



# **Hochleistungsrechnen**

## Vorlesung im Wintersemester 2016/17

---

Skriptversion 25.10.2016

Prof. Dr. Thomas Ludwig

Universität Hamburg – Informatik – Wissenschaftliches Rechnen



# **Einleitung**

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

2



# **Themenüberblick**

Teil I: Hardware- und Software-Konzepte

Teil II: Programmierung

Teil III: Programmierwerkzeuge

Teil IV: Allgemeine Fragestellungen



## Teil I: Hardware- und Software-Konzepte

- Klimaforschung und Hochleistungsrechnen (8-50)
- Hardware-Architekturen (51-83)
- Die TOP500-Liste (84-131)
- Vernetzungskonzepte
- Hochleistungs-Eingabe/Ausgabe
- Betriebssystemaspekte (132-164)



## Teil II: Programmierung

- Programmierung mit OpenMP (165-205)

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

5



## Teil III: Programmierwerkzeuge

- Optimierung sequentieller Programme (206-247)
- Fehlersuche (248-286)

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

6



## Teil IV: Allgemeine Fragestellungen

- Kosten-Nutzen-Analyse (287-333)

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

7



# **Klimaforschung und Hochleistungsrechnen**

1. Der Erkenntnisgewinnungsprozess
2. Klimamodellierung und Computer
3. Rechner- und Speicherinfrastruktur am DKRZ
4. Themengebiete der Informatik
5. Herausforderungen

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

8

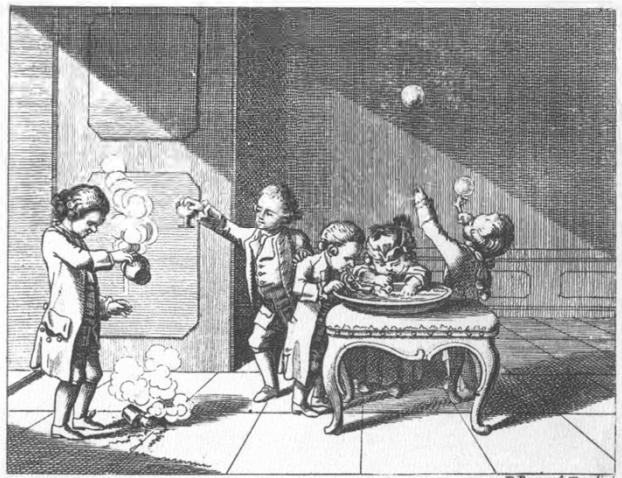


# 1. Der Erkenntnisgewinnungsprozess

- Das Experiment
- Die Theorie
- Die Dritte Säule: Numerische Simulation
- The Fourth Paradigm: Datenintensive Wissenschaft

# Das Experiment

„Ein **Experiment** (von lateinisch *experimentum* „Versuch, Beweis, Prüfung, Probe“) im Sinne der Wissenschaft ist eine methodisch angelegte Untersuchung zur **empirischen** Gewinnung von Information (*Daten*).“ (Wikipedia)



25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

10

Siehe z.B. <http://de.wikipedia.org/wiki/Experiment> . Wird hier aber nicht vertieft.

## Die (wissenschaftliche) Theorie

„Eine **Theorie** ist ein System von Aussagen, das dazu dient, **Ausschnitte der Realität** zu beschreiben beziehungsweise zu erklären und **Prognosen über die Zukunft** zu erstellen.“

(Wikipedia)

$$\begin{aligned} T_{gb} &= \frac{1}{4\pi} \int_0^{2\pi} \int_{-1}^1 T_i \, d\mu \, d\varphi \\ &= \frac{1}{4\pi} \int_0^{2\pi} \int_0^1 \sqrt[4]{\frac{S_o (1 - \alpha_o) \mu}{\epsilon \sigma}} \, d\mu \, d\varphi \\ &= \frac{1}{4\pi} \left[ \frac{S_o (1 - \alpha_o)}{\epsilon \sigma} \right]^{0.25} \int_0^{2\pi} \int_0^1 \mu^{0.25} \, d\mu \, d\varphi \\ &= \frac{2}{5} \left[ \frac{S_o (1 - \alpha_o)}{\epsilon \sigma} \right]^{0.25} \end{aligned} \quad (5)$$

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

11

Siehe z.B. <http://de.wikipedia.org/wiki/Theorie>. Wird hier aber nicht vertieft.

## Numerische Simulation (Die Dritte Säule)

„Als **numerische Simulation** bezeichnet man allgemein Computersimulationen, welche mittels numerische Methoden wie zum Beispiel mit Turbulenzmodellen durchgeführt werden. Bekannte Beispiele sind Wetter- und Klimaprognosen, numerische Strömungssimulation oder Festigkeits- und Steifigkeitsberechnungen.“ (Wikipedia)

- Computersimulation dient in sehr vielen modernen Wissenschaften als Methode der Erkenntnisgewinnung
- Die wissenschaftstheoretischen Probleme sollen hier nicht erörtert werden

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

12

Siehe

- [http://de.wikipedia.org/wiki/Numerische\\_Simulation](http://de.wikipedia.org/wiki/Numerische_Simulation) . Muss dringend ausgebaut werden
- Gabriele Gramelsberger: Computerexperimente: Zum Wandel der Wissenschaft im Zeitalter des Computers (2010)
- Wissenschaftsrat 07/2014 zu „Simulation in der Wissenschaft besser nutzen“ <http://www.wissenschaftsrat.de/index.php?id=1234&L=>
- Wissenschaftsrat 07/2014: Bedeutung und Weiterentwicklung von Simulation in der Wissenschaft – Positionspapier .  
<http://www.wissenschaftsrat.de/download/archiv/4032-14.pdf>

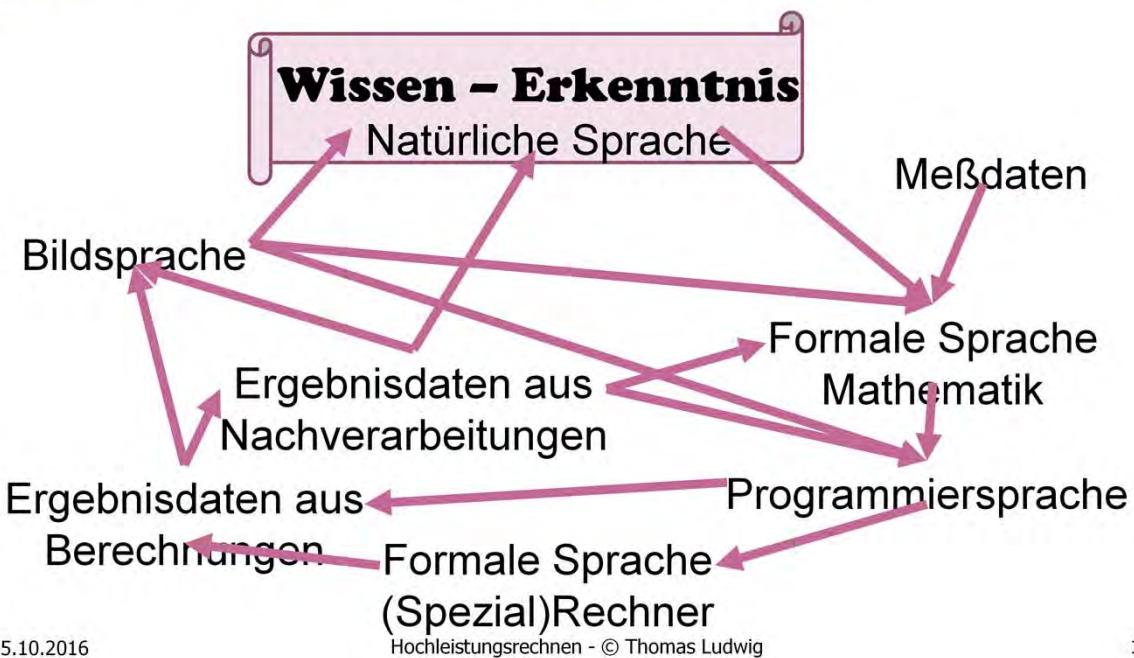


## Numerische Simulation

Ganz grob zur Vorgehensweise

- Wir haben eine Vorstellung der Zusammenhänge in einem System, die sich mathematisch darstellen lässt
- Mathematische Darstellung wird in ein Programm überführt
- Das Programm berechnet Ergebnisdaten
- Ergebnisdaten werden mit der Realität verglichen

# „Die Dialekte der Klimaforschung“



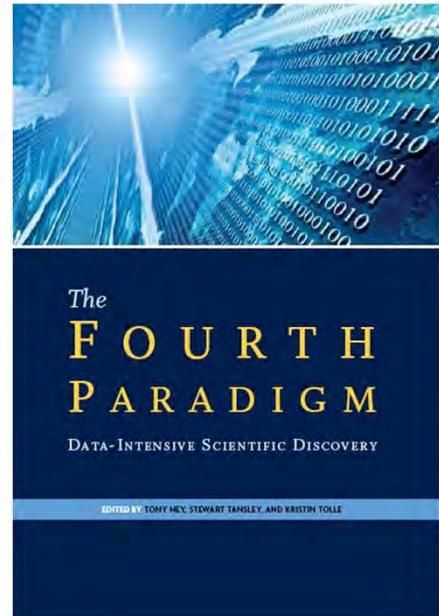
Siehe Unterlagen zum Workshop „Die Dialekte der Klimaforschung“  
<https://www.dkrz.de/Klimaforschung/dkrz-und-klimaforschung/theorie/dialekte>

## Datenintensive Wissenschaft (Fourth Paradigm)

"Increasingly, scientific breakthroughs will be powered by advanced computing capabilities that help researchers **manipulate** and **explore** massive **datasets**.

The speed at which any given scientific discipline advances will depend on how well its researchers collaborate with one another, and with technologists, in areas of **eScience** such as databases, workflow management, visualization, and cloud computing technologies."

Microsoft Research



25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

17

Siehe:

- <http://research.microsoft.com/en-us/collaboration/fourthparadigm/> .  
PDF des Buchs frei erhältlich.

## Datenintensive Wissenschaft

- Datenintensive Wissenschaft ist ein Teil von dem, was jetzt diffus als „Big Data“ bezeichnet wird
- Big Data meint meistens große unstrukturierte Datensätze aus potentiell verschiedenen Quellen
  - Z.B. Erkennen von Krankheitsausbreitungen durch entsprechende Eingaben in Google („Grippemittel“)
- Wichtigste Gemeinsamkeit: aus den Daten alleine werden neue Einsichten gewonnen

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

18

## **2. Klimamodellierung und Computer**

Für ein erstes intuitives Verständnis

- Ein Klimamodell ist repräsentiert durch einen Satz von Programmen, mit deren Hilfe ein Klimageschehen simuliert wird
- Typischerweise werden lange Zeiträume auf globaler Ebene simuliert
- Hierfür benötigen wir sehr hohe Rechenleistung und große Speichersysteme

Siehe z.B. Hans von Storch: „Das Klimasystem und seine Modellierung: Eine Einführung“

# Unterschied Wetter vs. Klima

## Wetter

„Als Wetter bezeichnet man den spürbaren, kurzfristigen Zustand der Atmosphäre an einem bestimmten Ort der Erdoberfläche, der unter anderem als Sonnenschein, Bewölkung, Regen, Wind, Hitze oder Kälte in Erscheinung tritt.“ (Wikipedia)

## Klima

„Klima ist die Gesamtheit aller an einem Ort möglichen Wetterzustände, einschließlich ihrer typischen Aufeinanderfolge sowie ihrer tages- und jahreszeitlichen Schwankungen.“ (Wikipedia)

**„Klima ist das 30-Jahres-Mittel des Wetters“ – mithin ein mathematisches Konstrukt**

# Wetter- und Klimasimulationen im Computer

- Erste Wettersimulationen 1950
  - Charney, Fjørtoft, von Neumann
    - Erste rechnergestützte 24-Stunden-Wetterprognose
    - Auf einem der ersten Rechner, der ENIAC
    - NWP – numerical weather prediction
- Erste Klimasimulation 1956
  - Princeton Institute for Advanced Studies
    - Realistische monatliche und saisonale Strukturen
    - 2 Schichten, 17x16 Gitterpunkte
    - Computer: 1KB Hauptspeicher, 2 KB Magnetspeicher
    - GCM – global circulation model

25.10.2016

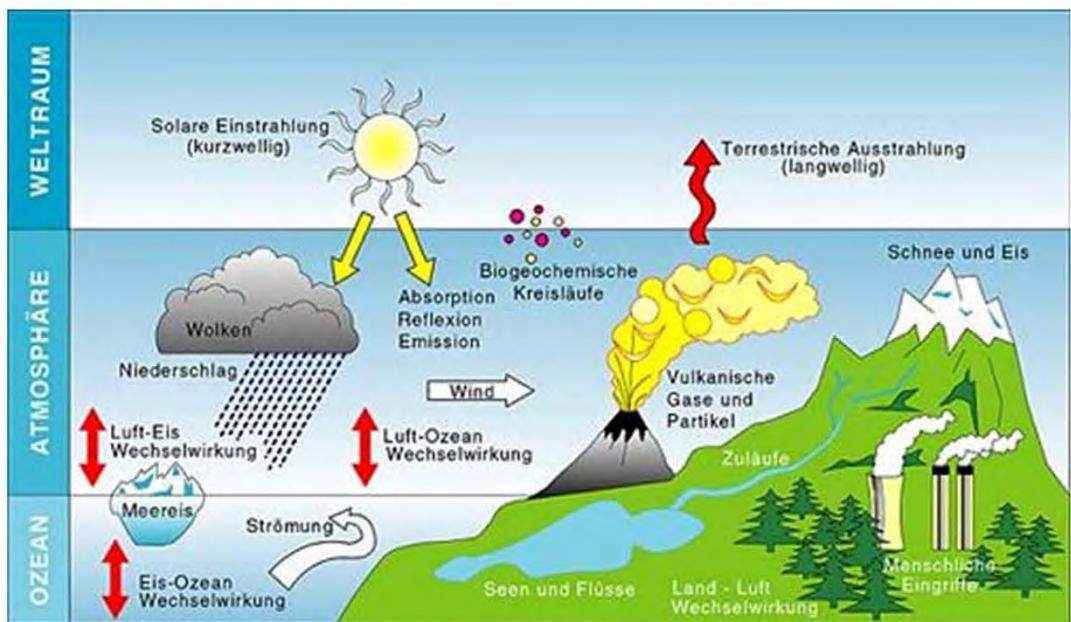
Hochleistungsrechnen - © Thomas Ludwig

21

Siehe:

- [http://de.wikipedia.org/wiki/John\\_von\\_Neumann](http://de.wikipedia.org/wiki/John_von_Neumann)
- <http://de.wikipedia.org/wiki/ENIAC>

# Komponenten im Klimasystem



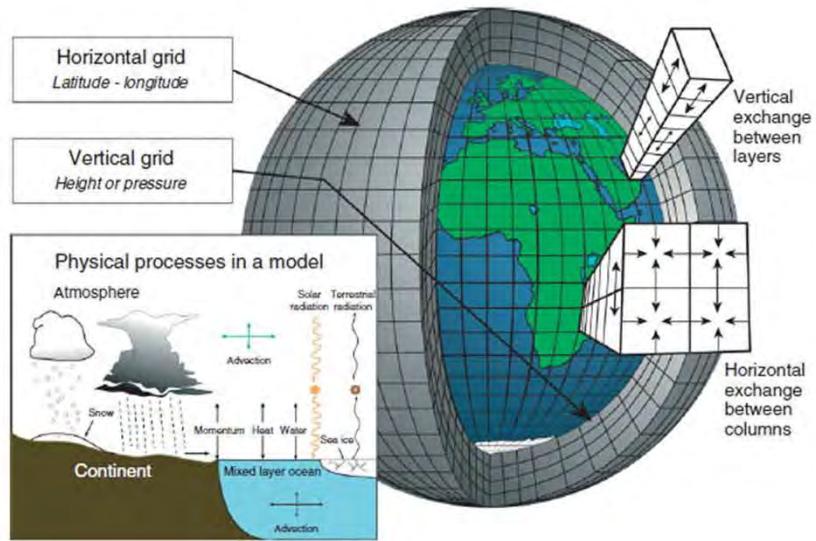
25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

22

# Numerischer Ansatz

- Diskretisierung des Raumes: Einteilung in Gitterzellen
- Diskretisierung der Zeit: feste Schritte der simulierten Zeit
- Halbierung des Gitterabstandes erfordert auch Halbierung des Zeitschrittes
- Erfordert 16-fache Rechnerleistung !!

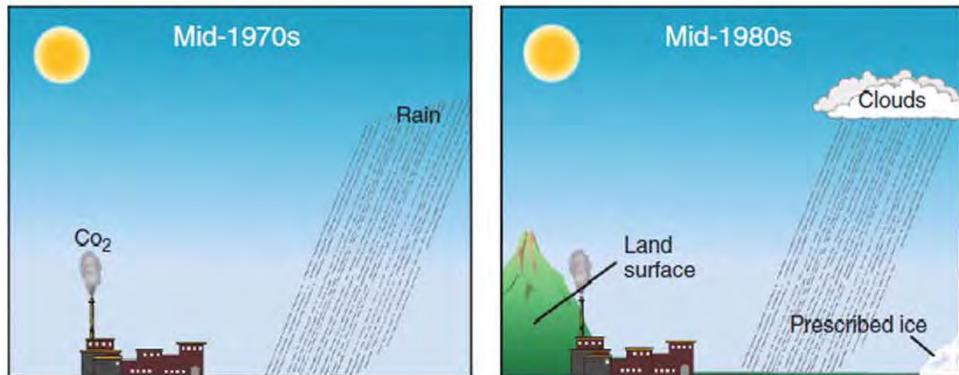


25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

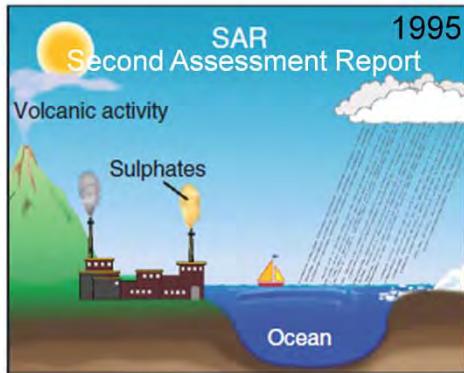
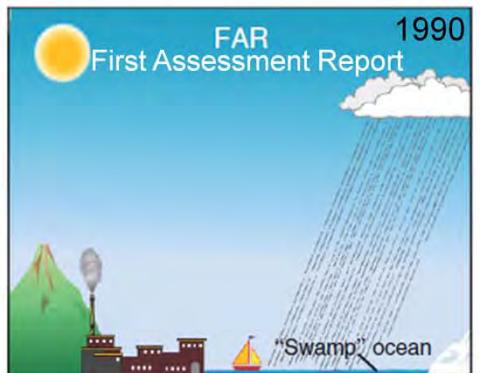
23

# Evolution der Klimamodelle



- Auflösungen:  
Horizontal  $5^\circ$   
Vertikal 12 Schichten
- Auflösungen:  
Horizontal  $\sim 5^\circ$   
Vertikal 16 Schichten

## Evolution der Klimamodelle (2) (IPCC)



- Auflösungen:
  - Horizontal 200-1000 km
  - Vertikal 2-20 Schichten
- Berechnung bis zu 10 000 Jahre

- Auflösungen:
  - Horizontal: 250 km
  - Vertikal 1 km

25.10.2016

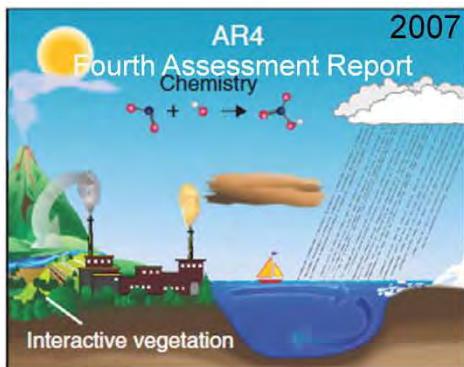
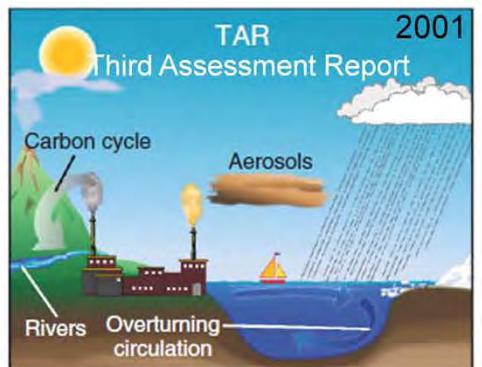
Hochleistungsrechnen - © Thomas Ludwig

