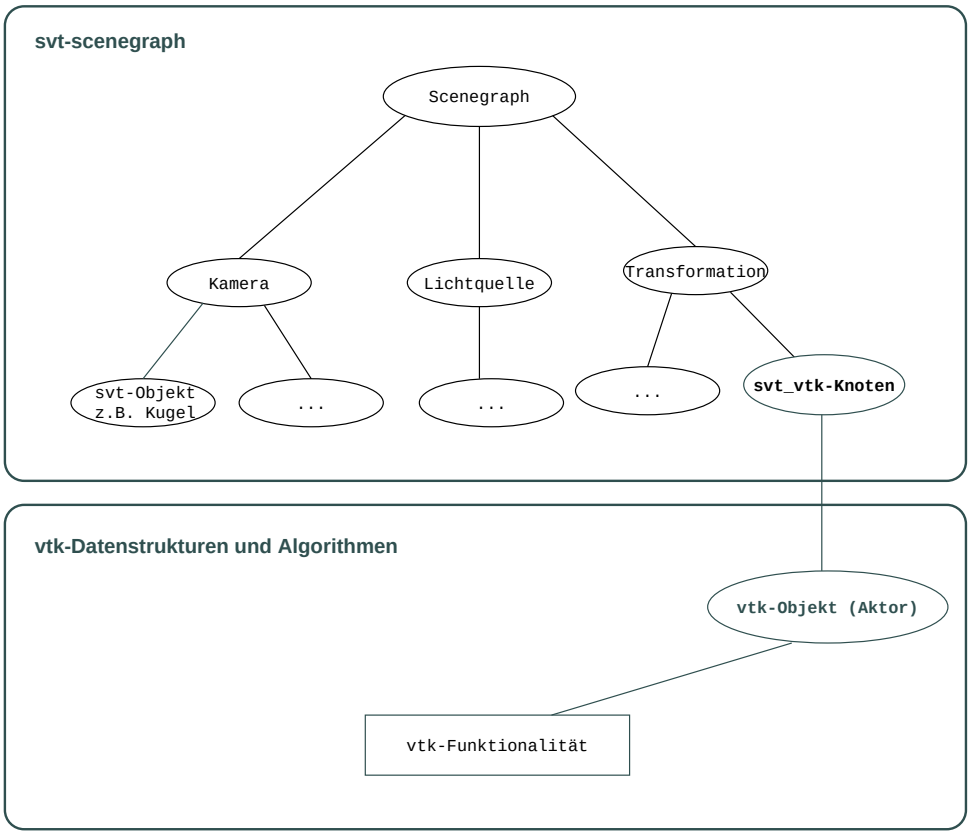


Abbildung 1: Hierarchischer Aufbau einer 3-dimensionalen Szene mit svt und vtk

PSfrag replacements

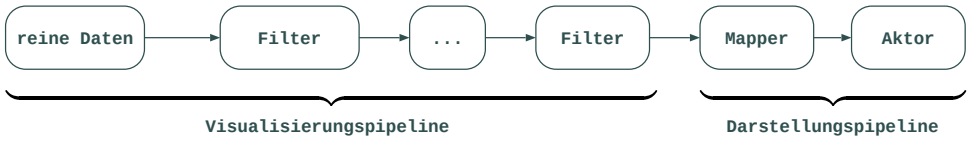


Abbildung 2: Pipeline-Verfahren bei vtk

Getrennt wird zwischen Aufbereitung der Daten (Visualisierung) und Darstellung der aufbereiteten Daten. Zur Visualisierung zählen z. B. Berechnung von Schnitten, Interpolation und die Berechnung von Umrissen, wogegen die Auswahl einer Farbtabelle oder die Positionierung des Objektes zur Darstellung zählen. Das Bindeglied, der *Mapper*, konvertiert eine abstrakte Datendarstellung (z. B. Gitter) in eine darstellungsgerechte Datenrepräsentation (z. B. Polygone und Dreiecke). Die letzte Stufe der Pipeline stellt der Aktor dar, ein Objekt, das letztendlich „gemalt", also auf dem Bildschirm dargestellt werden kann.

