

# Computerorientierte Quantenmechanik

## 1. Einführung

Marten Richter, EW 7 10

Tel: 3 14 - 24858

Email: [mrichter@itp.tu-berlin.de](mailto:mrichter@itp.tu-berlin.de)

Sprechstunde: n. V.

Thema: Was es nicht ist:

kein Programmier- und kein C-Kurs  
(Algorithm)

kein Quantencomputing

kein typisches theoretisches Fund

## Was es ist

Experiment: Verbindung der Praxis der  
computerorientierten Physik  
der Theoretischen Physik - mit der  
Informatik

Bunte Mischung an angewandter Informatik,  
Mathematik auf verschiedenen Gebieten der  
Theoretischen Physik.

## Das Programm (vorläufig)

1.) Einführung

Informatik { 2.) Rechnerarchitekturen

I. { 3.) Lineare Algebra für parallele Programmierung

- Physik
- II
- Infinite  
Mathe
- 4.) Lösung der Schrödinger gl. für Finite  
Differenz f. Exziton und Trion
- 5.) Iterationsverfahren: Lanczos Verfahren  
für Eigenwert. Krylov-Schur Verfahren
- 6.) Preconditioning von Matrizen
- Physik  
Quanten-  
III
- 7.) Basissystem der Quantenmechanik
- 8.) Konfiguration Interaktionsmethode
- 9.) Dichtefunktionaltheorie
- 10.) Bar-Optimum  $\rightarrow$  Optimierungsproblem
- Physik  
Infinite  
IV
- 11.) Zeitliche Dynamik: Diagonalmatrixtheorie  
Blockgleichung
- 12.) Beispiele für Zeitsolver
- 13.) Vielteilchendynamik, Koppelstärken,  
Coulombstreuung
- 14.) Stationäre Probleme und Nichtlineare  
Dynamik
- V
- 15.) Fourier Transform DFT / FFT
- 16.) Harmonik Inversion

## I.2) Rechnerarchitektur

Für effiziente Programmierung muss man

Wissen wie der Computer gebaut ist.  
(Sehr vereinfachte Darstellung)

## Die CPU

- Besteht aus einem Kern (Core)

Jeder Kern besteht aus  
mehreren Rechenwerken

- Fließkommeneinheiten

- SIMD (single instruction multiple data)

(mehrere gleichartige Instruktionen auf einmal)

z. B. bei Vektoroperationen z. B. MMX, SSE, SSE2  
AVX...

z. B.

$z_1 \mid z_2 \mid z_3 \mid z_4$

x

$y_1 \mid y_2 \mid y_3 \mid y_4$

//

$z_1 \times y_1 \mid z_2 \times y_2 \mid z_3 \times y_3 \mid z_4 \times y_4$

Optimiert Code  
hin

Vektoroperation  
(schwierig,  
gute Computer  
oder optimal  
BLAS z. B.  
f. lin. Algebra)

- Integer FPU

Bar: Mehrere Teile sind Cores FPU

Meist gibt es eine FPU, die die Befehle dekodiert  
und auf die Rechenwerke verteilt. (Teilweise autoperform)

(Bei Hyperthreading werden zwei Cores simuliert, um  
die FPU besser auszunutzen)

Wichtige Regel im Programmieren: Finde heraus, was am  
längsten im Programm dauert (z. B. Profiler)

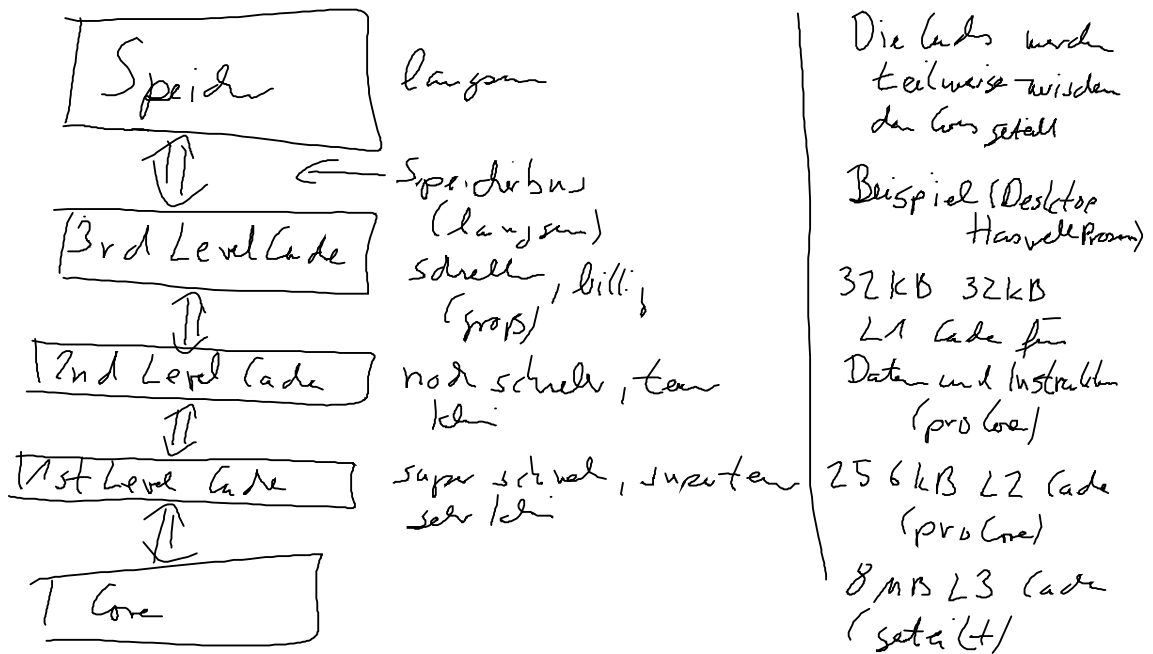
Finde heraus, ob es die Rechenzeit ist.