25

Siehe:

- [http://www.ipcc.ch/ Intergovernmental Panel on Climate Change](http://www.ipcc.ch/)

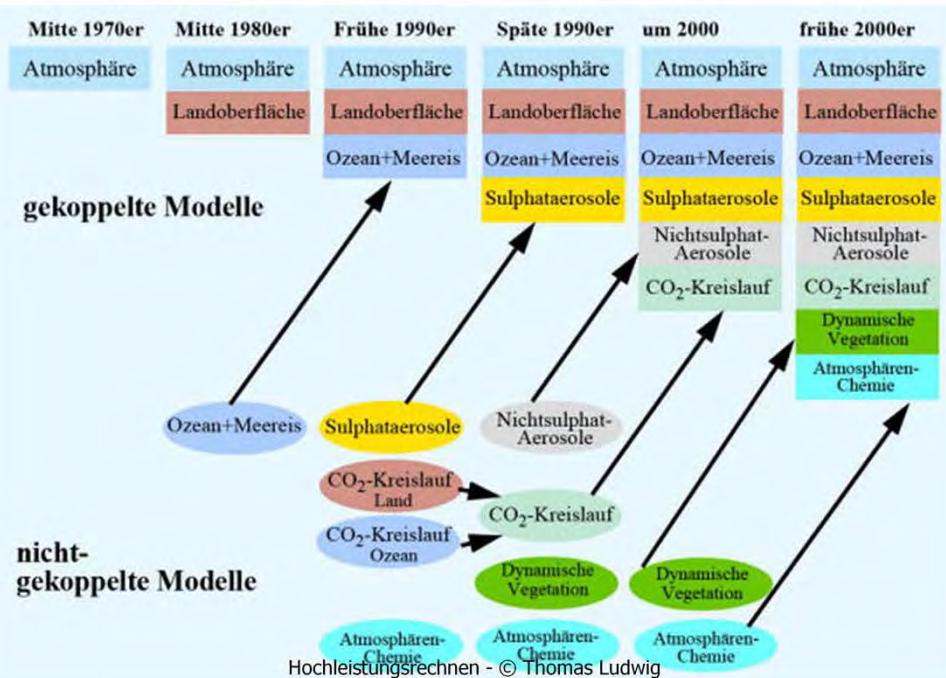
## Evolution der Klimamodelle (3) (IPCC)



- Auflösungen:  
Horizontal 140 km

- Auflösungen  
Horizontal: 110 km  
Vertikal 60 Schichten  
je 30 Ozean und Atmosphäre

# Prozesse im Klimamodell



25.10.2016

27

## Datenaufkommen bei IPCC-AR

CMIP – Coupled Model Intercomparison Project

- CMIP3 / IPCC-AR4 (Report (2007)
  - 17 beteiligte Zentren mit 25 Klimamodellen
  - Insgesamt 36 TB Modelldaten,  $\frac{1}{2}$  TB bei IPCC/DDC
- CMIP5 / IPCC-AR5 (Report 2013/14)
  - 29 beteiligte Gruppen mit 61 Modellen
  - Produzierte Datenmenge: ca. 10 PB, davon 640 TB aus HH
  - Datenvolumen IPCC/DDC: 1,6 PB
- Status CMIP5-Daten in gemeinsamen Archiven
  - 2,3 PB für 69.000 Datensätze in 4,3 Mio. Dateien in 23 Datenknoten
  - CMIP5 ist mehr als 50mal umfangreicher als CMIP3

**Extrapolation CMIP6: 150 PB in 280 Mio. Dateien**

25.10.2016

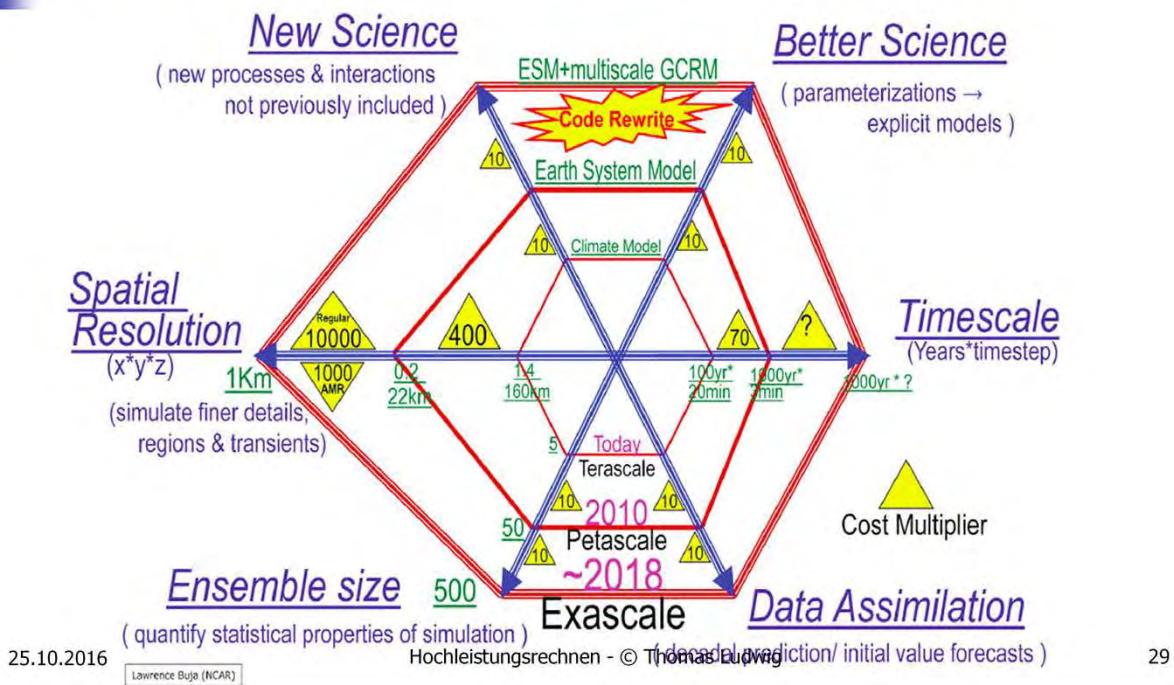
Hochleistungsrechnen - © Thomas Ludwig

28

Siehe:

- <http://www.wcrp-climate.org/index.php/wgcm-cmip/about-cmip>

# Der Big Bang der Klimamodellierung



## Neueste Entwicklungen: HP(CP)<sup>2</sup>-Projekt

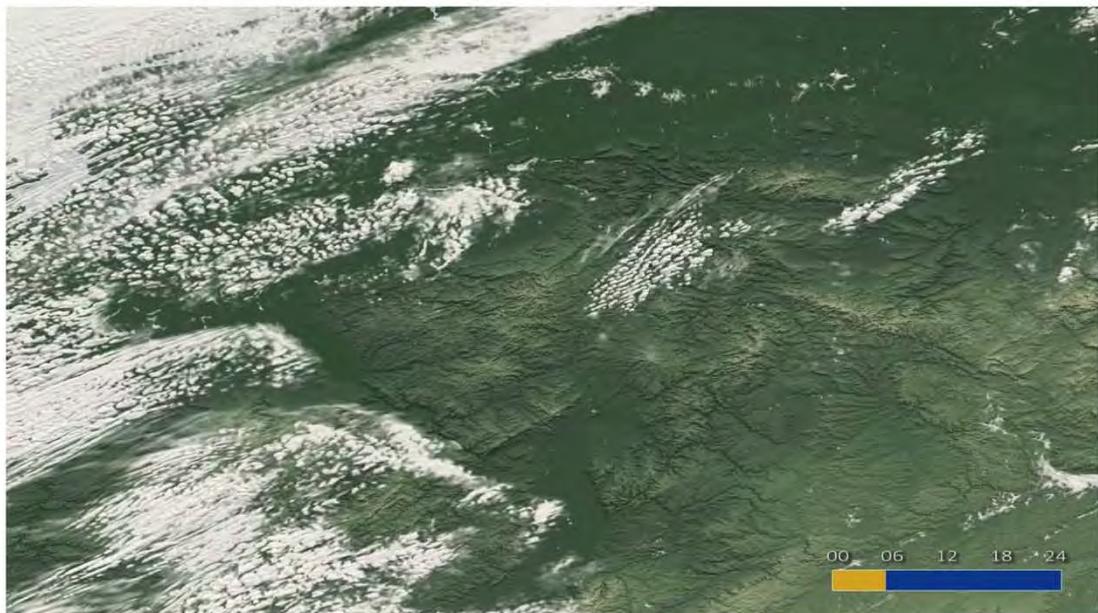
- High Definition Clouds and Precipitation for advancing climate Prediction
  - Gefördert vom BMBF
- Endlich Cloud-Computing ☺
  - Wolken werden berechnet, nicht mehr parametrisiert
- Auflösung auf bis zu 100m Gitterweite in einer Berechnungsbereich von 1000 x 1000 km
- $10^8$  Gitterpunkte horizontal [globale Gitterweite von etwa 2,2 km]
- Bedarf sehr hoher Rechen- und Speicherleistung

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

30

## Neueste Entwicklungen: HP(CP)<sup>2</sup>-Projekt...



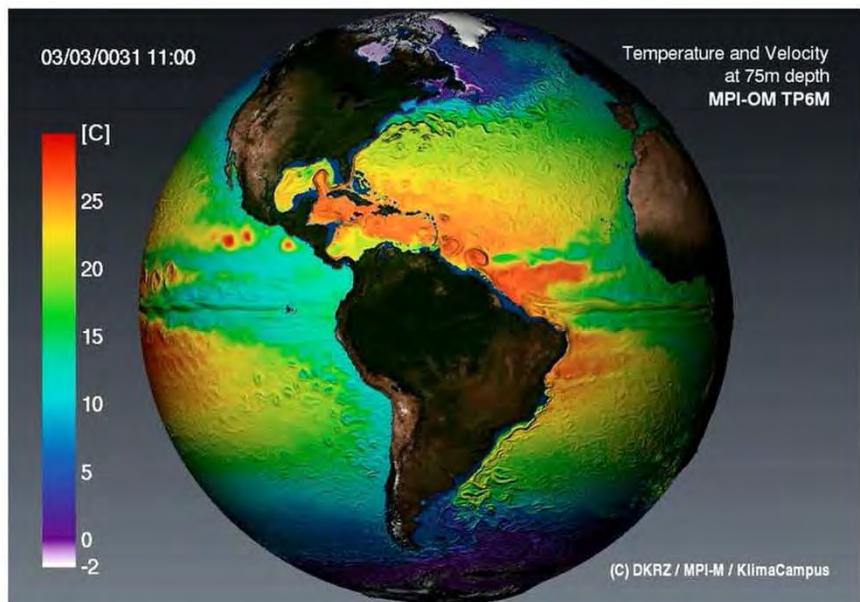
25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

31

# Ergebnisvisualisierung

- Visualisierungen in 2D und 3D von sehr großen Datenmengen
- Am liebsten auch während des Programmlaufs



25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

32

### **3. Rechner- und Speicherinfrastruktur**

Deutsches Klimarechenzentrum GmbH (DKRZ)

- Gegründet 1987 als nationale Einrichtung
- Betrieben von 4 Gesellschaftern:
  - MPG (55%), FHH/UHH (27%), HZG, AWI
- Ca. 70 Mitarbeiter

Rechner- und Speicherinfrastruktur

- Alle ca. 5 Jahre mit externer Finanzierung (BMBF, HGF)

**Leitbild**  
**DKZB – Partner der Klimaforschung**

**Höchste Rechenleistung.**  
**Ausgereiftes Datenmanagement.**  
**Kompetenter Service.**

**Vision**

**Das DKRZ erschließt der Klimaforschung verlässlich das Potenzial des sich beschleunigenden technischen Fortschritts.**

25.10.2016  
FHH BWF UNI

Hochleistungsrechnen - © Thomas Ludwig  
Deutsches Klimarechenzentrum Bundesstraße 45 Januar 2008

Lehmann + Partner Architekten  
34

## 29 Jahre DKRZ (1987-2016)

Erster Computer: Control Data Cyber-205

- 1 Prozessor, 200 MFLOPS, 32 MB Hauptspeicher
- 2.5 GB Festplatten, 100 GB Bandarchiv

**200 MFLOPS hat ein modernes Smartphone**

Aktuelle Maschine: Bull/Atos (HLRE-3)

- 100.000+ Prozessorkerne, 3,6 PFLOPS, 240 TB Hauptspeicher
- 54 PB Festplatten, 300-500 PB Kapazität im Bandarchiv

**TOP500-Liste: Faktor 1.000 alle 12,5 Jahre**

## Abgeschaltetes Rechnersystem Blizzard

- IBM Power6, installiert 2009
- Spitzenleistung: 158 TFLOPS, Linpack 115 TFLOPS
- 264 IBM Power6-Rechnerknoten (dualcore, 16-socket)
- 8.448 Prozessorkerne
- Über 26 TB Hauptspeicher – 6+ PB auf Platten



25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

36

## Neues Rechnersystem Mistral

- bullx B700 DLC, installiert 2015
- Spitzenleistung: ca. 3,6 PFLOPS
- Ca. 3.000 Dualsocket-Rechnerknoten
- Intel Haswell (12-core) und Broadwell (18-core)
- Ca. 100.000 Prozessorkerne
- Über 240 TB Hauptspeicher – 54 PB auf Platten



25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

37

## Aktuelles Bandarchiv

- 7 Sun StorageTek SL8500 Bandbibliotheken
- 67.000+ Stellplätze für Bänder (+10.000 in Garching)
- Ca. 80 Bandlaufwerke
- 130 PB Kapazität mit LTO6-Bändern (ca. 40 PB belegt)



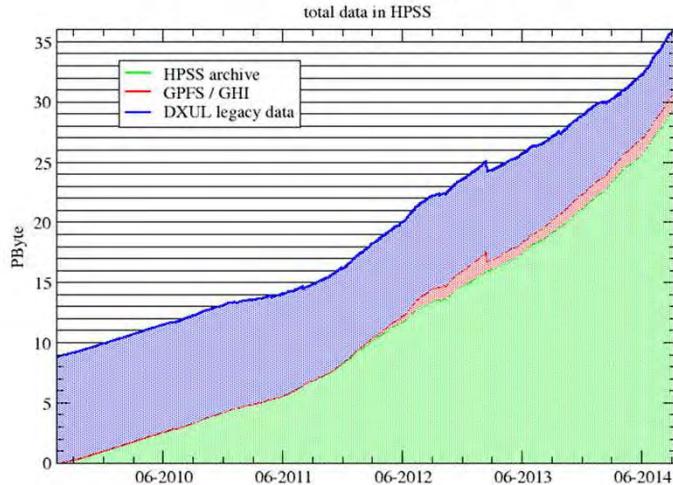
25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

38

# Datenvolumen

Aktuell ca. 8 PB/Jahr  
DKRZ tape archive



25.10.2016

Wed Sep 10 14:00:00 2014

Hochleistungsrechnen - © Thomas Ludwig

39

## Ausbau von HLRE-2 zu HLRE-3

Charakteristikum	2009	2015	Faktor
Leistung	150 TFLOPS	3,6 PFLOPS	<b>24x</b>
Rechnerknoten	264	3.000	12x
Hauptspeicher	20 TB	240 TB	12x
Festplattenkapazität	6 PB	54 PB	<b>9x</b>
Durchsatz Hauptspeicher-Festplatten	30 GB/s	400 GB/s	13x
Kapazität Bandbibliothek (2015, 2020)	120 PB	360 PB	3x
Durchsatz Festplatten-Bandbibliothek	10 GB/s	20 GB/s	2x
Leistungsaufnahme (mit Kühlung)	1.6 MW	1.4 MW	0.9x
Investitionskosten	30M€	35M€	1,2x

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

40

# Dienste am DKRZ

- Abteilung Anwendungen
  - Fehlersuche in parallelen Programmen
  - Leistungsanalyse und -optimierung
  - Verbesserung der Datenein-/ausgabe
  - Verbesserung der Skalierung der Programme
  - Evaluierung neuer HW-Konzepte (z.B. Grafikkarten)
  - Visualisierung
- Abteilung Datenmanagement
  - Qualitätssicherung der Daten
  - Zuteilung von dauerhaften IDs zu Datensätzen
  - Langzeitarchivierung
  - World Data Center Climate (Datenbereitstellung für Dritte)
- Abteilung Systeme
  - Unterstützung Betrieb Hochleistungsrechner und Archiv
  - Evaluierung und Betrieb neuer Speicherkonzepte (z.B. Cloud-Storage)
  - Unterstützung reguläre IT im DKRZ

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

41

## **4. Themengebiete der Informatik**

Denkbar sind speziell Ausrichtungen auf Klimaforschung bei

- Hardware
- Software
- Brainware (auch Peopleware, Wetware)



## Hardware

- HW (Prozessoren, Rechner) und Klimaforschung
  - Spezielle Anpassungen für die Numerik der Klimaforschung sind denkbar
  - Allerdings: es gibt keinen Markt, also wird auch kein Hersteller so etwas gezielt bauen
- Programmierbare Spezialprozessoren (FPGAs)
  - Möglich, aber insgesamt zu kompliziert und zu kostspielig

# Software – von höchster Bedeutung!

## Problemfelder bei Klimaforschung

- Code ist nie abgeschlossen
- Vorhandene Codebasis ist riesig: 20+ Jahre Entwicklung
- Fortran ☺
- Effiziente Datenein-/ausgabe ist nur bedingt erzielt
- Effizienzausbeute des Prozessors begrenzt
- Skalierbarkeit der Modelle kompliziert
- Modellkopplung komplex, endet in Mehrprogrammcode
- Visualisierung zunehmend schwierig
- Datenmanagement noch nicht gelöst

## Software – wo sind Lösungen ?

- Softwaretechnik, Softwareengineering
  - Wenige Ansätze für diesen Typ Software: Scientific SW
- Betriebssystemtechnik
  - Kaum Lehre zu Eingabe/Ausgabe, parallel E/A kaum beachtet
- Programmierung komplex
  - Wo sind gute Compiler, Bibliotheken, Programmiermodelle
- Datenmanagement als Thema in der Lehre existiert nicht
  - Trotz dem Aufkommen von Big Data!



## Brainware

- Hohe Investitionen in Hardware (schön! ☺)
- Aber
  - Nicht effizient nutzbare Hardware ist verschwendetes Geld
- Stattdessen
  - Angemessene Anteile in Personal investieren
  - Ausbildung in Bereichen wie z.B. Programmierung, Fehlersuche, Leistungsoptimierung
  - Könnte wissenschaftliche Produktivität mit dem Rechnersystem erhöhen

## 5. Herausforderungen

- Der technische Fortschritt in der Hardware der Informationssysteme hat ein exponentielles Wachstum
- Sowohl Informatiker als auch Nutzer müssen sich ständig mit neuen Konzepten befassen
- Auseinanderlaufen der Kompetenzen durch fehlende institutionalisierte Zusammenarbeit
- Erfolg und Wettbewerbsfähigkeit der HPC-nutzenden Wissenschaften hängt an der optimalen Ressourcennutzung der Systeme

## Besondere Teilprobleme für Klimaforschung

- Rechenleistung und Hauptspeicherausbau wächst viel schneller an als Speicherkapazität der Platten und die Zugriffsgeschwindigkeit
  - Es wird schwieriger, balancierte Systeme zu betreiben
  - Klimaforschung benötigt aber viele Speicherressourcen
- Die Anzahl der Prozessorkerne wächst sehr schnell an
  - Es wird schwieriger, die Systeme in vollem Umfang effizient zu nutzen
  - Klimaforschung hat aber sehr langlaufende Modelle

## **Klimaforschung und Hochleistungsrechnen**

### **Zusammenfassung**

- Durchführbare Forschung steht in engem Zusammenhang zu vorhandenen Ressourcen an Hardware – Software – Brainware
- In den vergangenen Jahrzehnten konnten Fortschritte in Hardware ohne komplexe Anpassungen direkt genutzt werden – dies ist aktuell nicht möglich
  - Folge: nur hoher Aufwand für die Software sichert weitere Fortschritte
- Das Softwareproblem ist nur durch intensives interdisziplinäres Arbeiten zu bewältigen

# Klimaforschung und Hochleistungsrechnen

## Die wichtigsten Fragen

- Welche Wege der Erkenntnisgewinnung kennen wir in der modernen Wissenschaft?
- Welche allgemeinen Probleme sehen Sie bei der numerischen Simulation von Systemen?
- Welchem Ansatz folgt die numerische Simulation in der Klimaforschung?
- Welche Evolution sieht man in der rechnergestützten Klimamodellierung?
- Welche typischen Probleme beim Umgang mit Software bei der Klimamodellierung kennen Sie?
- Welche künftigen Herausforderungen sehen Sie bei der Klimasimulation?