Wird ein Aktor aufgefordert, sich zu zeichnen, wird die Visualisierungspipeline ab dem Punkt neu ausgeführt, an dem sich Daten geändert haben (Ablauf von rechts nach links). Wenn sich also an den Daten nichts geändert hat, wird die Visualisierungspipeline nicht komplett neu durchlaufen, sondern es wird auf alte Daten zurückgegriffen. Das kann z. B. der Fall sein, wenn ein Objekt nur verschoben wurde. Hier würde nur der Darstellungsteil der Pipeline neu ausgeführt werden. Dies führt zu einer schnellen Darstellung der Daten.

Folgendes Codefragment kann die Benutzung der Pipeline verdeutlichen:

```
aFilter->SetInput(anotherFilter->GetOutput());
mapper->SetInput(aFilter->GetOutput());
actor->SetMapper(mapper);
```

Zusätzlich dazu können mehrere Filter oder Aktoren auf eine Datengrundlage zugreifen, so daß eine effiziente Speicherung der Daten möglich wird: z. B. kann ein Gitter sowohl als Drahtgitter als auch als Modell mit ausgefüllter Oberfläche auf dem Bildschirm angezeigt werden. Hierfür müssen nur zwei verschiedene Aktoren bereitgestellt werden, die ansonsten auf die gleiche Pipeline aufsetzen:

```
aFilter->SetInput(anotherFilter->GetOutput());
mapper->SetInput(aFilter->GetOutput());
aActor->SetMapper(mapper);
bActor->SetMapper(mapper);
bActor->GetProperty()->SetRepresentationToWireframe();
```

Von dieser Möglichkeit wird im entwickelten Programm des öfteren Gebrauch gemacht.

Ein Aktor kann nun, anstatt mit vtk-Mitteln auf dem Bildschirm dargestellt zu werden, in den svt-Scenegraphen eingefügt werden und als „normales" Objekt der Szene behandelt werden, es kann also mit svt-Mitteln gedreht, verschoben, skaliert oder beleuchtet werden. So kann auf die vorhandenen Datenstrukturen und Algorithmen des vtk und auf die Darstellungsmöglichkeiten (z. B. Holobench-System) des svt zurückgegriffen werden.

Implementation

Dem darzustellenden Modell liegt eine Finite-Element-Simulation zugrunde, es liegt also nahe, zur Visualisierung der Ergebnisse eine Gitterstruktur zu benutzen. vtk bietet zwei Gitterstrukturen an: ein strukturiertes und ein unstrukturiertes Gitter. Strukturiert bedeutet hierbei, daß die Anzahl der Gitterpunkte in den drei Raumrichtungen einheitlich sein muß, es darf kein Element fehlen, wogegen beim unstrukturierten Gitter alle möglichen Elementgeometrien denkbar sind: das Gitter kann sich zusammensetzen aus Tetraedern, Hexaedern, etc. Dadurch wird das unstrukturierte Gitter zwar sehr flexibel, als Datenstruktur aber aufwendig.

Bei Verwendung eines strukturierten Gitters trat jedoch das Problem auf, daß Schichten, bedingt z. B. durch Erosion, auskeilen können, so daß die Struktur des Gitters nicht ohne weiteres auf ein „structured grid" im Sinne von vtk abgebildet werden kann. Dieses Problem wurde gelöst durch das Einfügen von Pseudo-Elementen, denen eine Höhe von 0 zugewiesen wurde, wodurch sie im Modell nicht sichtbar sind.