Dann optimieren nur diese Teil!

CPU



Aber meistens ist es Speicher ..... , den  
Speicherzugriff ist langsam.



Zu Vergleich Serverprozessoren Xeon (variante extreme Preis), 10 Core 25 MB L3 Cache

Konsequenzen für die Programmierung:

1.) Beliebte Optimierungstechnik: Tabellieren

=> 1) kleine Tabellen anlegen, die möglichst in niedrigeren Cachelevel passen

2) Lieber viele kleine Tabellen, die mit wenigen Rechenoperationen umgesetzt werden können als eine große Tabelle:

Beispiel

	$n[i] + s[j]$	
$n[i]$	$e$	tabellieren,
$s[j]$	$e$	
sonst	$e$	und $e$

2.) Speicherzugriff: Möglichst zusammenhängend  
Breite, die in den Cache passen.

- 3.) Passen die Daten nicht in den Cache  
 ist die Speicherbandbreite entscheidend  
 (Bei Speicherbandbreite an Limit  $\Rightarrow$  viele Redun)

### Organisation des Zwischenspeichers im Cache:

Die Daten werden nicht Byte für Byte abgerufen,  
 sondern in festen Abschnitten. Dies entspricht  
 oft den Seiten in der Speicherorganisation  
 (Es gibt virtuelle (durch Betriebssystem verwaltet)  
 und physisch) (z.B. Hardware 4KB Seite!)

Optimierungsziel Verweilzeit in einem Codeabschnitt  
 möglichst wenige Seiten!

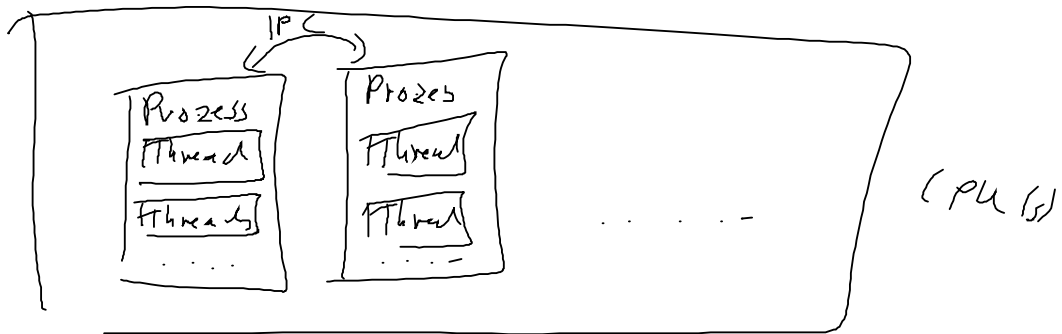
Das kann auf große Einheiten sein z.B. 1MB!  
 $\Rightarrow$  Wichtig ausprobiert und messen!

Beispiel 2D Array  $a[y \times \text{length} + x]$   
 Reihenfolge der Speicher

Der Unterschied kann sehr groß sein (Speicherordnung)  
 Umpadding (temporär) kann sich lohnen)

Parallelisierung auf einem Multicore

Programmausführung auf einem Multicore



Jeder Prozess hat sein eigen virtuelle Adressraum.  
 Ohne zusätzliche Mechanismen kann ein Prozess  
nicht auf der Speicher ein andere Prozessen zugreifen.  
 (Segmentation Fault (MMU))

Threads sind in gleiche Prozess und können  
 auf den gleichen Speicher zugreifen.

Im Prinzip 3 Möglichkeiten zur Parallelisierung  
 (auf einer Maschine)

- 1) Mehrere Threads in einem Prozess  
 (Kommunikation durch gemeinsamen Speicher  
 (Mutex, Locks ...))
- 2) Mehrere Prozesse  
 (Kommunikation durch Interprozesskommunikation  
 (geteilter Speicher, Pipes ...)) oder Netzwerkinterfaces ...
- 3) Mischform <sup>beide</sup>  
 Grundproblem bei beiden Varianten  
 • Synchronisation

Besonders problematisch ist das Warten

⇒ Ziel: Algorithmen verwenden, die Teilprobleme  
 bearbeiten, die kein von einander abhängen.

Berechnung von Threads/Prozessen sollten möglichst  
 wenig oder gar nicht von den anderen Threads/Prozessen  
 abhängen.