# Hardware-Architekturen

1. Parallelismus
2. Klassifikation nach Flynn
3. Gemeinsamer und verteilter Speicher
4. Skalierbarkeit
5. Verbindungsnetze und Topologien
6. Hintergrundspeicher
7. Spezialkonzepte

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

51

## **1. Parallelismus**

Unter Parallelverarbeitung verstehen wir die parallele Verwendung (gleichartiger) Komponenten zur Erzielung höherer Leistung

### Beispiel

- Ein Arbeiter schaufelt ein 1m\*1m\*1m großes Loch in einer Stunde
- Zwei Arbeiter schaufeln ein 1m\*1m\*1m großes Loch in einer halben Stunde
- Tausend Arbeiter? ☹
- Ein 1000m\*1m\*1m großes Loch? ☺

# Ebenen des Parallelismus im Rechner

- Parallele Rechnerarchitekturen
  - Besitzen Verarbeitungseinheiten, die koordiniert gleichzeitig an einer Aufgabe arbeiten
- Verarbeitungseinheiten
  - (Auch die Bits in einem Byte/Wort)
  - Spezialisierte Einheiten wie z.B. Rechenwerke
  - Prozessorkerne
  - Prozessoren
  - Vollständige Rechner
  - Hochleistungsrechnersysteme



## Ebenen des Parallelismus (2)

- Vernetzung der Rechner
  - Viele parallele Wege zwischen zwei Verbindungspunkten
- Datenspeicherung
  - Viele Festplatten zu einem Verbund geschaltet
  - Viele Bandlesegeräte zu einem Verbund geschaltet

## Ebenen des Parallelismus (3)

### Prozessortechnologie bis ca. 2005

- Erhöhung der Frequenz → höhere Leistung
  - Entspricht quasi stärkerem Bauarbeiter
- Zusätzliche Prozessoren → noch mehr Leistung

### Prozessortechnologie ab ca. 2005

- Frequenzerhöhung nicht weiter möglich, da Abwärme zu hoch – weiter Miniaturisierung klappt allerdings
  - Deshalb: mehrere vollständige Teilprozessoren (Kerne) in einem Prozessor

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

55

Siehe:

- [https://en.wikipedia.org/wiki/Microprocessor#Multi-core\\_designs](https://en.wikipedia.org/wiki/Microprocessor#Multi-core_designs)
- [https://en.wikipedia.org/wiki/Clock\\_rate](https://en.wikipedia.org/wiki/Clock_rate)
- [https://en.wikipedia.org/wiki/Multi-core\\_processor](https://en.wikipedia.org/wiki/Multi-core_processor)

Die Leistungsaufnahme eines Prozessors wächst quadratisch mit seiner Betriebsfrequenz.

## Ebenen des Parallelismus (4)

Prozessortechnologie der 80er Jahre

- Thinking Machines Corporation produziert die Connection Machine
- CM-1 (1985): 65.536 Ein-Bit-Prozessoren

Seymour Cray (1925-1996)

*If you were plowing a field, what would you rather use? Two strong oxen or 1024 chickens?*

W. Gropp, E. Lusk, A. Skjellum

*To pull a bigger wagon, it is easier to add more oxen than to grow a giant ox.*

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

56

Siehe:

- [https://en.wikipedia.org/wiki/Connection\\_Machine](https://en.wikipedia.org/wiki/Connection_Machine)
- [https://en.wikipedia.org/wiki/Thinking\\_Machines\\_Corporation](https://en.wikipedia.org/wiki/Thinking_Machines_Corporation)

## 2. Klassifikation nach Flynn

### Klassifikation nach Flynn (1972)

- Rechner arbeiten mit Befehlsströmen und Datenströmen
  - Aus ihrer Kombination ergeben sich 4 Varianten
- 
- SISD single instruction, single data stream
  - SIMD single instruction, multiple data stream
  - MISD multiple instruction, single data stream
  - MIMD multiple instruction, multiple data stream

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

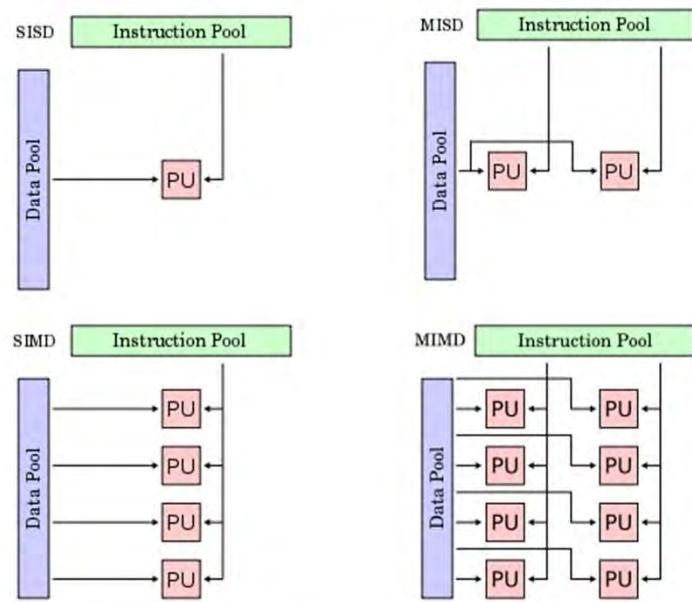
57

Siehe:

- [http://en.wikipedia.org/wiki/Flynn%20s\\_Taxonomy](http://en.wikipedia.org/wiki/Flynn%20s_Taxonomy)

Die Klassifikation ist historisch. Sie dient hier einer ersten Unterteilung unserer Rechner in verschiedene Klassen, genügt aber nicht den aktuellen Anforderungen und kann die aktuellen Systeme nicht adäquat erfassen.

## Klassifikation nach Flynn (2)



25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

58

Quelle: Wikimedia Commons (<http://en.wikipedia.org/wiki/File:SISD.svg>,  
<http://en.wikipedia.org/wiki/File:MISD.svg>,  
<http://en.wikipedia.org/wiki/File:SIMD.svg>,  
<http://en.wikipedia.org/wiki/File:MIMD.svg>)

## Klassifikation nach Flynn (3)

### Was ist was bei Flynn?

- SISD: klassische von-Neumann-Architektur  
Monoprozessor-Rechner
  - quasi ausgestorben
- SIMD: Vektorrechner und Feldrechner
  - umgesetzt in Spezialarchitekturen wie z.B. Grafikkarten
- MISD: diese Klasse ist leer
- MIMD: alles, was uns interessiert
  - die Mehrprozessorsysteme

## Unterteilung von Flynn's MIMD-Klasse

Die Rechner bestehen aus mehreren Prozessoren, die über ein Verbindungsnetz kommunizieren

- Über die Verbindungen erfolgt der Informationsaustausch zwischen Prozessen auf verschiedenen Prozessoren sowie Synchronisation und Kooperation

### Neue Unterscheidungskriterien

- Wie sehen die Prozessoren den Adressraum des Speichers?
- Wie sind die Speicherkomponenten mit dem Prozessor gekoppelt?

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

60

Moderne Prozessoren enthalten jetzt fast immer mehrere Prozessorkerne, die wie Prozessoren geringerer Leistungsfähigkeit arbeiten. Dies verkompliziert unsere Betrachtungen. Später mehr dazu.

Siehe:

- [https://en.wikipedia.org/wiki/Multi\\_core](https://en.wikipedia.org/wiki/Multi_core)

Ein Prozess ist ein auf einem Prozessor ablaufendes Programm. Ein Prozessor mit z.B. vier Prozessorkernen kann vier Prozesse echt gleichzeitig abarbeiten. Daneben werden ja auf jedem Einzelprozessor normalerweise auch mehrere Prozesse zeitlich verzahnt (also quasi-gleichzeitig) abgearbeitet.

## 3. Gemeinsamer & verteilter Speicher

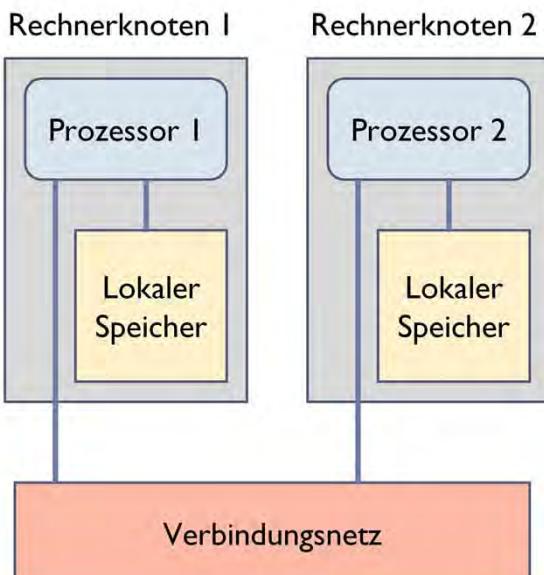
Neue Klassen von Rechnerarchitekturen

- Rechner mit verteiltem Speicher
- Rechner mit gemeinsamem Speicher
- Mischformen

Mischformen sind in der Informatik sehr beliebt

- Man versucht, die Vorteile der Ansätze zu vereinen, ohne die Nachteile in Kauf nehmen zu müssen

## Mehrprozessorsysteme mit verteiltem Speicher



- Prozesse sehen nur den Adressraum im lokalen Speicher
- Leistungssteigerung:  
Dasselbe Programm läuft parallel auf allen Prozessoren; seine Daten sind auf die lokalen Speicher der Rechnerknoten aufgeteilt
- In dieser klassischen Form ausgestorben

25.10.2016

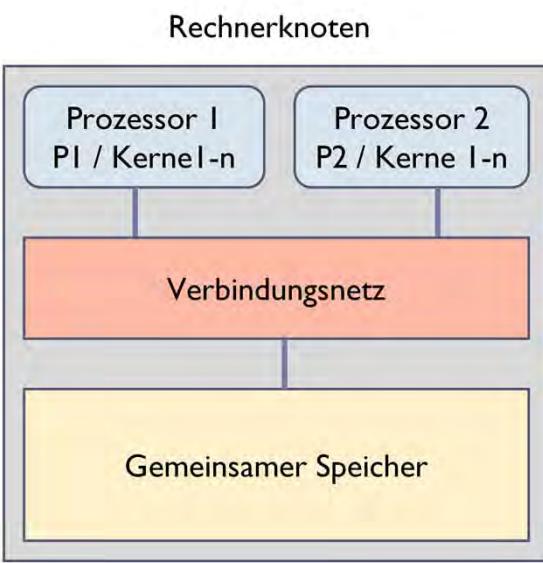
Hochleistungsrechnen - © Thomas Ludwig

62

Im uns am besten bekannten Fall ist der Rechnerknoten ein einzelner Rechner (z.B. PC) und das Verbindungsnetz ein Ethernet-Netz. Bei Hochleistungsrechnern ist der Rechnerknoten ein Einschub in einem Rack (oder auch nur ein Teil eines solchen Einschubs) und das Verbindungsnetz ist etwas Hochspezielles.

Da es heute praktisch nur noch Mehrkernprozessoren gibt, ist diese klassische Form quasi ausgestorben.

## Mehrprozessorsysteme mit gemeinsamem Speicher



- Jeder Prozess/Thread sieht den gesamten Adressraum des gemeinsamen Speichers
- Leistungssteigerung:  
Dasselbe Programm läuft parallel auf allen Prozessoren; seine Daten sind auf im gemeinsamen Speicher für alle zugreifbar

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

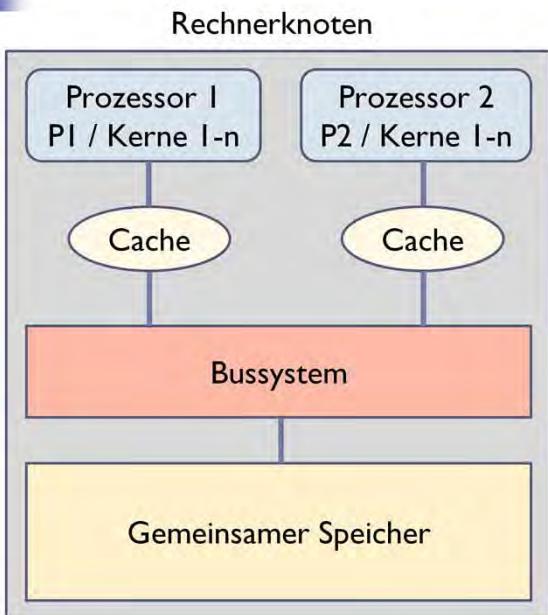
63

Siehe:

- [https://en.wikipedia.org/wiki/Multi-core\\_processor](https://en.wikipedia.org/wiki/Multi-core_processor)

Im uns am besten bekannten Fall handelt es sich hier um einen Rechner mit einem Prozessor und z.B. zwei Prozessorkernen, wie wir ihn heute als normalen PC kaufen. Das Verbindungsnetz ist hier der Prozessor-Speicher-Bus im Rechner. Bei Hochleistungsrechnern finden wir komplexe Formen der Spezialhardware für das Verbindungsnetz.

# Gemeinsamer Speicher und Cache



- In der Realität immer auch mehrstufige Cache-Speicher
- Sehr komplex mit Konsistenz und Kohärenz
- Neue Fragen der Prozessorzuteilung treten auf (Scheduling)

25.10.2016

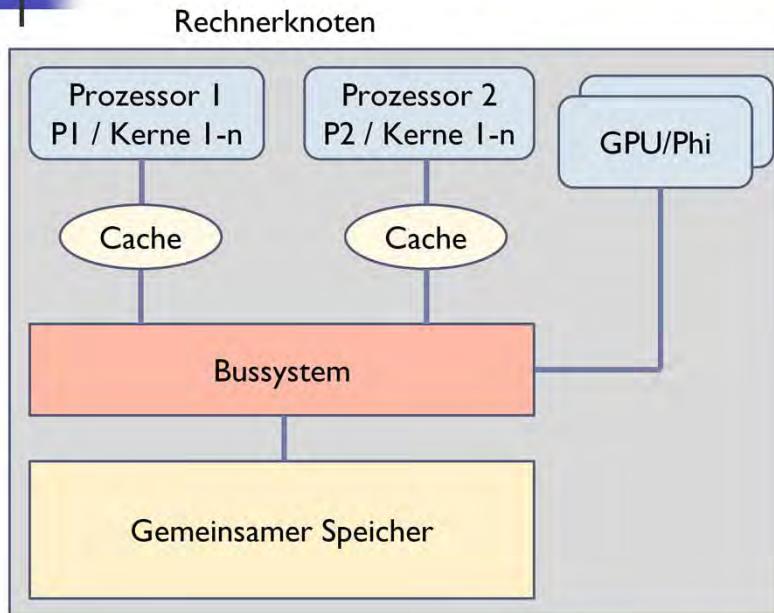
Hochleistungsrechnen - © Thomas Ludwig

64

Siehe:

- [https://en.wikipedia.org/wiki/Cache\\_coherence](https://en.wikipedia.org/wiki/Cache_coherence)

# Gemeinsamer Speicher und Beschleuniger



Weitere Leistungssteigerung (und Stromeinsparung) durch Beschleunigerkarten

- General purpose computing on graphics processing units (GPGPU)
- Intel Xeon Phi

25.10.2016

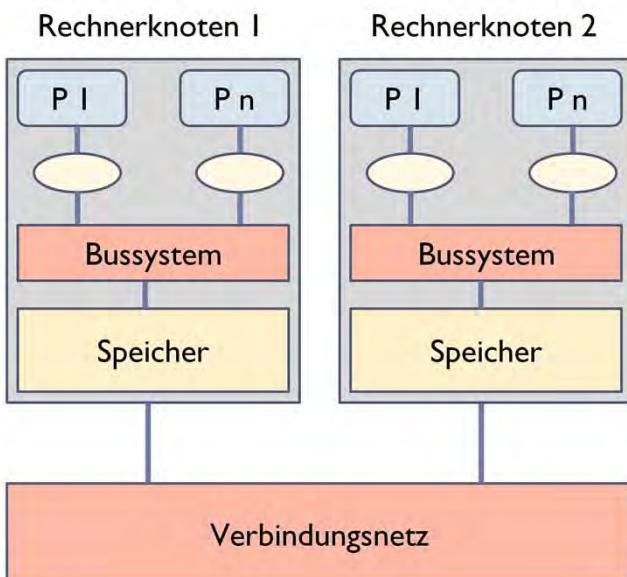
Hochleistungsrechnen - © Thomas Ludwig

65

Siehe:

- [https://en.wikipedia.org/wiki/General-purpose\\_computing\\_on\\_graphics\\_processing\\_units](https://en.wikipedia.org/wiki/General-purpose_computing_on_graphics_processing_units)
- [https://en.wikipedia.org/wiki/Xeon\\_Phi](https://en.wikipedia.org/wiki/Xeon_Phi)

## Mischform gemeinsamer/verteilter Speicher (Standard)



Existierende HLR sind heute meist eine Kombination aus Rechnerknoten mit gemeinsamem Speicher, von den man viele verwendet und sie über ein Verbindungsnetz verbindet

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

66

Die Elemente P1 bis Pn sind Prozessorkerne, die in einem oder mehreren Prozessoren untergebracht sind. Der aktuelle Intel Haswell-Prozessor hat z.B. je nach Typ bis zu 18 Prozessorkerne. Auf einem Rechnerknoten sind typischerweise zwei Prozessoren verbaut, also bis zu 36 Kernen.

## Vor- und Nachteile der Ansätze

- Mehrprozessorsysteme mit verteiltem Speicher
  - Hohe Ausbaubarkeit (>100.000 Prozessoren)
  - Komplexe Programmierung (Nachrichtenaustausch)
  - Reine Variante existiert aber nicht mehr, nur Mischformen mit gemeinsamem Speicher
- Mehrprozessorsysteme mit gemeinsamem Speicher
  - Geringe Ausbaubarkeit (einige dutzend Prozessorkerne und/oder Prozessoren)
  - „Einfachere“ Programmierung (Verwendung gemeinsamer Speicherbereiche)

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

67

Die Programmierung von Systemen mit gemeinsamem Speicher ist nur auf den ersten Blick einfacher. Soll maximale Effizienz erzielt werden, so ist das auch beliebig schwierig. Künftig wird man Kenntnisse der Programmierung dieser Architekturen vermehrt brauchen, wenn nämlich die Mehrkernprozessoren sich weiter verbreiten.

Ausbauen kann man die Systeme natürlich beliebig – die Frage ist, wie lange die Leistungssteigerung den aufgewendeten Finanzen folgt.