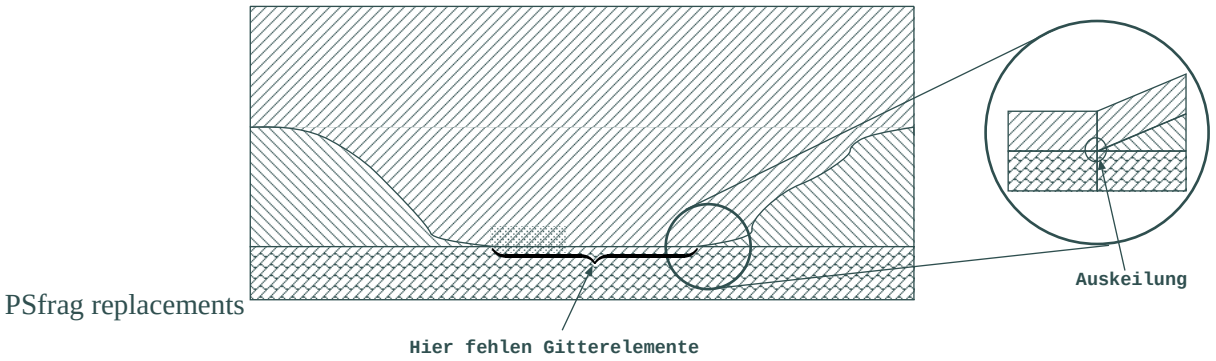


Abbildung 3: Gitterelemente können nach außen „herausgedrückt" werden

Nachdem die Geometrie des Gitter aufgebaut ist, werden simulierte Attributwerte eingelesen (*overlays*), wobei exemplarisch nur drei Datensätze genutzt werden: Die IDs der geologischen Fazies, die Temperaturverteilung und die Porosität. Die Fazies ist elementweise gegeben und wird farblich nicht interpoliert

(*flat shading*). Die geologische Fazies stellt in diesem Zusammenhang insofern eine Besonderheit dar, daß jedes Element während der gesamten Simulation einer festen Gesteinsformation zugeordnet ist, die sich im Zeitablauf nicht ändern kann. Deswegen genügt es, die Fazies-IDs einmal einzulesen, wenn das Gitter aufgebaut wird. Im Gegensatz dazu sind die meisten Overlays knotenweise gegeben und ändern sich in jedem Zeitschritt, wie z. B. die Temperaturverteilung. Da der Wert dieser Knotenattribute im Inneren eines Elements nicht eindeutig gegeben ist, wird hier der Farbverlauf interpoliert dargestellt (*gouraud shading*).

Zur Verdeutlichung der Modellgeometrie wurde ein Aktor erzeugt, der das Modell als Drahtgitter darstellt. Die Ausmaße werden mit Hilfe einer Umgebungsbox (*bounding box*) verdeutlicht.

Schnittebenen - Werkzeug

Zur Auswertung der Daten wurde ein Werkzeug entwickelt, mit dem eine Schnittebene frei im Raum positioniert werden kann. Die Schnittebene wird auf der einen Seite dafür benutzt, um einen Teil des Modellgitters auszublenden (*clipping*). Durch diesen Schnitt entsteht eine neue Oberfläche, die neu berechnet werden muß (*cutting*). Hierzu wird die Schnittfläche aus dem Modell und der positionierbaren Schnittebene berechnet. Es müssen sowohl Geometrie- als auch Overlay-Informationen des Modells interpoliert werden, weshalb der Rechenaufwand hoch ist. Aus diesem Grunde wird die Schnittfläche erst berechnet, sobald die Schnittebene mit der Maus o. ä. im Modell positioniert wurde. Somit wird ein interaktives Schneiden ermöglicht, das allerdings durch die Wartezeit bei der Berechnung der Ebene eingeschränkt wird. Um die Schnittmöglichkeiten zu erhöhen, kann dem Modell mehr als eine Schnittebene hinzugefügt werden.