## Weitere Bezeichnungen

### Verteilter Speicher

- Multicomputersystem
- Schwache Kopplung
- Lose Kopplung
- Massiv paralleles System
- MPP – massive parallel processing

### Gemeinsamer Speicher

- Multiprozessorsystem
- Multi-core system
- Enge Kopplung
- SMP – symmetric multiprocessing
- Mit Beschleunigern
  - Many-core system

## Abgrenzungen

### Verteilter Speicher

- Rechnerknoten pro HLR:
  - O(100)-O(10.000)
- Kommunikation:
  - Nachrichtenaustausch
- Betriebssysteme:
  - eine Instanz pro Knoten

### Gemeinsamer Speicher

- Prozessorkerne pro Rechnerknoten:
  - O(10)
- Kommunikation:
  - gemeinsame Variable
- Betriebssystem:
  - eine Instanz

## 4. Skalierbarkeit

„Skalierbarkeit“ nirgends eindeutig definiert, aber der wohl am häufigsten benutzte Begriff beim Hochleistungsrechnen

Gemeint ist: Ausbaubarkeit unter Beibehaltung gewisser positiver Charakteristika

- Z.B. Ein Programm skaliert gut, wenn es bei großer Prozesszahl noch hohe Leistung bringt
- Ein Netz skaliert gut, wenn beim Ausbau die Leistung mit dem investierten Geld korreliert

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

70

Siehe:

- <https://en.wikipedia.org/wiki/Scalability>

## **5. Verbindungsnetze und Topologien**

- Im einfachsten Fall
  - Gemeinsamer Speicher: Bussystem
  - Verteilter Speicher: Stern topologie mit Switch
- Im komplexen Fall
  - Alle Varianten, jedoch keine Vollvernetzung

### Probleme

- Latenzzeiten, Übertragungszeiten
- Netzbelastung, Kollisionen

# Beispiele von Verbindungsnetzen

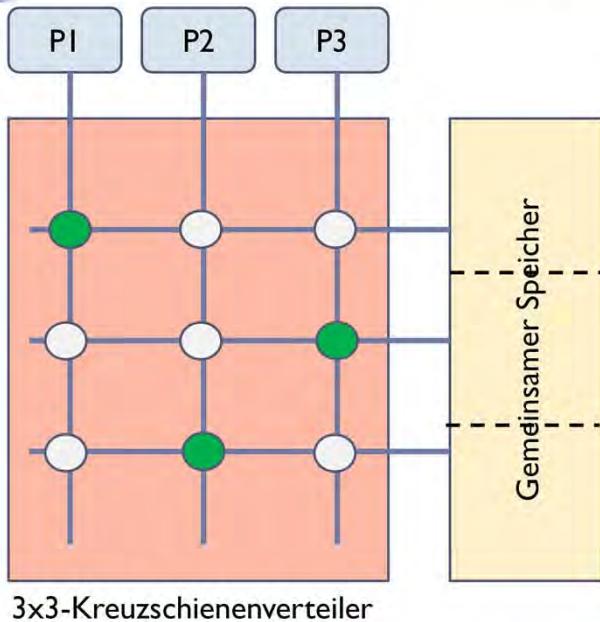
Es gibt hier eine Vielzahl von Konzepten !

- Immer wieder neue Konzepte mit der Werbung  
„das beste je entwickelte Netzwerk“

Wir greifen drei davon zur Illustration heraus:

- Kreuzschienenverteiler
- Zweidimensionaler Torus
- Hypercube

## Verbindungsnetz bei gemeinsamem Speicher



- Kreuzschienenverteiler  
 $n \times m$
- Im günstigsten Fall wie ein m-Bus-System
- Hoher technischer Aufwand
- Reduktion der Konflikte auf dem Bus
- Verwendet zwischen Prozessorkernen im Prozessor

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

73

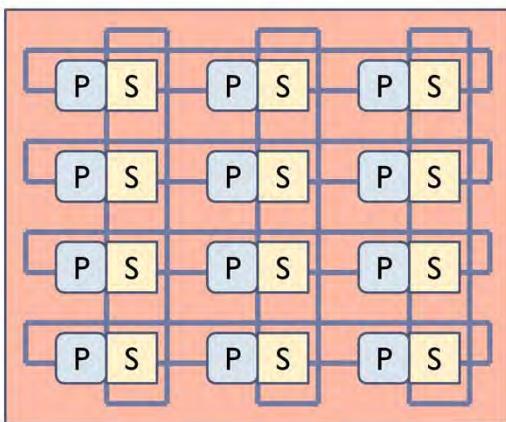
Engl.: crossbar switch

Siehe:

- [https://en.wikipedia.org/wiki/Crossbar\\_switch](https://en.wikipedia.org/wiki/Crossbar_switch)

Verband z.B. drei altmodische Einkern-Prozessoren mit ihren Speichermodulen; die Module (hier drei) können unabhängig voneinander angesprochen werden.

## Verbindungsnetz bei verteiltem Speicher (1)



- Zweidimensionaler Torus/Array
- Konstante Nachbarschaft, deshalb beliebig erweiterbar
- Entfernungsabhängige Übertragungszeiten
- Knotenzahl vervierfacht, maximaler Pfad verdoppelt sich

25.10.2016

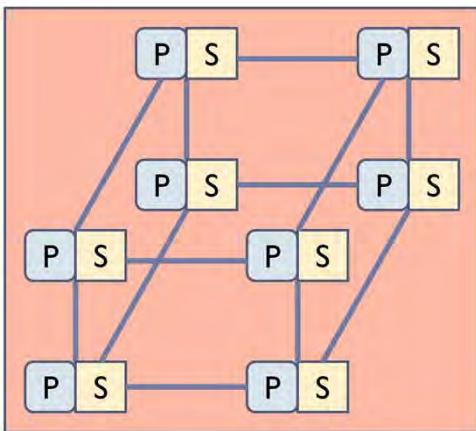
Hochleistungsrechnen - © Thomas Ludwig

74

Siehe:

- [https://en.wikipedia.org/wiki/Torus\\_interconnect](https://en.wikipedia.org/wiki/Torus_interconnect)

## Verbindungsnetz bei verteiltem Speicher (2)



- Hypercube (n-dimensionaler Binärer Würfel)
- #Nachbarn = Dimension
  - Für technische Umsetzung problematisch
- Kurze maximale Entfernungen
- Hoher Grad der Vernetzung
- Knotenzahl vervierfacht, maximaler Pfad wächst um zwei

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

75

Siehe:

- <https://en.wikipedia.org/wiki/Hypercube>

## 6. Hintergrundspeicher

- Lokale Platte an jedem Rechnerknoten
  - Heute meist nur zur Zwischenspeicherung
- Dateiserver ins Netz eingebunden
  - Persistente Datenhaltung
  - Flaschenhals bei Datenzugriff
- Storage Area Network (SAN)
  - Speicherkomponenten mit eigenem Netz an die Komponenten des Clusters angehängt
- Hierarchical Storage Management (HSM) und Bandarchive

Ein-/Ausgabe war bisher vernachlässigte Fragestellung – jetzt intensiver untersucht



## 7. Spezialkonzepte

### Historische Architekturen

- Workstationcluster
- Gridcomputing

### Aktuelle Architekturen

- Cloudcomputing

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

77

# Workstationcluster



- Cluster of workstations (COW)
- Network of workstations (NOW)
- Beowulf cluster (Sterling et al.)
  - Nur Standardkomponenten  
(commodity of the shelf  
components, COTS)  
Intel/AMD, Ethernet, Linux

## Der Arme-Leute-Parallelrechner

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

78

Siehe:

- [https://en.wikipedia.org/wiki/Beowulf\\_cluster](https://en.wikipedia.org/wiki/Beowulf_cluster)

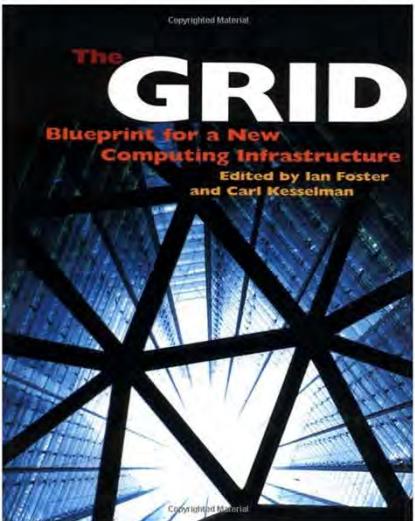
# Helics-Cluster Uni Heidelberg 2001



25.10.2016 Hochleistungsrechnen - © Thomas Ludwig

79

# Gridcomputing



- Jederzeit verfügbare (hohe) Rechenleistung
  - Vergleichbar zu Elektrizität heute
- Netz von Hochleistungsrechnern

Der Superrechner der reichen Leute

Neues Konzept ab ca. 1999, aber:

- kam nie so richtig zum Fliegen trotz vieler Millionen von Forschungsmitteln weltweit

25.10.2016

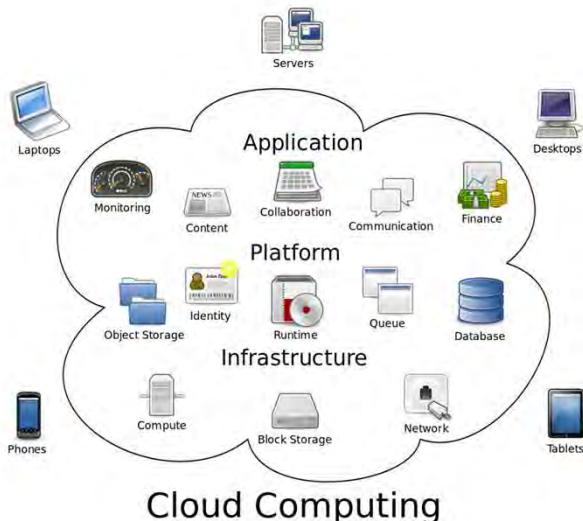
Hochleistungsrechnen - © Thomas Ludwig

80

Siehe:

- [https://en.wikipedia.org/wiki/Grid\\_computing](https://en.wikipedia.org/wiki/Grid_computing)

# Cloudcomputing



- Jederzeit verfügbare (hohe) Leistung zum Rechnen, Speichern, Programmenutzen ...
- Netz von IT-Komponenten

Der Rechner/Speicher für die Zukunft ?

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

81

Siehe:

- [https://en.wikipedia.org/wiki/Cloud\\_computing](https://en.wikipedia.org/wiki/Cloud_computing)

"Cloud computing" by Sam Johnston - Created by Sam Johnston using OmniGroup's OmniGraffle and Inkscape (includes Computer.svg by Sasa Stefanovic) This vector image was created with Inkscape.. Licensed under CC BY-SA 3.0 via Commons -  
[https://commons.wikimedia.org/wiki/File:Cloud\\_computing.svg#/media/File:Cloud\\_computing.svg](https://commons.wikimedia.org/wiki/File:Cloud_computing.svg#/media/File:Cloud_computing.svg)



## **Hardware-Architekturen**

### **Zusammenfassung**

- Leistungssteigerung durch Parallelismus
- Erste wichtige Begriffsbildung durch Flynn
- Wir unterscheiden Architekturen mit verteiltem und mit gemeinsamem Speicher
- Die Skalierbarkeit ist bei verteiltem Speicher sehr hoch, dafür erschwert sich die Programmierbarkeit
- Reale Hochleistungsrechner sind meist viele vernetzte Rechnerknoten mit jeweils gemeinsamem Speicher und mehreren Mehrkernprozessoren
- Verbindungsnetze gibt es mit vielen Topologien
- Speichersysteme nutzen ebenfalls Parallelität

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

82



## **Hardware-Architekturen**

### **Die wichtigsten Fragen**

- Auf welchen Ebenen finden wir Parallelismus?
- Wie unterteilt Flynn die Rechnerarchitekturen?
- Wie funktionieren Systeme mit verteiltem Speicher?
- Wie funktionieren Systeme mit gemeinsamem Speicher?
- Welche Vor- und Nachteile haben die Ansätze?
- Wie sind reale Systeme aufgebaut?
- Welche Aufgabe hat das Verbindungsnetz und wie ist es strukturiert?
- Welche Konzepte finden wir beim Hintergrundspeicher?
- Welche weiteren Architekturen finden wir im Umfeld?

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

83



# **Die TOP500-Liste**

1. Vergleich von Rechnersystemen
2. Die TOP500-Liste
3. Beispielsysteme
4. Historische Sicht

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

84

# 1. Vergleich von Rechnersystemen

## Komplexe Fragestellung

- Erster Ansatz: FLOPS (auch Flop/s)  
floating point operations per second
- Theoretisches Maximum ergibt sich aus der Anzahl der Zyklen pro Gleitkommaoperation

## Bewertung durch sogenannte Benchmark-Programme

- Synthetische Benchmarks (meist Assembler)
- CPU-Benchmark (meist numerische Programme)
- I/O-Benchmark

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

85

Siehe:

- <http://en.wikipedia.org/wiki/Flops>
- [http://en.wikipedia.org/wiki/Benchmark\\_%28computing%29](http://en.wikipedia.org/wiki/Benchmark_%28computing%29)

# Zum Vergleich: Prozessoren

Beispiele der GFLOP-Werte an einigen CPUs<sup>[10]</sup>

LINPACK 1kx1k (DP)	Höchstleistung (in GFLOPS)	Durchschnittsleistung (in GFLOPS)	Effizienz (in %)
Cell, 1 SPU, 3,2 GHz	1,83	1,45	79,23
Cell, 8 SPUs, 3,2 GHz	14,63	9,46	64,66
Pentium 4, 3,2 GHz	6,4	3,1	48,44
Pentium 4 + SSE3, 3,6 GHz	14,4	7,2	50,00
Core i7, 3,2 GHz, 4 Kerne	51,2	33,0 (HT enabled) <sup>[11]</sup>	64,45
Core i7, 3,47 GHz, 6 Kerne	83,2		
Core i7 Sandy-Bridge, 3,4 GHz, 4 Kerne	102,5	92,3	90,05
Itanium, 1,6 GHz	6,4	5,95	92,97

**Xeon E5 v3 Haswell-EP Performance – Linpack** (September 8, 2014 by Donald Kinghorn)

A good approximation of theoretical peak for Haswell looks like this:

CPU GHz \* number of cores \* SIMD vector ops (AVX) \* special instructions effect (FMA3)

For the dual Xeon E5-2687W v3 @ 3.10GHz system theoretical peak would be

$$3.1 * 20 * 8 * 2 = 992 \text{ GFLOPS}$$

**What did I get? 788 GFLOPS approx. 80% of theoretical peak**

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

86

Siehe:

- [http://de.wikipedia.org/wiki/Floating\\_Point\\_Operations\\_Per\\_Second](http://de.wikipedia.org/wiki/Floating_Point_Operations_Per_Second)
- <http://www.pugetsystems.com/blog/2014/09/08/Xeon-E5-v3-Haswell-EP-Performance-Linpack-595/>

## Zum Vergleich: Anwendungen

- Rechnersystem Blizzard am DKRZ 2010
  - 110 TFLOPS LINPACK-Leistung
  - Bei ca. 8.500 Prozessorkernen macht das ca. 13 GFLOPS/Prozessorkern
- Klimaberechnung IPCC AR5
  - Geschätzter Bedarf: 30 Millionen Prozessorkernstunden
    - DKRZ stellt 60 Millionen pro Jahr bereit
  - Ca. 50 TFLOP pro Prozessorkernstunde
    - Entspricht 50.000.000.000.000 FLOP pro Prozessorkernstunde

# Vergleich von Rechnersystemen...

## Der parallele LINPACK-Benchmark

- Entwickelt von Jack Dongarra (Knoxville, TN)
- Ist gleichzeitig eine vollwertige Bibliothek für lineare Algebra
- Benchmark: dicht besetztes Gleichungssystem
- $R_{\max}$  ist maximale Leistung bei Problemgröße  $N_{\max}$
- $R_{\text{peak}}$  ist die theoretische Maximalleistung

Bezeichnet als HPL – High Performance Linpack

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

88

Siehe:

- [http://en.wikipedia.org/wiki/LINPACK\\_benchmarks](http://en.wikipedia.org/wiki/LINPACK_benchmarks)

## Vergleich von Rechnersystemen...

### Kritik

- HPL läuft zu lange
  - Z.B. eine Woche den Rechner dafür benutzen bei 260 Wochen Standzeit des Rechners und 100 M€ Vollkosten kostet somit 380 T€ für den Linpack
- HPL repräsentiert nur wenige parallele Programme
- Manche Rechnern werden auf guten Linpack hin entworfen
- In der Praxis deshalb Anwendungsbenchmarks

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

89

## 2. Die TOP500-Liste

Website [www.top500.org](http://www.top500.org)

- Hans Meuer (†) (Universität Mannheim)
- Jack Dongarra (Univ. Tennessee, Knoxville)
- Erich Strohmeier (NERSC/LBNL)
- Horst Simon (NERSC/LBNL)

Zwei Aktualisierungen pro Jahr

- Juni: International Supercomputing Conference Deutschland
- November: Supercomputing Conference USA

Basiert auf dem LINPACK-Benchmark

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

90

Siehe:

- <http://top500.org/>



RANK	SITE	SYSTEM	CORES	RMAX [TFLOP/S]	RPEAK [TFLOP/S]	POWER (kW)
1	National Super Computer Center in Guangzhou China	Tianhe-2 [MilkyWay-2] - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808
2	DOE/SC/Oak Ridge National Laboratory United States	Titan - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0	27,112.5	8,209
3	DOE/NNSA/LLNL United States	Sequoia - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2	20,132.7	7,890
4	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705,024	10,510.0	11,280.4	12,660
5	DOE/SC/Argonne National Laboratory United States	Mira - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786,432	8,586.6	10,066.3	3,945
6	DOE/NNSA/LANL/SNL United States	Trinity - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Aries interconnect Cray Inc.	301,056	8,100.9	11,078.9	
7	Swiss National Supercomputing Centre [CSCS] Switzerland	Piz Daint - Cray XC30, Xeon E5-2670 8C 2.600GHz, Aries interconnect , NVIDIA K20x Cray Inc.	115,984	6,271.0	7,788.9	2,325
25.10.2016	HLRS - Höchstleistungsrechenzentrum Stuttgart Germany	Hazel Hen - Cray XC40, Xeon E5-2680v3 12C 2.5GHz, Aries interconnect Cray Inc.	185,088	5,640.2	7,403.5	

Nov  
2015

### TOP500 Nov 2014 1-8 / Ein paar wichtige Daten sollten Sie kennen:

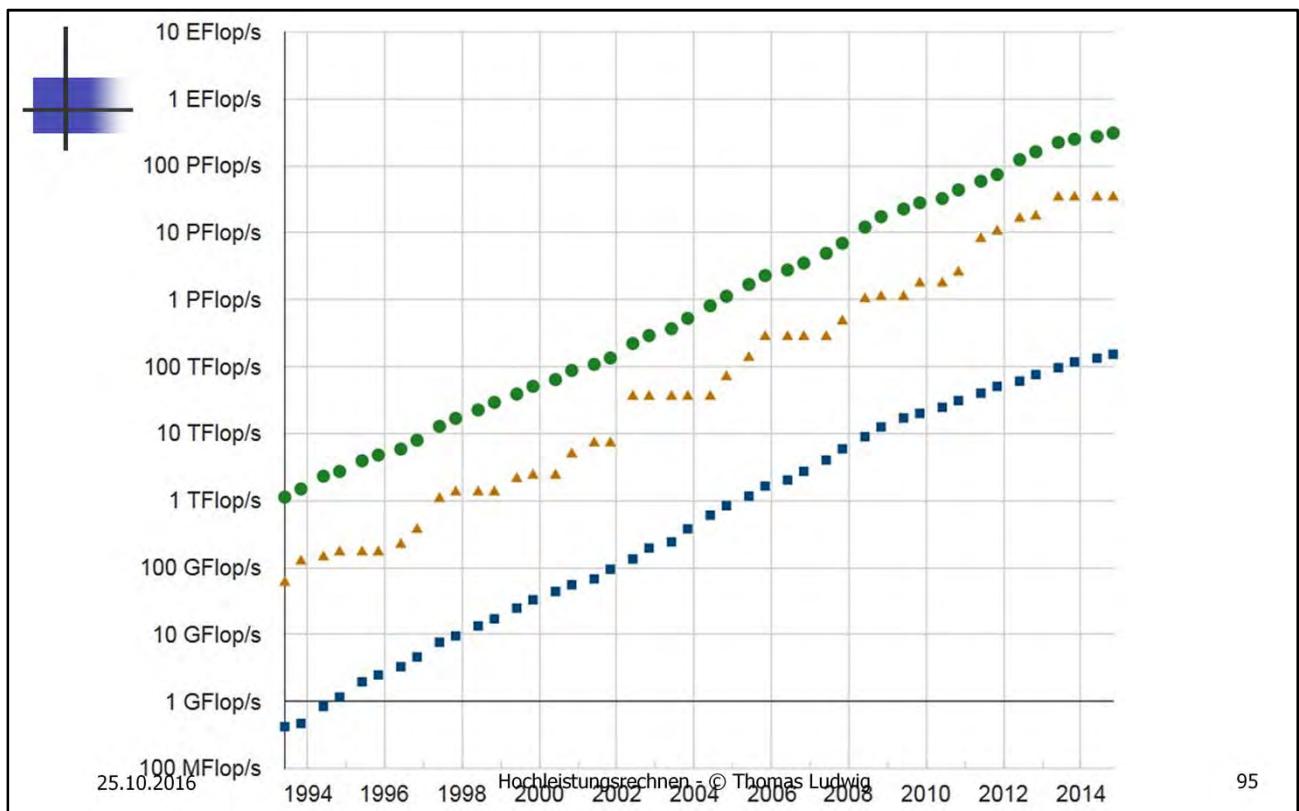
- Nr 1: Mit über 3,1 Millionen Prozessorkernen kommt Tianhe-2 auf über 33 PFLOPS bei ca. 17 MW. Das liegt am verwendeten Intel Xeon Phi Manycore-Prozessor.
- Nr 2: Mit über 560.000 Prozessorkernen kommt Titan auf 17 PFLOPS. Es werden NVIDIA-Grafikkarten verwendet.
- Nr 3: Mit über 1,5 Millionen Prozessorkernen kommt Sequoia auf über 16 PFLOPS bei ca. 8 MW. Das liegt am verwendeten stromsparenden Power-Prozessor.
- Nr 4: Mit über 700.000 Prozessorkernen verwendet K computer einen SUN Sparc-Prozessor. Ca. 10 PFLOPS bekommen wir für ca. 12 MW. Die Maschine verwendet keine Beschleunigerkarten.
- Nr 8: Ein neuer deutscher Höchstleistungsrechner ist gerade in die vorderen Ränge gekommen.

Nov  
2015  
D

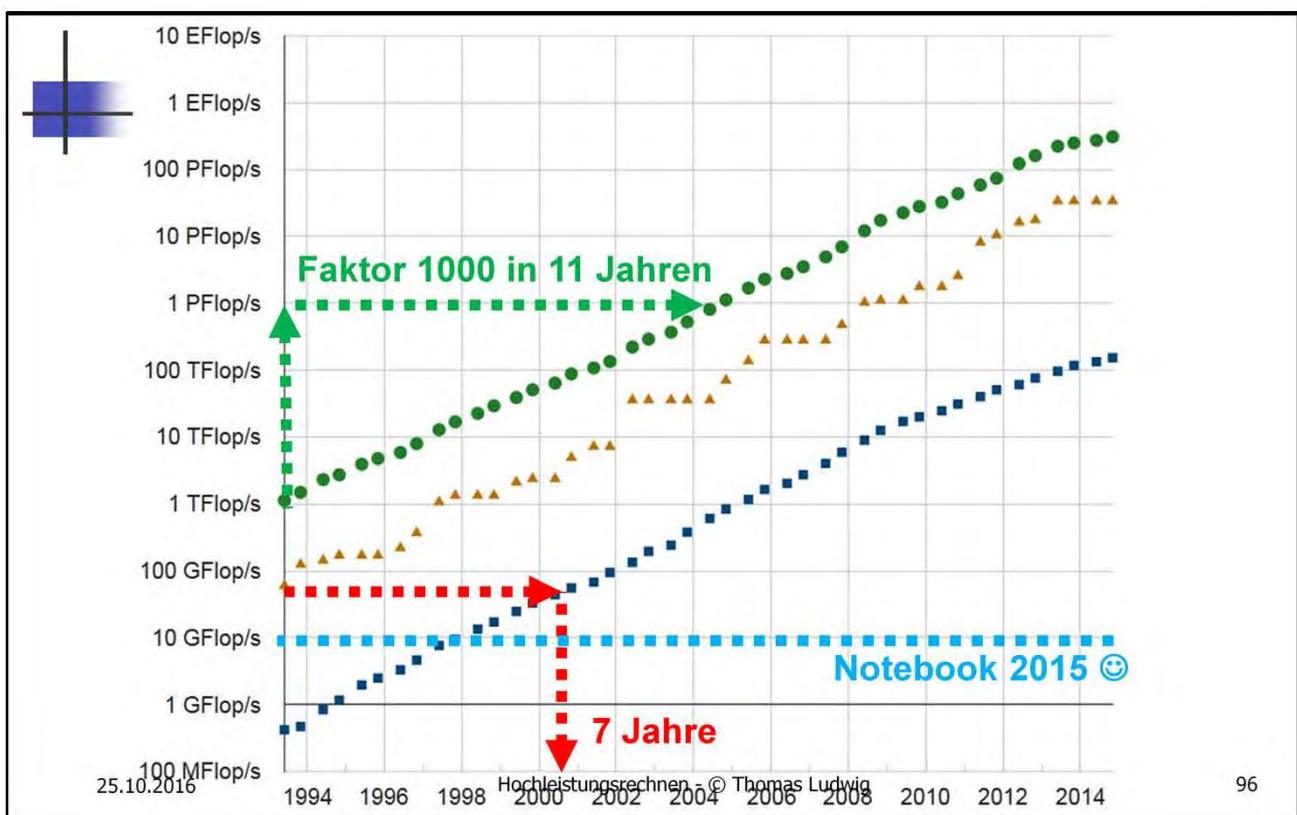
32 Einträge

25.10.2016

	RANK	SITE	SYSTEM		RMAX CORES	RPEAK [TFLOP/S]	POWER [KW]
	8	HLRS – Höchstleistungsrechenzentrum Stuttgart Germany	<b>Hazel Hen</b> - Cray XC40, Xeon E5-2680v3 12C 2.5GHz, Aries interconnect Cray Inc.		185,088	5,640.2	7,403.5
	11	Forschungszentrum Juelich [FZJ] Germany	<b>JUQUEEN</b> - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect IBM		458,752	5,008.9	5,872.0
	23	Leibniz Rechenzentrum Germany	<b>SuperMUC</b> - iDataPlex DX360M4, Xeon E5-2680 8C 2.70GHz, Infiniband FDR IBM/Lenovo		147,456	2,897.0	3,185.1
	24	Leibniz Rechenzentrum Germany	<b>SuperMUC Phase 2</b> - IBM NeXtScale nx360M5, Xeon E5-2697v3 14C 2.6GHz, Infiniband FDR14 Lenovo/IBM		86,016	2,813.6	3,578.3
	50	Forschungszentrum Juelich [FZJ] Germany	<b>JURECA</b> - T-Platforms V-Class, Xeon E5-2680v3 12C 2.5GHz, Infiniband EDR/ParTec ParaStation ClusterSuite, NVIDIA Tesla K80/K40 T-Platforms		49,476	1,424.7	1,693.4
	56	Max-Planck-Gesellschaft MPI/IPP Germany	iDataPlex DX360M4, Intel Xeon E5-2680v2 10C 2.800GHz, Infiniband FDR Lenovo/IBM		65,320	1,283.3	1,463.2
	64	DKRZ - Deutsches Klimarechenzentrum Germany	<b>Mistral</b> - bullx DLC 720, Xeon E5-2680v3 12C 2.5GHz, Infiniband FDR Bull, Atos Group		37,344	1,139.2	1,493.8
	77	TU Dresden, ZIH Germany	<b>Taurus</b> - bullx DLC 720, Xeon E5-2680v3 12C 2.5GHz, Infiniband FDR Bull, Atos Group		34,656	1,029.9	1,386.2



Grün: aggregiert alle 500 Systeme – braun: Systeme auf Rang 1 – blau: Systeme auf Rang 500





<http://de.slideshare.net/top500/top500-list-november-2014>

# Highlights of the 44<sup>th</sup> TOP500 List

SC14,  
New Orleans,  
November 17,  
2014

Erich  
Strohmaier

25.10.2016

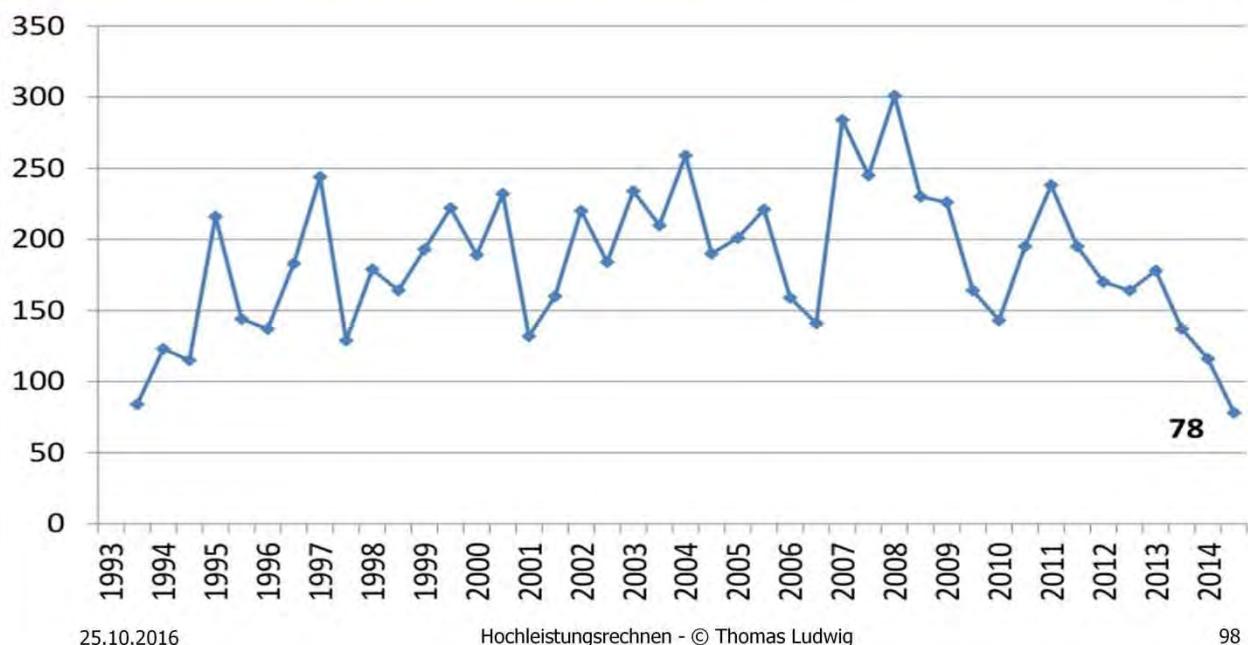
Hochleistungsrechnen - © Thomas Ludwig

97

Siehe:

- <http://de.slideshare.net/top500/top500-list-november-2014> .