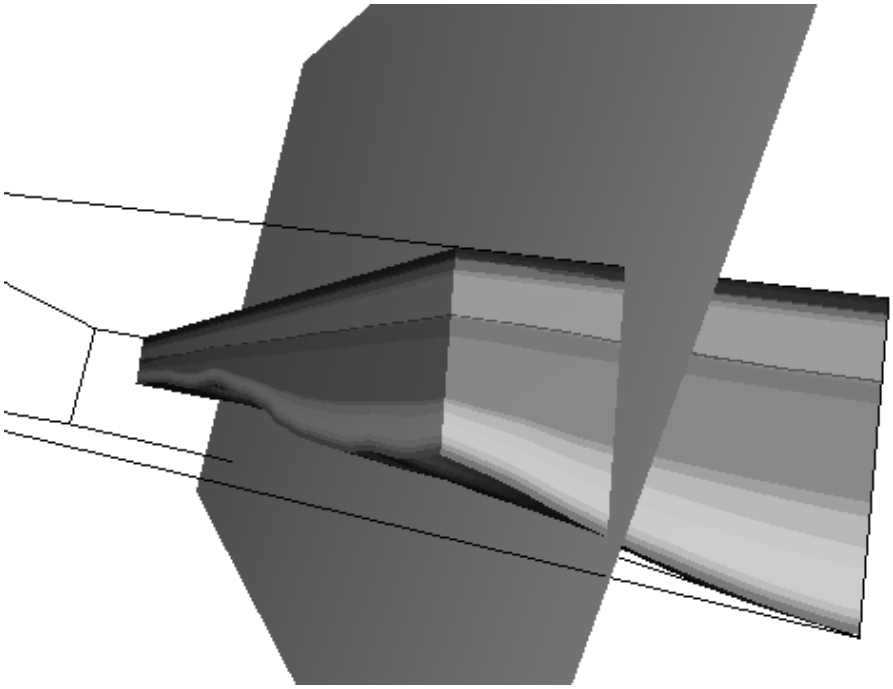


Abbildung 4: Mit einer frei im Raum positionierbaren Ebene können beliebige Schnitte durch das Modell erzeugt werden

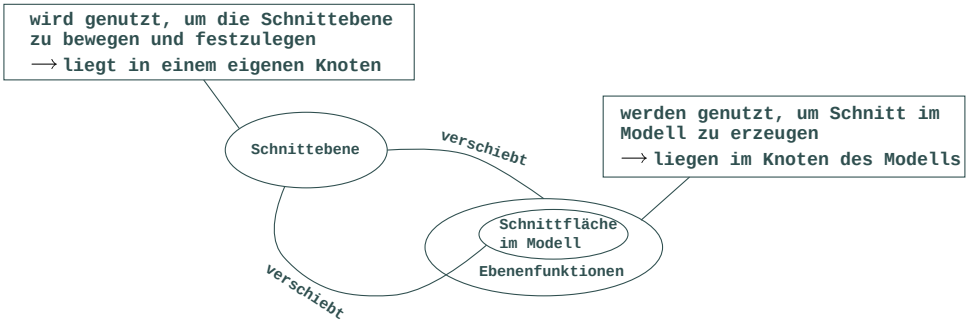
Im folgenden wird die Arbeitsweise des Schnittebenenwerkzeugs näher erklärt. Hierzu muß man zunächst wissen, daß es zwei verschiedene Ebenentypen in vtk gibt: eine Ebenenfunktion, die eine unendlich ausgedehnte Ebene beschreibt, und eine begrenzte Ebene, die als Aktor angezeigt werden kann. Das Schnittwerkzeug geht nun so vor, daß mit der Maus ein Aktor (eine begrenzte Ebene) bewegt werden

kann (das Werkzeug ist ein svt-Knoten, kann also wie jedes andere svt-Objekt verschoben werden). Läßt der Nutzer die Maus los, werden Ebenen (sowohl Aktoren als auch Ebenenfunktionen) nachgezogen, die in anderen Knoten liegen können.

Da die nachgezogenen Ebenen in anderen scenegraph-Knoten liegen, sind hier Koordinatentransformationen nötig.

Vereinfacht kann man die Vorgehensweise folgendermaßen beschreiben: Ein svt-Knoten, der eine Ebene enthält, wird verschoben (das ist das Schnitt-Werkzeug). Beim Loslassen der Maus werden dann in einem anderen Knoten Ebenen und Ebenenfunktionen (Zielebenen) nachgezogen und auf die verschobene Ebene ausgerichtet. Auch wenn das Modell bewegt wird, wird die Schnittebene nachgezogen.

Mit Hilfe von anschließendem Clipping und Cutting konnte so die frei positionierbare Schnittebene realisiert werden. Die Möglichkeit, mehrere Schnittebenen durch den Datensatz zu legen, wird dadurch gegeben, daß aus mehreren Ebenenfunktionen die Vereinigungsmenge gebildet werden kann und diese dann als „Schnittmuster" verwendet werden konnte.



PSfrag replacements

Abbildung 5: Prinzip des Schnittebenen-Werkzeugs

Weitere Funktionen

Über die Tastatursteuerung können einige Funktionen aufgerufen werden:

zoom/unzoom: Es gibt die Möglichkeit, das Modell zu vergrößern und zu verkleinern. Die Benutzung der Funktionen hat den Vorteil gegenüber der Navigation mit der Maus, daß das Modell nicht verschoben wird. Verschiebt man nämlich das Modell, so daß das virtuelle Bild nahe vor dem Auge des Betrachters steht, entsteht folgender Effekt: Der Betrachter meint, das Bild in kleiner Entfernung vor sich stehen zu sehen, das Auge hat jedoch nach wie vor auf den Bildschirm fokussiert (denn nur dort wird das Bild in Wirklichkeit angezeigt). Hält er nun z. B. seine Hand „neben" das virtuelle Objekt, so ist die Hand nicht fokussiert, weil sie sich ja weit vor dem Bildschirm befindet, so daß die Betrachtung eine Belastung für die Augen wird. Es ist deswegen günstiger, das virtuelle Objekt in der Bildschirmebene darzustellen.