## REPLACEMENT RATE

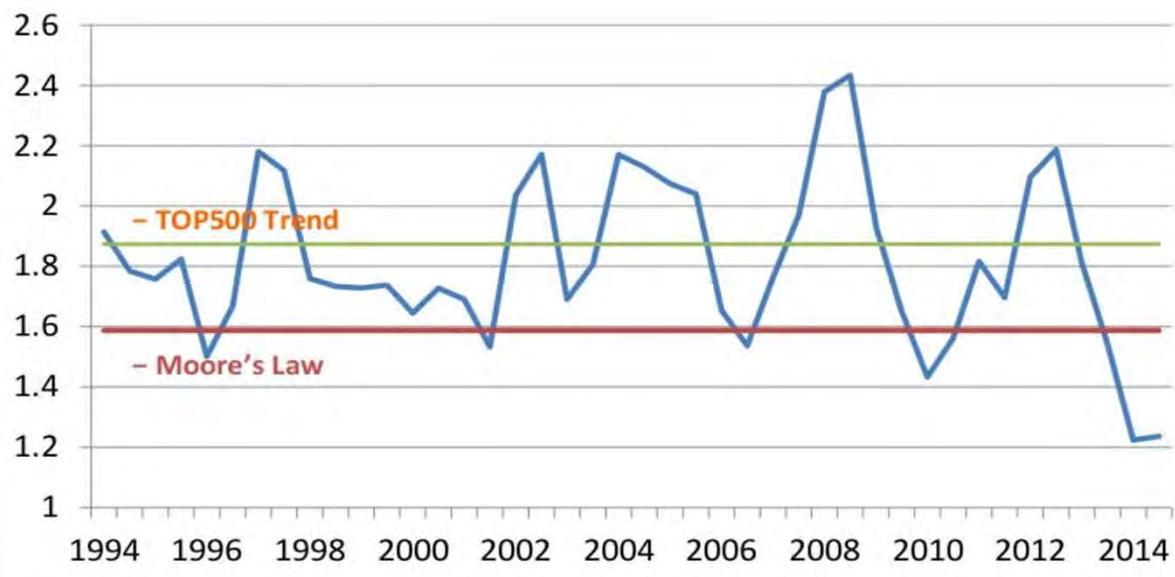


25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

98

# ANNUAL PERFORMANCE INCREASE OF THE TOP500

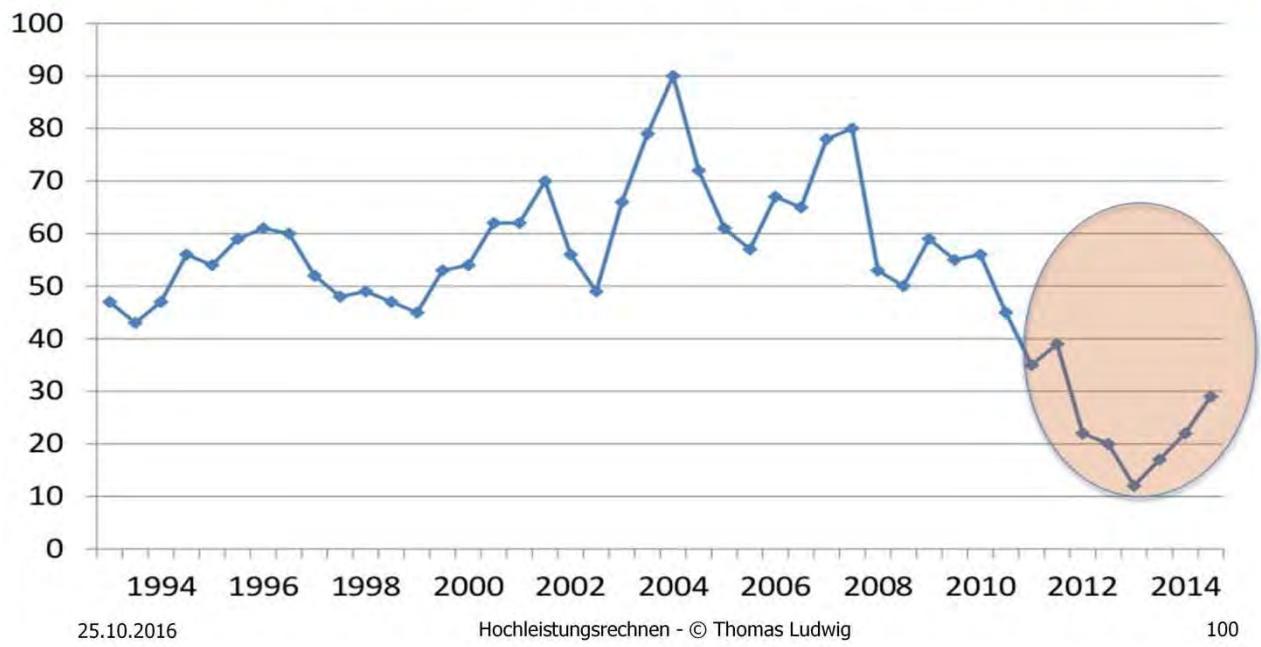


25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

99

# RANK AT WHICH HALF OF TOTAL PERFORMANCE IS ACCUMULATED

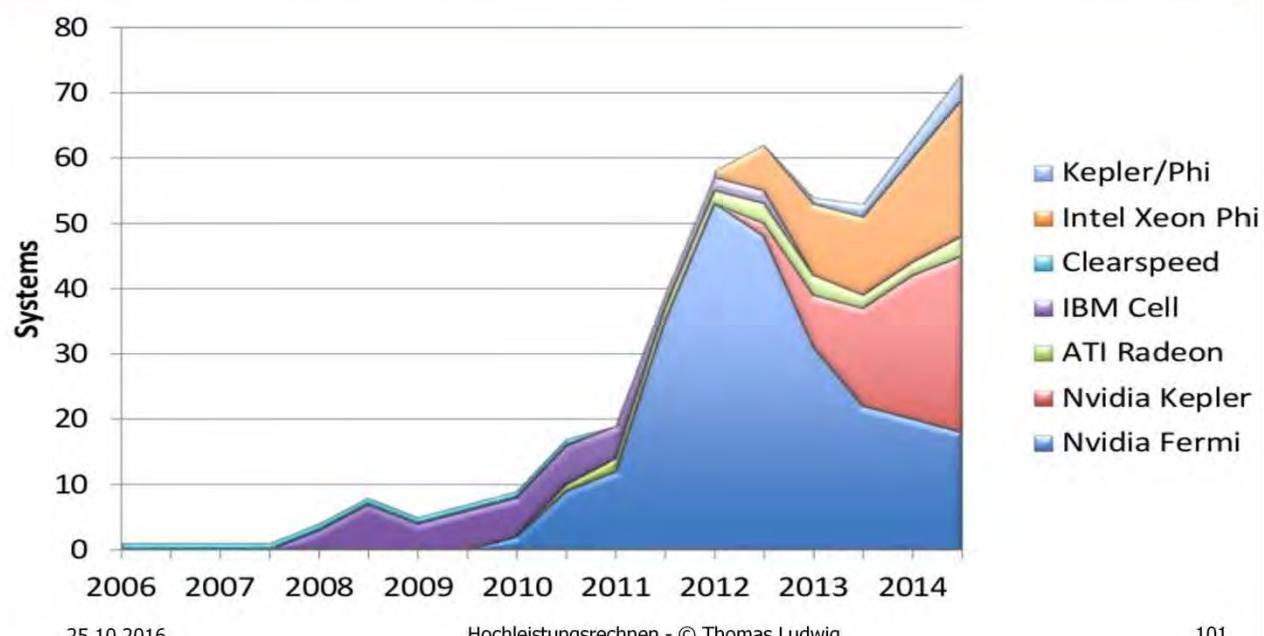


25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

100

## ACCELERATORS

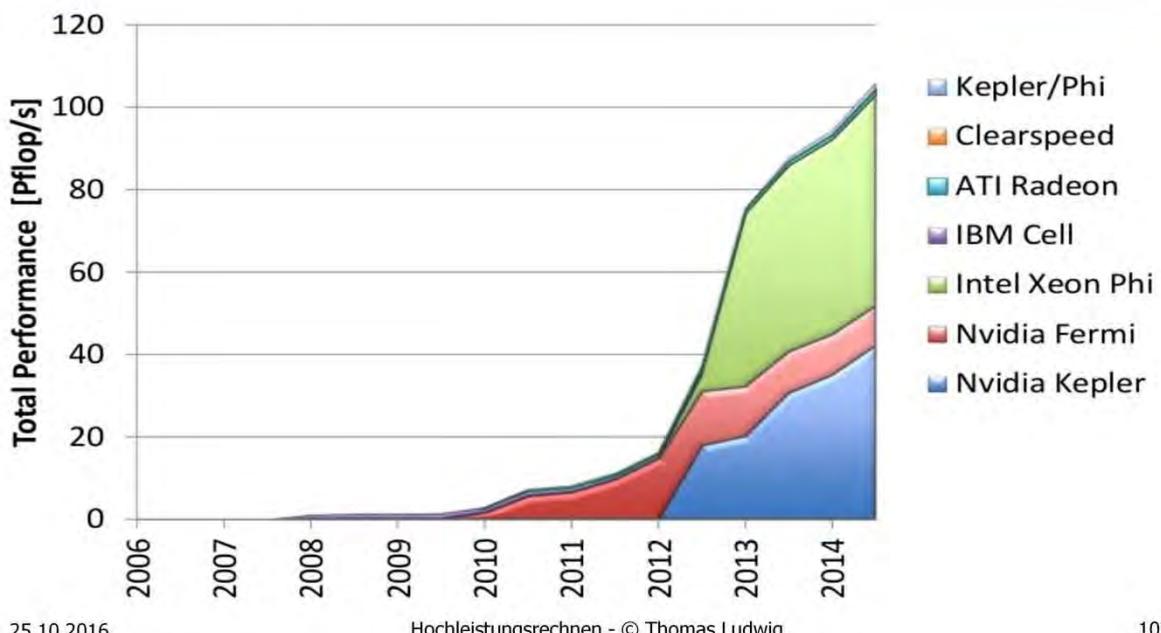


25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

101

# PERFORMANCE OF ACCELERATORS

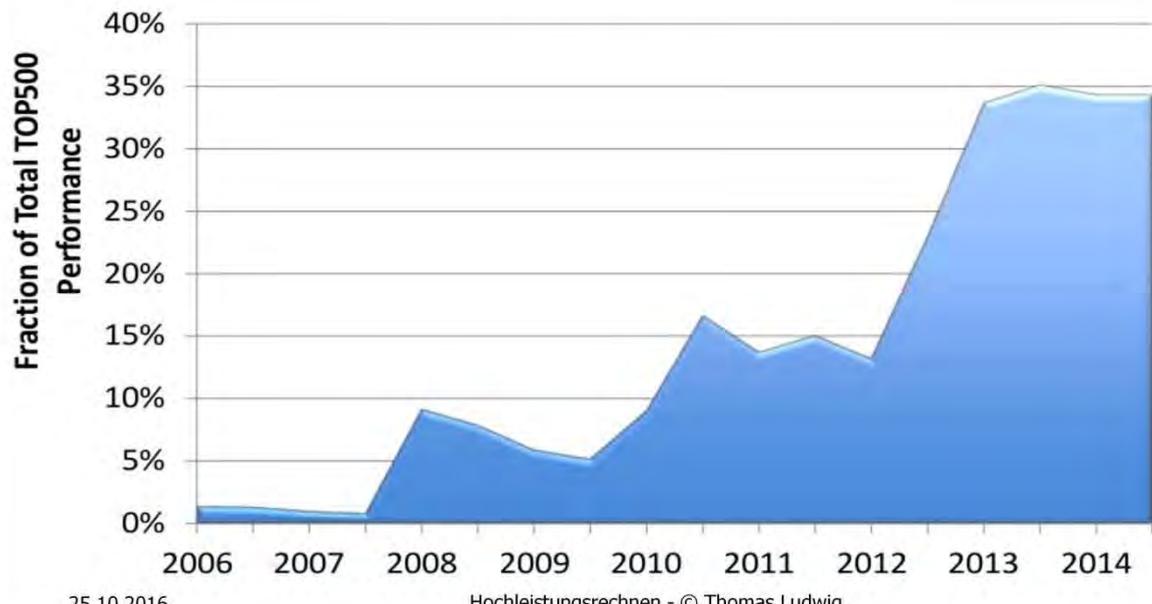


25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

102

# PERFORMANCE SHARE OF ACCELERATORS

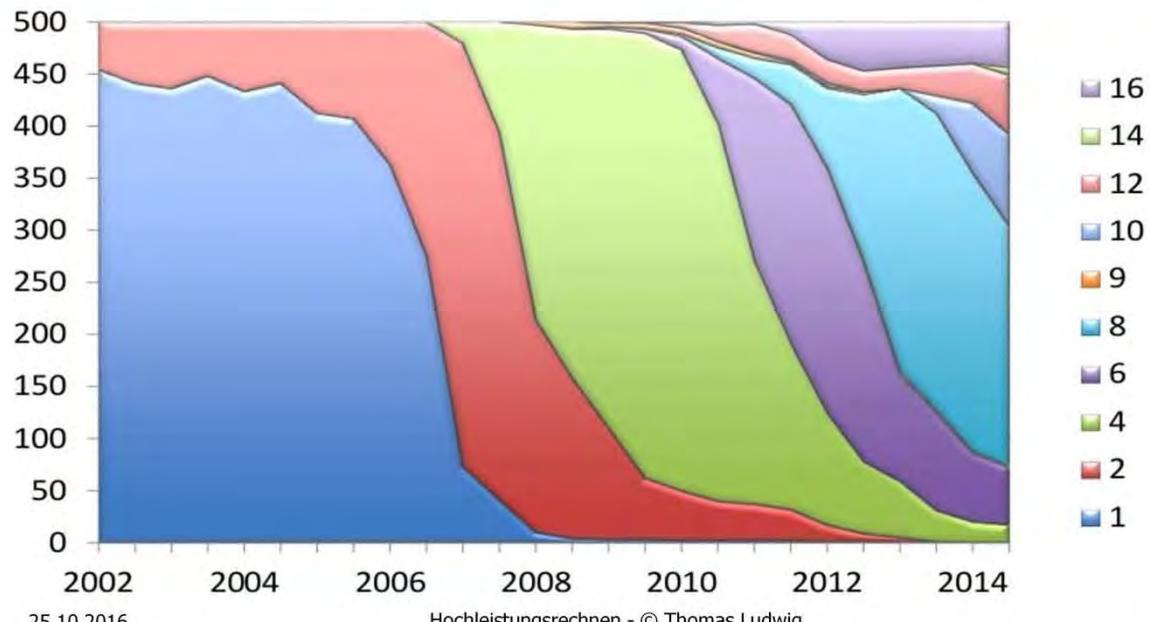


25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

103

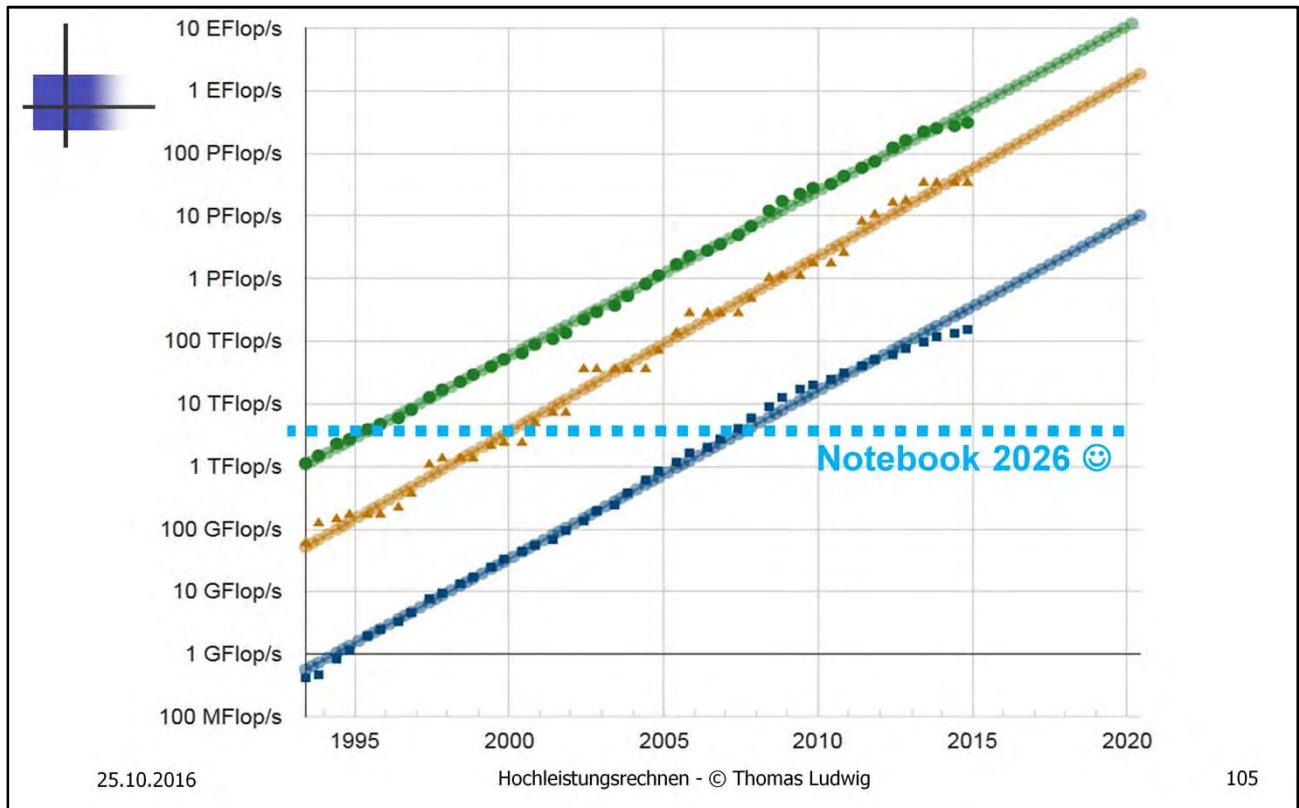
## CORES PER SOCKET



25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

104



Grün: aggregiert alle 500 Systeme – braun: Systeme auf Rang 1 – blau: Systeme auf Rang 500

## Leistungsentwicklung bis November 2014

Moore's Law

„Verdopplung der Transistorzahl alle 18 Monate“  
(entspricht evtl. Leistungsverdopplung)

Jun93–Nov14: 21,5 Jahre = 14,3x18 Monate  
etwa Faktor  $2^{14}=16384$

Leistung Summe: 1 TFlop/s – 309.000 TFlop/s (x 309k)

Leistung #1: 60 GFlop/s – 34.000 TFlop/s (x 570k)

Leistung #500: 0,4 GFlop/s – 153.000 GFlop/s (x 382k)



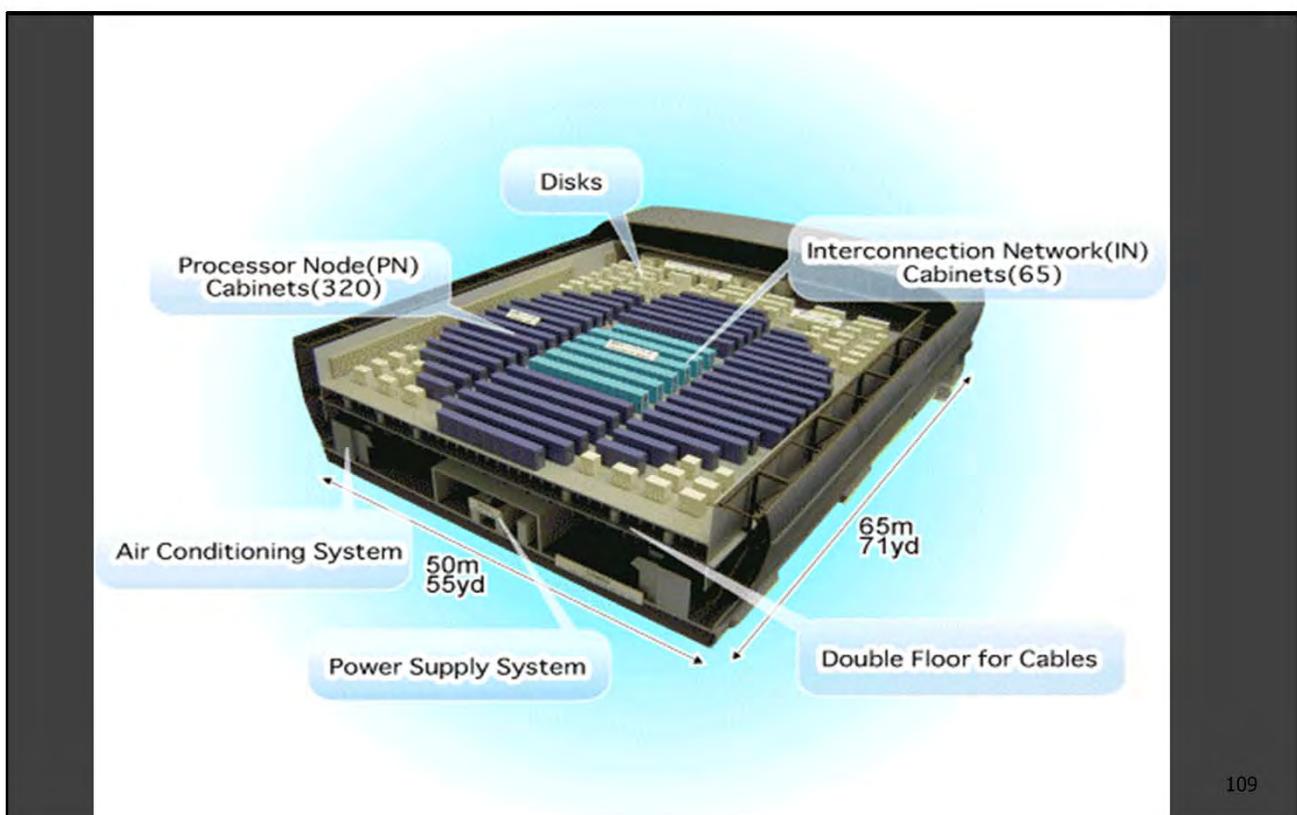
## 3. Beispielsysteme

- NECs Earth Simulator (Yokohama, Japan)
- Fujitsu K Computer (Kobe, Japan)
- Mare Nostrum (Barcelona, Spanien)

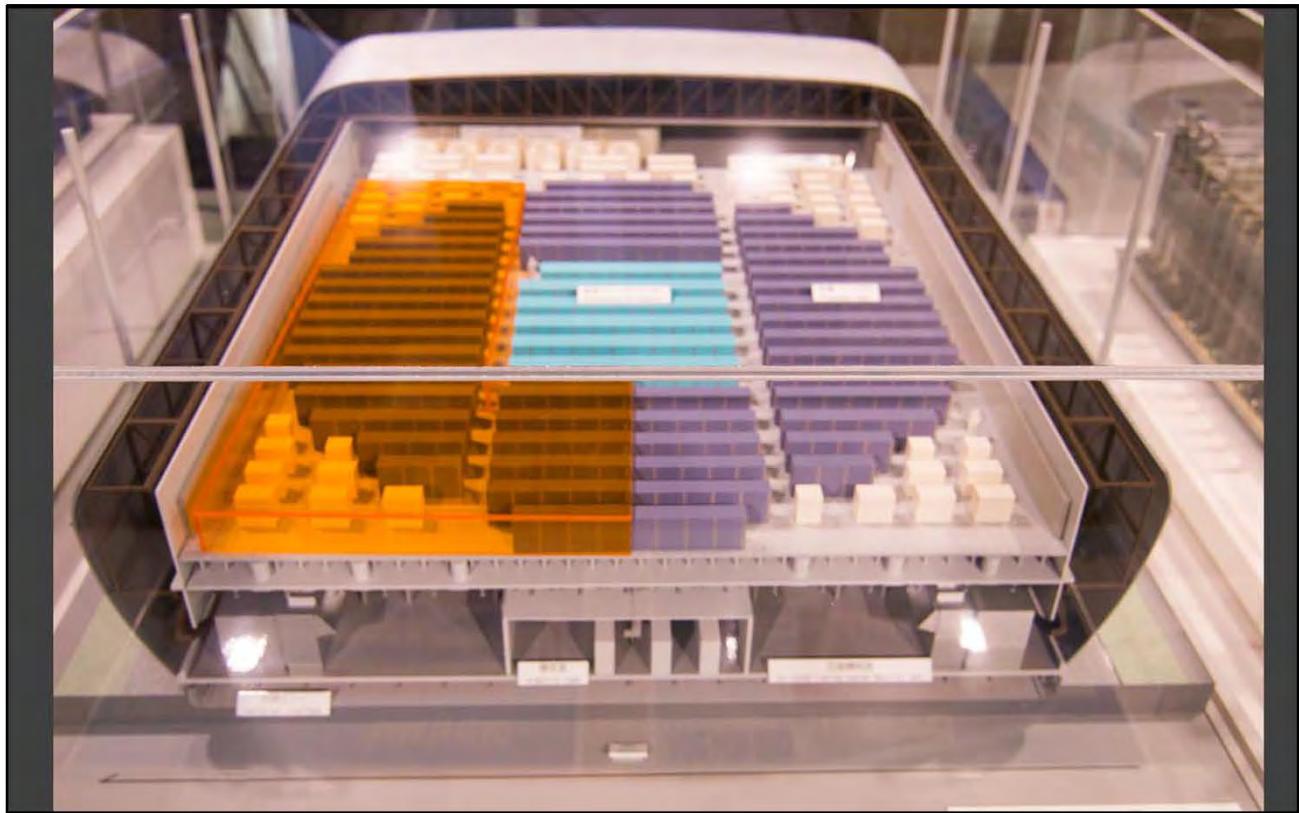
## NECs Earth Simulator (6/2002-11/2008)

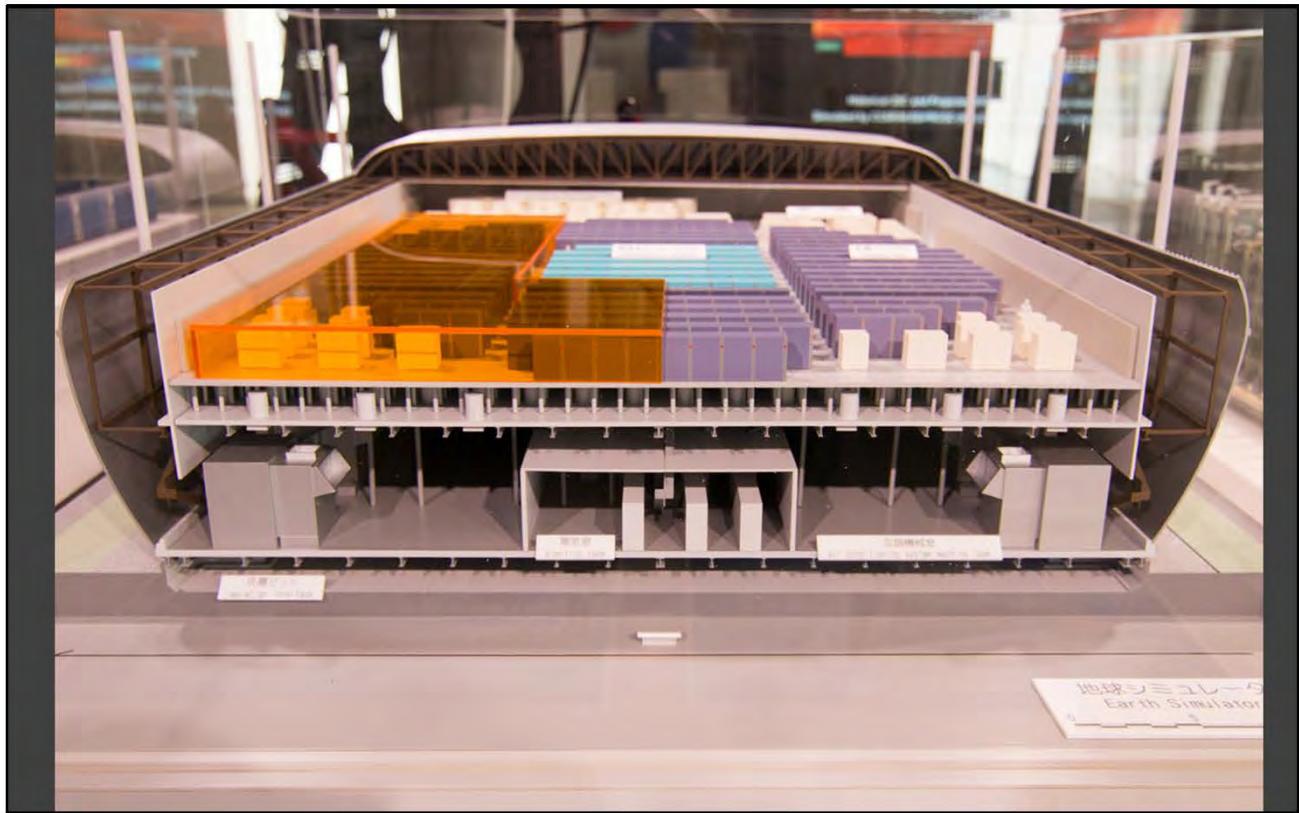
- 640 Knoten
- Zu je 8 Vektorprozess.
- 5120 Prozessoren
- 0,15 mikron Kupfer
- 200 MioUSD Rechner
- 200 MioUSD Gebäude und Kraftwerk
- Zum Zwecke der Klimaforschung etc.
- ▶ 36 TFLOPS
- ▶ 10 TByte Hauptspeicher
- ▶ 700 TByte Festplatten
- ▶ 1,6 PByte Bandspeicher
- ▶ 83.000 Kupferkabel
- ▶ 2.800 km/220 t Kabel
- ▶ 3250qm
- ▶ Erdbebensicher