Wechseln des Events: Um die zeitliche Entwicklung der Simulation zu verstehen, kann man zwischen den aufgezeichneten Events hin- und herschalten. Beim Wechsel des Events werden die Daten neu eingelesen.

Wechsel des angezeigten Overlays: Das angezeigte Overlay kann geändert werden. Drei Overlays sind vorhanden: (0) geologische Fazies, (1) Temperatur, (2) Porosität. Da alle Daten im Speicher liegen, ist hier kein Nachladen von der Festplatte notwendig.

Schnittebenenmodus: Zwischen den Schnittstellenmodi kann durch Tastendruck hin- und hergeschaltet werden: Es sind zwei Schnittebenen vorhanden, es werden abwechselnd das Modell, Ebene 1 und Ebene 2 als aktives Objekt, das bewegt werden kann, angewählt.

Ein- und Ausblenden von Schichten: Einzelne geologische Fazies können ein- und ausgeblendet werden. Dies geschieht, indem die Nummern der Schichten angegeben werden, die im Modell angezeigt werden sollen. Es wird daraufhin ein Untergitter erzeugt, das nur die gewünschten Schichten enthält. Die Handhabung des so entstandenen Modells ist deutlich langsamer, was daran liegt, daß das Gitter durch die Operation unstrukturiert geworden ist. Durch Druck auf *u* können wieder alle Schichten angezeigt werden. Die Darstellung einzelner Schichten ist sinnvoll, wenn es darum geht, eine Übersicht über die Ausmaße und Form des Datensatzes zu bekommen. Z. B. kann man nur die untere und obere Schicht anzeigen, dazwischen z. B. Vektordaten (die leider nicht implementiert sind).

Programmaufbau

Es wurde objektorientiert in C++ entwickelt. Entstanden sind folgende Klassen:

svt_ies_grid: Kapselung der Daten eines Events. Die Daten dienen dann als Grundlage für die Darstellung, die in *ies_scene* vorgenommen wird. Hier wird das Pipelineprinzip des vtk benutzt: wird z. B. im *svt_ies_grid* (Daten) zu einem anderen Event gewechselt, so wird die *ies_scene* (Darstellung) automatisch aktualisiert.

svt_ies_move_plane: Ebene, die mehrere andere Ebenen „mitziehen" kann.

svt_ies_node: Ein Gitterknoten des Datensatzes. Gespeichert werden die Koordinaten und eine Knotennummer.

svt_ies_element: Ein Gitterelement, gespeichert werden die Nummern der Punkte, die das Element beschreiben, die Elementnummer und die geologische Fazies.

svt_ies_nodeDataReader: Einlesen von Knotendaten (Temperatur, Porosität).

svt_ies_vectorData: Kapselung eines Vektordatensatzes.

svt_ies_vectorDataReader: Einlesen von Vektordaten, leider wurde die Anzeige von Vektordaten nicht mehr realisiert.

svt_ies_reader: Einlesen der Gitterstruktur und der geologischen Fazies-Informationen.

svt_ies_utils: Nützliches...

ies_scene: Aufbau der Szene, als „Datenlieferant" fungiert *svt_ies_grid.*

Alle Klassen sind abgelegt im Projektverzeichnis *ies*, zum Kompilieren genügt ein Aufruf von *make*. Getestet wurde das Programm unter Solaris 5.8 mit dem g++-Compiler (Version 2.95.1) und unter Irix mit dem MIPSpro C++-Compiler (Version 7.3.1.2m).

Der Programmaufruf lautet:

```
ies_holo_main -c holobench_config Projektname [eventnummer]
```

Unter Nutzung der Möglichkeiten des vtk ist man schnell in der Lage, auch komplexe Daten zu visualisieren. Den Umfang an Möglichkeiten bezahlt man allerdings damit, daß die vtk-Bibliotheken recht groß sind ($\approx$ 130 MB), wodurch die Portabilität des Programms zumindest erschwert wird. Zum anderen ist der Bedarf an Hauptspeicher der Anwendung immens: Beim benutzten Modelldatensatz, der einer realistischen Größe entspricht, beträgt der Bedarf an Hauptspeicher mehr als 350 MB, so daß sich die Anwendung kaum für einen Standard-PC eignet.

Im folgenden nun einige Bilder, die die Ergebnisse des Projektes wiederspiegeln sollen:

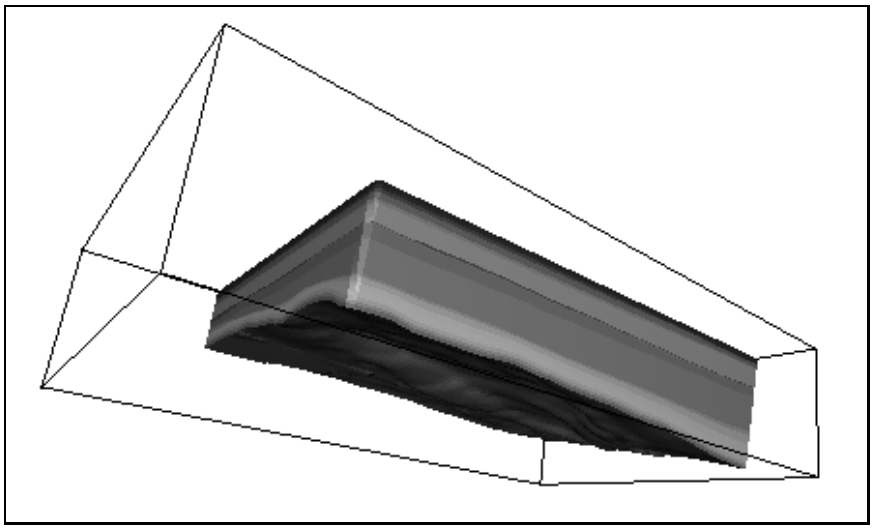


Abbildung 6: Mit Hilfe zweier Schnittebenen kann man das Modell eingrenzen

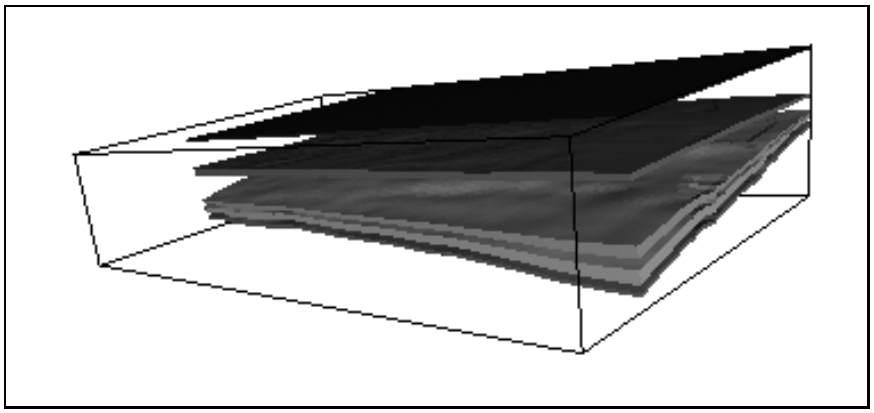


Abbildung 7: Ebenen können ein- und ausgeblendet werden

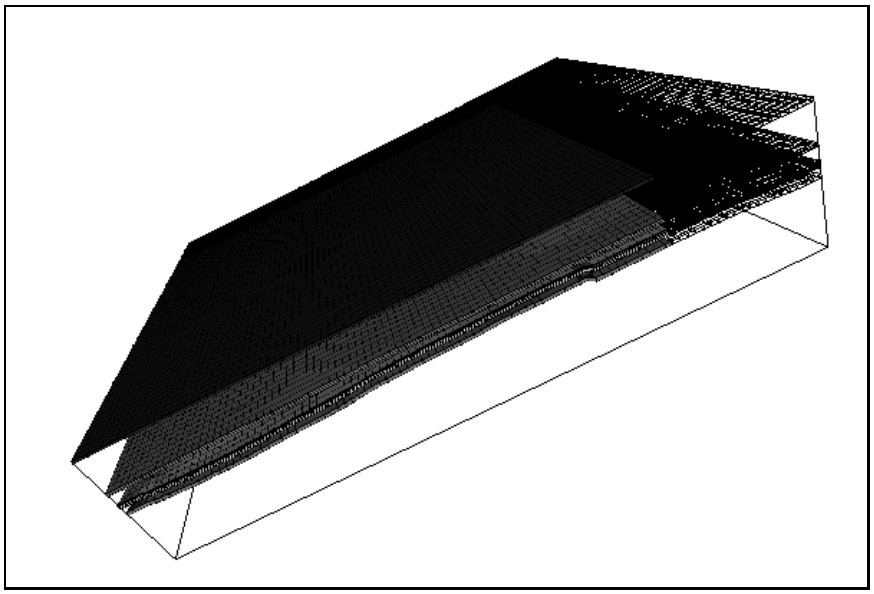


Abbildung 8: Ein Drahtgitter macht die Geometrie des Gitters deutlich

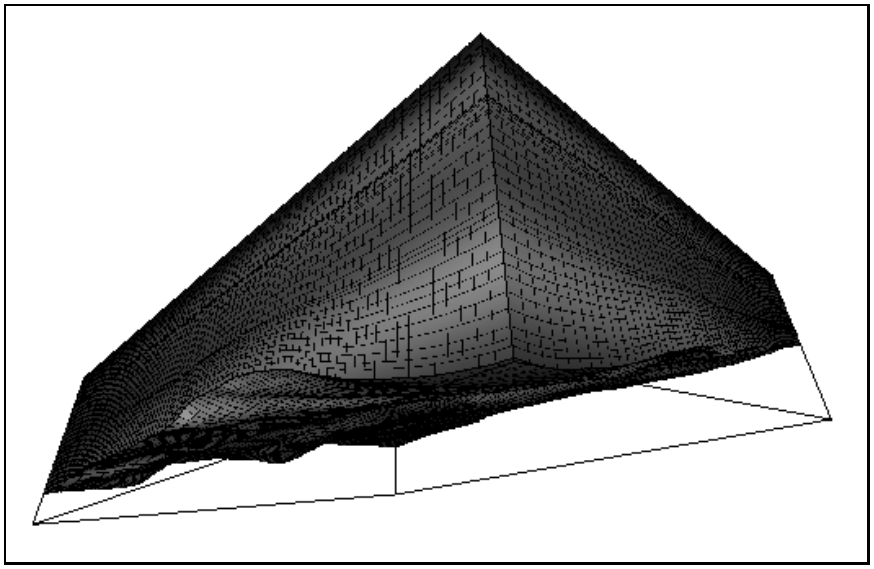


Abbildung 9: Die Temperatur wird interpoliert angezeigt (gouraud shading)

Als sehr nützlich haben sich die dreidimensionalen Eingabegeräte erwiesen, die eine einfache Navigation und komfortable Interaktion mit den Daten ermöglichen, die mit Hilfe einer Maus, die ja nur zwei Dimensionen bietet, nur über Umwege (z. B. Drücken der Maustaste plus Mausbewegung = Bewegung in die Tiefe) erreicht werden kann. Auf diesem Gebiet dürfte es auch in Zukunft interessante Entwicklungen, z. B. in Form neuer Force-Feedback Geräte, geben.

Danksagung

Ich möchte mich bedanken bei der Firma IES, die mir den Datensatz zur Verfügung gestellt hat, insbesondere bei Herrn Dr. T. Hantschel und bei Herrn B. Kurtenbach, die mir während des Aufenthaltes für Fragen zur Verfügung standen. Außerdem bei Herrn Dr. H. Zilken, Herrn S. Birmanns und Herrn F. Delonge, die mir im ZAM als Ansprechpartner zur Verfügung standen und mich mit Rat und Tat unterstützt haben.

Literatur

1. www.ies.de
2. www.fz-juelich.de/vislab
3. W. J. Schroeder, L. S. Avila, K. M. Martin, W. A. Hoffman, C. C. Law (2001): The Visualization Toolkit - User's Guide
4. W. Schroeder, K. Martin, B. Lorensen (1997): The Visualization Toolkit - An Object-Oriented Approach to 3D Graphics / 2nd Edition
5. www.kitware.com
6. B. Stroustrup (1997): The C++ Programming Language / Third Edition
7. B. Mohr (2001): Programming in C++. FZJ-ZAM-BHB-0154
8. J. H. Ferziger, M. Perić (1999): Computational Methods for Fluid Dynamics. Springer