**Computenic-Shock**



109





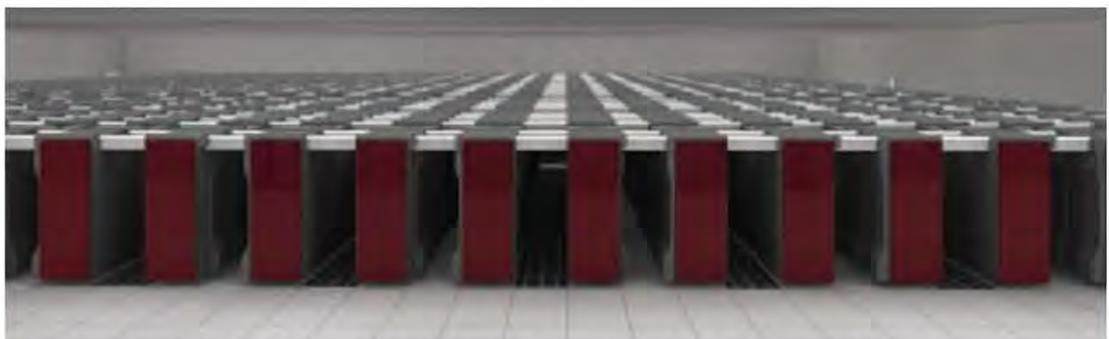






## Fujitsu K Computer

- RIKEN Advanced Institute for Computational Science (AICS), Japan



- 68.544 SPARC64 VIIIfx CPUs (2,0 GHz) mit je 8 Prozessorkernen in 672 Schränken
- November 2012: 864 Schränke und 10 PFLOPS

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

115









MareNostrum, Barcelona (11/2004)

# Gewinner des Schönheitswettbewerbs

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

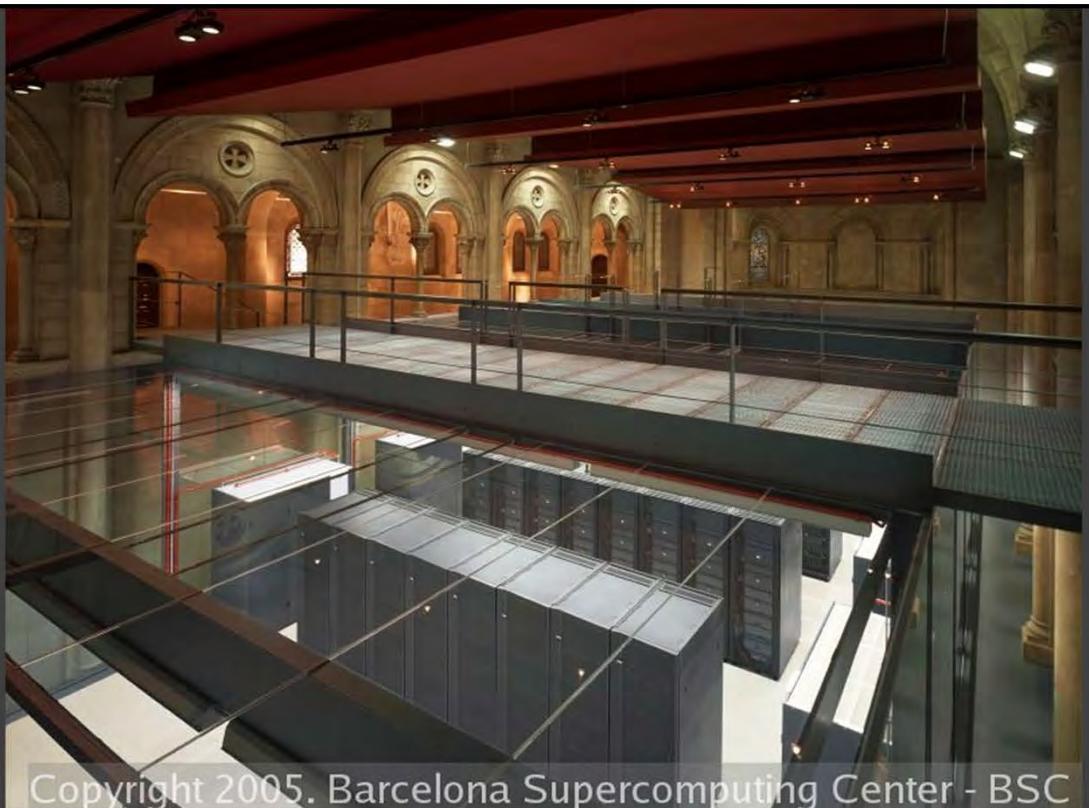
119



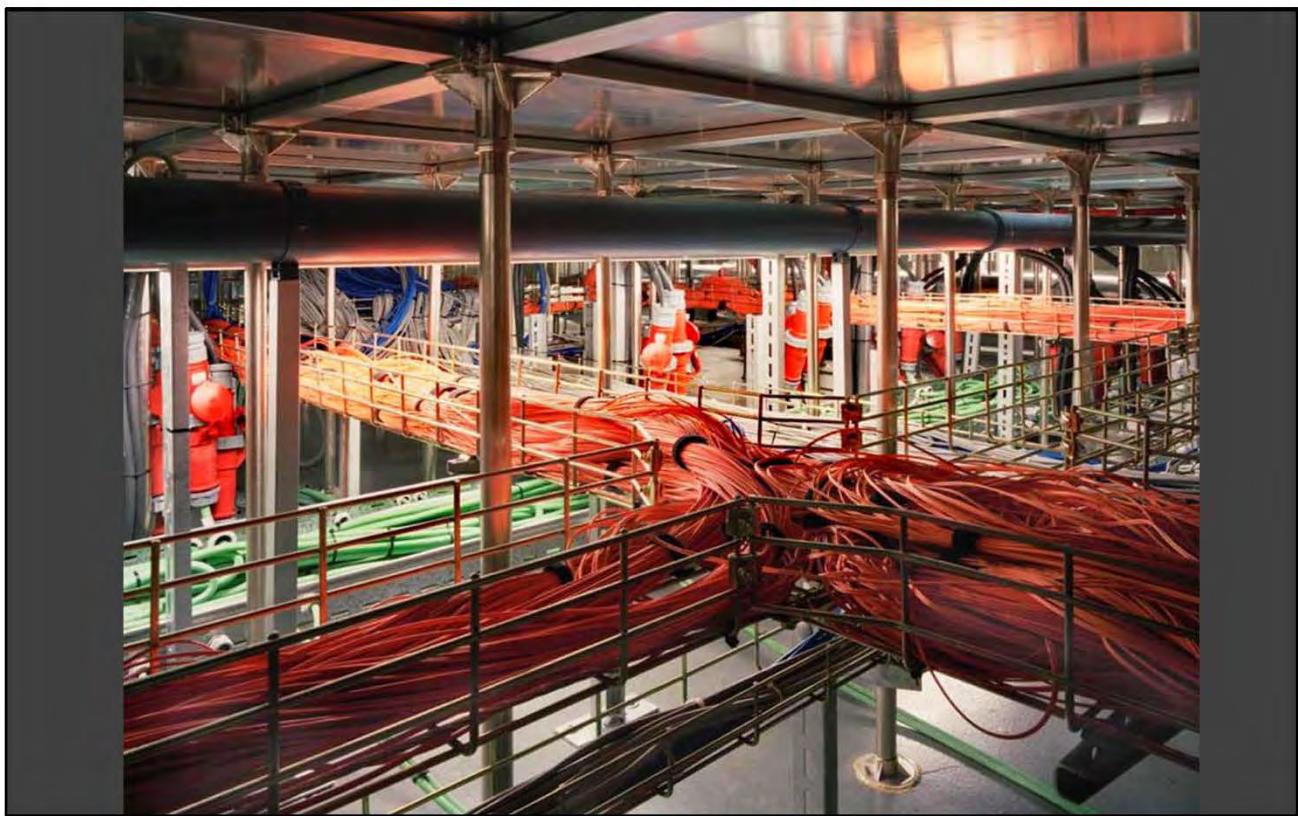
Copyright 2006, Barcelona Supercomputing Center - BSC



Copyright 2005. Barcelona Supercomputing Center - BSC



Copyright 2005. Barcelona Supercomputing Center - BSC





## 4. Historische Sicht

- Die TOP500 im Juni 1993
- Deutschland in der TOP500 im Juni 1993

Jun  
1993

	Rank	Manufacturer Computer/Procs	R <sub>max</sub> R <sub>peak</sub>	Installation Site Country/Year	Inst. type Installation Area	Nmax Nhalf	Computer Family Computer Type
	1	TMC CM-5/1024/ 1024	<b>59.70</b> 131.00	Los Alamos National Laboratory USA/	Research Energy	52224 24064	TMC CMS CM5
	2	TMC CM-5/1024/ 1024	<b>59.70</b> 131.00	National Security Agency USA/	Classified	52224 24064	TMC CMS CM5
	3	TMC CM-5/544/ 544	<b>30.40</b> 70.00	Minnesota Supercomputer Center USA/	Industry	36864 16384	TMC CMS CM5
	4	TMC CM-5/512/ 512	<b>30.40</b> 66.00	NCSA USA/	Academic	36864 16384	TMC CMS CM5
	5	NEC SX-3/44R/ 4	<b>23.20</b> 26.00	NEC Fuchu Plant Japan/1990	Vendor	6400 830	NEC Vector SX3
	6	NEC SX-3/44/ 4	<b>20.00</b> 22.00	Atmospheric Environment Service (AES) Canada/1991	Research Weather	6144 832	NEC Vector SX3
	7	TMC CM-5/256/ 256	<b>15.10</b> 33.00	Naval Research Laboratory (NRL) USA/1992	Research	26112 12032	TMC CMS CM5
	8	Intel Delta/ 512	<b>13.90</b> 20.48	Caltech USA/	Academic	25000 7500	intel Paragon Paragon
	9	Cray/SGI Y-MP C916/16256/ 16	<b>13.70</b> 15.24	Cray Research USA/	Vendor	10000 650	Cray Vector C90
	10	Cray/SGI Y-MP C916/16256/ 16	<b>13.70</b> 15.24	DOE/Bettis Atomic Power Laboratory USA/1993	Research	10000 650	Cray Vector C90
	11	Cray/SGI Y-MP C916/16256/ 16	<b>13.70</b> 15.24	DOE/Knolls Atomic Power Laboratory USA/1993	Research	10000 650	Cray Vector C90
	12	Cray/SGI Y-MP C916/16128/ 16	<b>13.70</b> 15.24	ECMWF UK/1993	Research Weather	10000 650	Cray Vector C90
25.10.201	13	Cray/SGI Y-MP C916/161024/ 16	<b>13.70</b> 15.24	Government USA/1992	Classified	10000 650	Cray Vector C90
	14	Cray/SGI Y-MP C916/161024/ 16	<b>13.70</b> 15.24	Government USA/1992	Classified	10000 650	Cray Vector C90

Rmax-Angabe in GFLOPS

Jun  
1993  
D

25.10.201

	Rank	Manufacturer Computer/Procs	R <sub>max</sub> R <sub>peak</sub>	Installation Site Country/Year	Inst. type Installation Area	Nmax Nhalf	Computer Family Computer Type
	56	Fujitsu S600/20/ 1	4.01 5.00	Universitaet Aachen Germany/1991	Academic		Fujitsu VP VP2000
	57	Fujitsu S600/20/ 1	4.01 5.00	Universitaet Karlsruhe Germany/1990	Academic		Fujitsu VP VP2000
	60	TMC CM-5/64/ 64	3.80 8.19	GMD Germany/1993	Research	13056 6016	TMC CM5 CM5
	65	Fujitsu S400/40/ 2	3.62 5.00	Universitaet Darmstadt Germany/1991	Academic		Fujitsu VP VP2000
	66	Fujitsu S400/40/ 2	3.62 5.00	Universitaet Hannover / RRZN Germany/1991	Academic		Fujitsu VP VP2000
	76	TMC CM-2/32k/ 1024	2.60 7.00	AMK Germany/1990	Classified		TMC CM2 CM2
	98	Cray/SGI Y-MP8/832/ 8	2.14 2.67	Forschungszentrum Juelich (FZJ) Germany/1989	Research		Cray Vector YMP
	102	Cray/SGI Y-MP8/864/ 8	2.14 2.67	Leibniz Rechenzentrum Germany/1992	Academic		Cray Vector YMP
	142	TMC CM-5/32/ 32	1.90 4.10	Universitaet Wuppertal Germany/1992	Academic	9216 4096	TMC CM5 CM5
	149	Intel XP/S5/ 66	1.90 3.30	Forschungszentrum Juelich (FZJ) Germany/1992	Research		intel Paragon Paragon
	155	Intel XP/S5-32/ 66	1.90 3.30	Universitaet Stuttgart Germany/1992	Academic		intel Paragon Paragon
	190	Cray/SGI CRAY-2s/4-128/ 4	1.41 1.95	DKRZ Germany/1988	Research Weather		Cray 2/3 Cray 2
	206	Cray/SGI CRAY-2/4-256/ 4	1.41 1.95	Universitaet Stuttgart Germany/1986	Academic		Cray 2/3 Cray 2
	218	TMC CM-2/16k/ 512	1.30 3.50	GMD Germany/1990	Research		TMC CM2 CM2
	223	NEC SX-3/11/ 1	1.30 1.37	Universitaet Koeln Germany/1990	Academic	2816 192	NEC Vector SX3



## **Die TOP500-Liste**

### **Zusammenfassung**

- Die Rechnerleistung wird mit einem numerischen Benchmark-Programm (LINPACK) evaluiert
- Die TOP500-Liste verzeichnet halbjährig die schnellsten Rechner weltweit
- Die schnellsten Rechner haben die Petaflops-Grenze durchbrochen
- Wir erwarten für ca. 2020 den ersten Exaflops-Rechner
- Aktuelles Problem: Energiebedarf

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

128



## **Die TOP500-Liste**

### **Die wichtigsten Fragen**

- In welcher Maßeinheit wird die Rechnerleistung angegeben?
- Wie wird die Leistung evaluiert?
- Welche Zielsetzung verfolgt das TOP500-Projekt?
- In welchen Größenordnung der Rechnerleistung und des Stromverbrauchs liegen die größten Systeme?
- Wie verhält sich die beobachtete Leistungssteigerung zu Moore's Law?
- Welche Leistung brachten Systeme 1993?

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

129

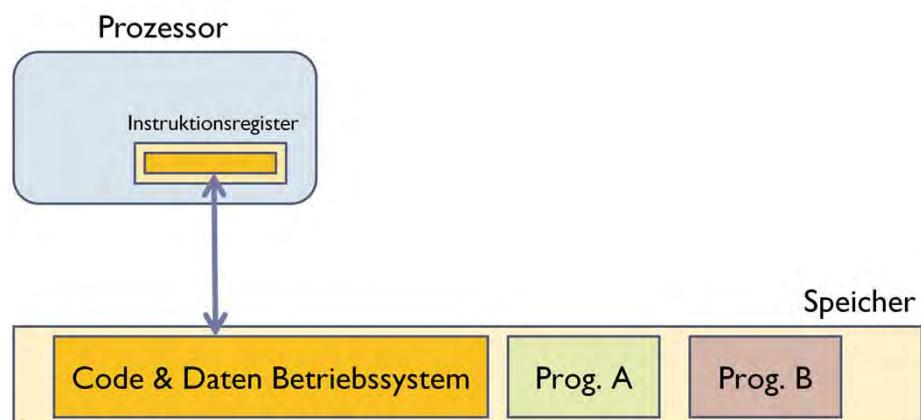


# Betriebssystemaspekte

1. Einprozessor-Einkern-Systeme (†)
2. Mehrprozessor/Mehrkern-Systeme
3. SMP-Hardware
4. SMP-Betriebssystem
5. Synchronisationsmechanismen
6. Erweiterte Betriebssystemfunktionalität

# 1. Einprozessor-Einkern-Systeme

Gibt es seit ca. 2005 nicht mehr in Rechnern



Der Cache wird hier immer vernachlässigt, da er zunächst nur auf der Ebene der Prozessorhardware eine Rolle spielt.

# Erinnerung zu Betriebssystemen

- (Ausführbares) Programm
  - Binärcode, der auf Platte steht und geladen wird
- Prozess
  - Objekt des Betriebssystems zur Verwaltung eines Programms in der Ausführung
  - Prozesse haben geschützte Adressräume – kein anderer Prozess hat Zugriff
  - Ressourcen wie z.B. Adressräume, Dateien, Unterbrechungen usw. werden prozessbezogen verwaltet
- Thread (in einem Prozess)
  - So eine Art Unterprozess im Prozess
  - Threads teilen sich den Adressraum eines Prozesses und können darin allen möglichen Unsinn anstellen
- Betriebssystem
  - Bodensatz an Code, der alles Obige organisiert und am Laufen hält
  - NICHT als Prozesse oder Threads organisiert ! (zumindest nicht in Linux)

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

134

Siehe:

- [https://en.wikipedia.org/wiki/Computer\\_program](https://en.wikipedia.org/wiki/Computer_program)
- [https://en.wikipedia.org/wiki/Process\\_%28computing%29](https://en.wikipedia.org/wiki/Process_%28computing%29)
- [https://en.wikipedia.org/wiki/Thread\\_%28computing%29](https://en.wikipedia.org/wiki/Thread_%28computing%29)

# Moduswechsel

## Der Prozessor arbeitet

- **entweder** im Betriebssystemmodus
  - und bearbeitet Code des Betriebssystems
- **oder** im Benutzermodus
  - und bearbeitet Code des Benutzerprozesses

## Wechsel des Modus

- Systemaufruf im Programm (**synchron**)
- Unterbrechung (**asynchron**)

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

135

Siehe:

- <https://en.wikipedia.org/wiki/Usermode>
- [https://en.wikipedia.org/wiki/System\\_call](https://en.wikipedia.org/wiki/System_call)
- <https://en.wikipedia.org/wiki/Interrupt>

Bei modernen Betriebssystemen ist die Wahrheit ein bisschen komplizierter, insbesondere, welche Adressen der Prozessor im Betriebssystemmodus bearbeitet. Der Code des Betriebssystem wird hier in den Adressraum des Anwendungsprozess eingeblendet und da im Systemmodus abgearbeitet. Davon wollen wir hier abstrahieren.

Moduswechsel sind teuer und sollten vermieden werden, wenn es geht. Dummerweise entsteht bei jeder E/A-Operation ein Moduswechsel.

## Synchronisation im traditionellen Unix-Kern

Feststellung: der UNIX-Kern ist wiedereintrittsfähig (reentrant)

- Mehrere Prozesse arbeiten im Kern zur selben Zeit und evtl. im selben Code-Bereich
- Natürlich nicht echt gleichzeitig sondern verschränkt !
  - Es gibt ja nur einen Prozessor

Frage: Könnte es zu inkonsistenten Daten kommen?

Wenn ja, wie vermeidet man es?

- Die Situation könnte auftreten, wenn zwei Prozesse dieselben Daten, z.B. Puffer, benutzen
- Natürlich darf Dateninkonsistenz nicht auftreten

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

136

Siehe:

- [https://en.wikipedia.org/wiki/Reentrant\\_%28subroutine%29](https://en.wikipedia.org/wiki/Reentrant_%28subroutine%29)

Man beachte: Bibliotheken werden natürlich nur einmal in den Hauptspeicher geladen, möglicherweise aber von mehreren Prozessen verwendet. Auch hier muss der Code wiedereintrittsfähig (reentrant) sein.

## Beispiel einer Problemsituation (1)

- Prozess A will etwas von Datei 1 lesen
- A ruft read() und geht in Systemmodus
- BS sendet Befehl an Platte und legt A schlafen
- Neuer lauffähiger Prozess wird ausgesucht: B
- Prozess B will etwas von Datei 2 lesen
- B ruft read() und geht in Systemmodus
- BS sendet Befehl an Platte und legt B schlafen
- Neuer lauffähiger Prozess wird ausgesucht: C
- C läuft
- Platte erzeugt Unterbrechung, weil Daten von 1 vorliegen
- C wird unterbrochen und geht in Systemmodus
- BS nimmt Daten entgegen, schreibt sie in Puffer 1
- Neuer lauffähiger Prozess wird ausgesucht: noch immer C

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

137

So klingt das Ganze sehr unverdächtig. Ist es an dieser Stelle zunächst auch noch.



## Synchronisation im traditionellen Unix-Kern...

### Vermeidung von Inkonsistenzen

- Das Betriebssystem ist zunächst einmal nicht unterbrechbar (non-preemptive)
- D.h. eine BS-Aktivität wird zu Ende geführt, auch wenn dadurch die Zeitscheibe des zugehörigen Prozesses überschritten wird
- Nach dem Ende sind alle Datenstrukturen konsistent und ein anderer Prozess kann unbedenklich damit arbeiten

Aber ...

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

138

Siehe:

- [http://en.wikipedia.org/wiki/Preemption\\_%28computing%29](http://en.wikipedia.org/wiki/Preemption_%28computing%29)

## Synchronisation im traditionellen UNIX-Kern...

### Problem: Unterbrechungen

- Während der Aktivität im BS-Kern könnte eine Unterbrechung erfolgen
- Die Unterbrechungsbehandlungsroutine könnte zufällig dieselben Datenstrukturen bearbeiten

### Lösung:

- Vorübergehendes Verbieten von Unterbrechungen durch Erhöhung des *ip* (interrupt priority level)
- Kritischer Bereich mit Erhöhung/Verringerung eingerahmt

## Beispiel einer Problemsituation (2)

- Prozess A will etwas von Datei 1 lesen
- A ruft read() und geht in Systemmodus
- BS sendet Befehl an Platte und legt A schlafen
- Neuer lauffähiger Prozess wird ausgesucht: B
- Prozess B will etwas von Datei 2 lesen
- B ruft read() und geht in Systemmodus
- BS sendet Befehl an Platte und legt B schlafen
- Neuer lauffähiger Prozess wird ausgesucht: C
- C läuft
- Platte erzeugt Unterbrechung, weil Daten von 1 vorliegen
- C wird unterbrochen und geht in Systemmodus
- BS nimmt Daten entgegen, schreibt sie in Puffer 1
- Hier könnte weitere Unterbrechung kommen, weil Daten von 2 vorliegen – an dieser Stelle aber unerwünscht und deshalb unterdrückt
- Danach: BS nimmt Daten von Datei 2 entgegen, schreibt sie in Puffer 2
- Neuer lauffähiger Prozess wird ausgesucht: noch immer C

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

140

Natürlich könnte das BS die Daten auch verzahnt entgegennehmen. Ist aber kritisch, weil ja die Datenstrukturen zur Verwaltung der Puffer nicht durcheinandergeraten dürfen. Sozusagen erhält man hier weitere Leistungssteigerung durch eine komplexere Programmierung des BS.



## Synchronisation im traditionellen Unix-Kern...

### Beim Einprozessorsystem

- Synchronisation problemlos möglich
- Nur kleinere Probleme
- Ununterbrechbarkeit des Kerns ist starker Schutz

(Bei Echtzeitbetriebssystemen ist die Ununterbrechbarkeit des Kern nicht mehr gegeben – damit neue Probleme)

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

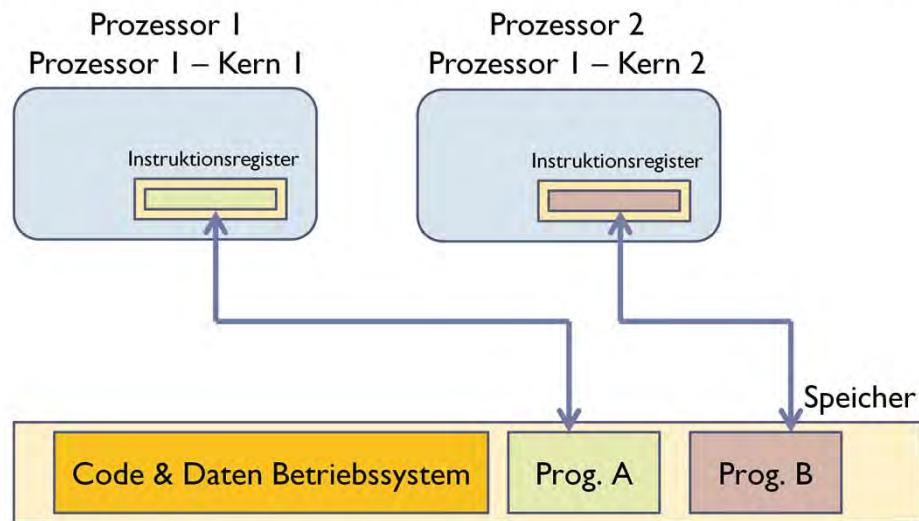
141

Siehe:

- [https://en.wikipedia.org/wiki/Realtime\\_operating\\_system](https://en.wikipedia.org/wiki/Realtime_operating_system)

## 2. Mehrprozessor/Mehrkern-Systeme

Mehrkern-Systeme sind seit ca. 2005 Standard



25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

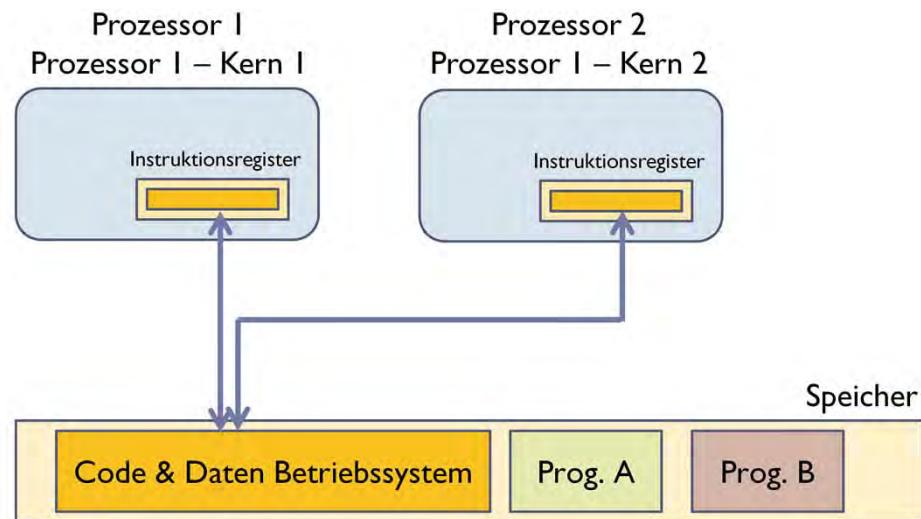
142

Siehe:

- [https://en.wikipedia.org/wiki/Multi-core\\_processor](https://en.wikipedia.org/wiki/Multi-core_processor)

Unkritischer Fall: Prozessoren bearbeiten Code von verschiedenen Programmen/Prozessen.

# Mehrprozessor/Mehrkern-Systeme...



25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

143

Kritischer Fall: Beide Prozessoren arbeiten gleichzeitig im Code des Betriebssystem.

# Mehrprozessor/Mehrkern-Systeme...

## Unkritisch

- Beide Prozessoren bearbeiten Code verschiedener Programme/Prozesse

## Beginn aller Probleme

- Ein Prozessor wechselt in den Systemmodus
  - Wegen Systemaufruf oder Unterbrechung
- Frage: Was macht anderer Prozessor?
  - Systemmodus verbieten? Ineffizient!
  - Systemmodus erlauben? Gefährlich!

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

144

Fall 1: Es läuft dasselbe Programm zwei Mal und die beiden Kerne arbeiten zufällig echt gleichzeitig diese beiden Prozesse ab. Die Code-Seiten sind nur einmal im Hauptspeicher, es wird lesend auf sie zugegriffen. Kein Problem. Die Daten der Prozesse sind in voneinander getrennten und somit geschützten Adressräumen. Auch kein Problem.

Fall 2: Zwei Prozesse verwenden dieselbe Bibliothek. Diese ist mit Code und Daten nur einmal im Hauptspeicher (shared library). Wichtig: Bibliotheken müssen jetzt auch wiedereintrittsfähig sein. Z.B. dürfen keine globalen Variablen verwendet werden. Stattdessen muss eine Bibliotheksroutine bei jedem Aufruf lokale Datenstrukturen erstellen, bzw. Parameter vom Aufruf übernehmen.

## Beispiel einer Problemsituation (3)

- Prozess A **auf Prozessorkern 1** will etwas von Datei 1 lesen
- A ruft read() und geht in Systemmodus
- **Echt gleichzeitig: Prozess B auf Prozessorkern 2 will etwas von Datei 2 lesen**
- B ruft read() und geht in Systemmodus
- **Ab hier Gefahr, dass beide Prozessorkerne identischen Code bearbeiten, z.B. den zur Pufferverwaltung**
- ...

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

145

Man sieht, dass jetzt Schutzmechanismen nötig sind, die verhindern, dass kritische Codeteile von jeweils mehr als einem Prozessorkern zu einem Zeitpunkt durchlaufen werden. Alle Schutzmechanismen hierzu laufen über Sperren.



## Mehrprozessor/Mehrkern-Systeme...

Was benötige ich alles?

- Spezial-Hardware zur Unterbrechungssteuerung an jedem einzelnen Prozessor
- Cache-Kohärenz-Mechanismen
- **Nebenläufig ausführbares Betriebssystem**
  - „Threads“ im Betriebssystem sind etwas anderes, als Threads in Prozessen

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

146

Siehe:

- [https://en.wikipedia.org/wiki/Thread\\_%28computer\\_science%29](https://en.wikipedia.org/wiki/Thread_%28computer_science%29)

„Nebenläufig“ (concurrent) bedeutet, dass zwei Aktionsstränge gleichzeitig ablaufen, diese aber nicht notwendigerweise miteinander zu tun haben. Von „parallel“ spricht man, wenn die Abläufe zu einem Programm gehören und logisch miteinander verknüpft sind, indem z.B. Synchronisationen erfolgen.

## 3. SMP-Hardware

### SMP Symmetric Multiprocessing

- Variante des Betriebssystems, die mehrere Prozessoren/Kerne unterstützen kann

### Intels Multiprocessor Specification MPS 1.4 (1995)

- Ursprünglich eingeführte Referenz
- Beschreibt Intel-SMP-Systeme
- Festlegung der BIOS-Eigenschaften
- Beschreibung des APIC
  - Advanced Programmable Interrupt Controller
  - Jetzt: x2APIC
- Cache-Kohärenz, MESI-Protokoll
- In der Praxis damals typisch: 2- und 4-Wege-Systeme

Heute enthalten in Advanced Configuration and Power Interface (ACPI)

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

147

Siehe:

- [https://en.wikipedia.org/wiki/Symmetric\\_multiprocessing](https://en.wikipedia.org/wiki/Symmetric_multiprocessing)
- [https://en.wikipedia.org/wiki/MultiProcessor\\_Specification](https://en.wikipedia.org/wiki/MultiProcessor_Specification)
- [https://en.wikipedia.org/wiki/Intel\\_APIC\\_Architecture](https://en.wikipedia.org/wiki/Intel_APIC_Architecture)
- <https://en.wikipedia.org/wiki/X2API>
- [https://en.wikipedia.org/wiki/Advanced\\_Configuration\\_and\\_Power\\_Interface](https://en.wikipedia.org/wiki/Advanced_Configuration_and_Power_Interface)

## **4. SMP-Betriebssystem**

Zunächst eine Begriffsbestimmung

- In einem SMP-System sind alle Prozessoren gleichberechtigt
- Sie greifen gemeinsam auf denselben Code und dieselben Daten des Betriebssystemkerns zu und stehen im Wettbewerb um Systemressourcen
- Jeder Benutzerprozess kann auf jedem Prozessor zur Ausführung gebracht werden



## SMP-Betriebssystem...

Was bedeutet das?

- Tatsächlich ist das SMP-Betriebssystem ein paralleles Programm auf einer Maschine mit gemeinsamem Speicher
- Der Code wird nebenläufig ausgeführt
  - (vgl. nebenläufig vs. parallel)
- Die Daten können beliebig manipuliert werden
- Inkonsistenzen vermeidet man durch Sperren
  - Sperren von Code-Abschnitten
  - Sperren von Datenbereichen

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

149

Siehe:

- [https://en.wikipedia.org/wiki/Concurrency\\_%28computer\\_science%29](https://en.wikipedia.org/wiki/Concurrency_%28computer_science%29)
- [https://en.wikipedia.org/wiki/Parallel\\_computing](https://en.wikipedia.org/wiki/Parallel_computing)



## SMP-Betriebssystem...

Alles dreht sich um die Sperren

- Eine große Sperre (sog. giant lock):  
Nur ein Prozessor kann in das Betriebssystem  
Daten bzgl. Der Konsistenz geschützt
  - Problem gelöst; Effizienz vernichtet
- Viele kurze Sperren:  
Daten bzgl. der Konsistenz geschützt
  - Gute Nebenläufigkeit bei geringen Kosten
- Ganz viele sehr kurze Sperren:  
Daten bzgl. der Konsistenz geschützt
  - Bessere Nebenläufigkeit bei höheren Kosten

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

150

Siehe:

- [https://en.wikipedia.org/wiki/Giant\\_lock](https://en.wikipedia.org/wiki/Giant_lock)

## SMP-Betriebssystem...

Wie mache ich mein Einprozessor-Einkern-Betriebssystem SMP-fähig?

- Analysiere alle Datenstrukturen und Abläufe im BS-Code
  - Zunächst Subsysteme wie Speicherverwaltung, Scheduling, Ein-/Ausgabe etc.
- Schütze kritische Bereich vor gleichzeitigem Zugriff
- Verfeinere den Schutz (mehr Nebenläufigkeit)
  - Inkrementelle Parallelisierung

## SMP-Betriebssystem... Linux

- Bei Linux dauert(e) dieser Vorgang Jahre!
  - Erstmals gut SMP-fähig in Kernel 2.2 (Jan. 1999)
- Im Kernel 2.4 (Jan. 2001)
  - Alle Subsysteme durch feingranulare Sperren abgesichert
  - „Kernel subsystems are fully threaded“
- Skalierbarkeit bzgl. Anzahl der Prozessoren ist problematisch
  - Bisherige Grenze: 4 (?)

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

152

Siehe:

- [https://en.wikipedia.org/wiki/Linux\\_kernel#Architecture](https://en.wikipedia.org/wiki/Linux_kernel#Architecture)

## 5. Synchronisationsmechanismen

- Es müssen jetzt verschiedenste Datenstrukturen geschützt werden, die bei einem einzelnen Prozessor unkritisch waren
- Einfache Flags reichen nicht aus, da diese gleichzeitig manipuliert werden könnten
- Unterbrechungssperren müssen global gesetzt werden können
- Weitere BS-Details
  - Traditioneller Sleep/wakeup-Mechanismus auf Mehrprozessor-Systemen unbrauchbar
  - Wiederaufwecken von Threads kritisch
    - Thundering-herd-problem

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

153

Siehe:

- [https://en.wikipedia.org/wiki/Thundering\\_herd\\_problem](https://en.wikipedia.org/wiki/Thundering_herd_problem)

Thundering-herd-problem: Beim Freiwerden einer Ressource werden viele Threads gleichzeitig deblockiert, nur um alle bis auf einen sofort wieder blockiert zu werden.

# Synchronisationsmechanismen...

Essentiell: Hardware-Unterstützung

- **Atomares Testen-und-Setzen**
  - Testet Bit, setzt Wert auf '1', gibt alten Wert zurück
  - Nach Abschluß ist das Bit '1'
  - Rückgabewert '0': man hat jetzt Zugriff, Ressource war frei
  - Rückgabewert '1': Ressource von anderen belegt
- Anweisung kann nicht einmal durch eine Unterbrechung unterbrochen werden
- In vielen Systemen sogar atomare Nutzung des Speicherbusses

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

154

Siehe:

- [https://en.wikipedia.org/wiki/Atomic\\_operation](https://en.wikipedia.org/wiki/Atomic_operation)

## Synchronisation mittels Semaphor

Standardverfahren:

- P() dekrementiert Semaphor und blockiert, wenn Wert kleiner 0 wird
- V() inkrementiert Semaphor und weckt Thread auf, wenn Wert kleiner oder gleich 0 ist

BS-Kern garantiert Atomarität der Aktion

- Einprozessor-Einkern-System:  
durch Ununterbrechbarkeit der BS-Kerns
- Mehrprozessor/Mehrkern-System:  
durch tieferliegende unteilbare Aktion

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

155

Siehe:

- [https://en.wikipedia.org/wiki/Semaphore\\_%28programming%29](https://en.wikipedia.org/wiki/Semaphore_%28programming%29)



## Synchronisation mittels Semaphor...

### Problem

- Blockieren und Aufwecken erfordert Kontextwechsel im Betriebssystem: langsam
- Nicht akzeptabel für kurze Blockierungen
- Weniger aufwendiger Mechanismus benötigt



## Synchronisation mittels Spinlock

### Einfachster Sperrenmechanismus: Spinlock

- engl: *spin lock*, *simple lock*, *simple mutex*
- Skalare Variable
  - '0' bedeutet verfügbar
  - '1' bedeutet belegt
- Manipulation mittels aktivem Warten (busy-wait) und atomarem Testen-und-Setzen

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

157

Siehe:

- <https://en.wikipedia.org/wiki/Spinlock>

## Synchronisation mittels Spinlock...

```
void spin_lock (spinlock_t *s)  {
    while (test_and_set (s) != 0) /* belegt */
        ;                      /* warte auf Freigabe */
}

void spin_unlock (spinlock_t *s)  {
    *s = 0;
}
```

Möglicher Nachteil: Busblockierung

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

158

test\_and\_set() ist eine unteilbare Aktion, typischerweise ein spezieller Maschinenbefehl, der durch Hardware-Unterstützung atomar ist.

## Synchronisation mittels Spinlock...

```
void spin_lock (spinlock_t *s)  {
    while (test_and_set (s) != 0) /*belegt*/
        while (*s != 0);
            /* warte auf Freigabe          */
            /* hier nur lesender Zugriff ! */
    }

void spin_unlock (spinlock_t *s)  {
    *s = 0;
}
```

# Synchronisation mittels Spinlock...

## Verwendung von Spinlocks

- Kurzzeitige Sperre kritischer Bereiche im Betriebssystem-Code
- Niemals für blockierenden Code einsetzen

## Analyse

- Aktives Warten (normalerweise unerwünscht)
- Sehr billig, wenn Ressource nicht belegt ist
- Insgesamt billig bei geringem Belegt-Grad

## 6. Erweiterte BS-Funktionalität

Was brauchen wir sonst noch?

- Feingranulare Ausführungsobjekte
  - Kernel-Threads
    - Werden im BS-Code erzeugt und teilen sich mit ihm den Adressraum
  - Speicherverwaltung für Mehrprozessor-Systeme
    - Verwaltung mehrerer Speicherbänke
  - Scheduling für Mehrprozessor-Systeme

## Erweiterte BS-Funktionalität

### Mehrprozessor-Scheduling

- Prozessor-Affinität

Ein Prozeß oder ein Thread sollte auf dem Prozessorkern fortgesetzt werden, auf dem er zuletzt lief

Grund: Gültiger Inhalt im Cache des Prozessorkerns

- Gang-Scheduling

Alle Threads eines Prozesses sollten zusammen zugeteilt werden

Grund: Verzögerungen an gemeinsam genutzten Sperren vermeiden

Siehe:

- [https://en.wikipedia.org/wiki/Processor\\_affinity](https://en.wikipedia.org/wiki/Processor_affinity)
- [https://en.wikipedia.org/wiki/Gang\\_scheduling](https://en.wikipedia.org/wiki/Gang_scheduling)

## Betriebssystemaspekte

### Zusammenfassung

- Synchronisation in Einprozessor-Einkern-Systemen fast ohne Probleme
- Bei SMP-Systemen läuft das Betriebssystem auf allen Prozessoren
- Hauptproblem: innere Synchronisation und Schutz der Datenstrukturen
- Betriebssystem-Code wird durch den Einbau von Sperren parallelisiert
- Feinere Sperren bedeuten mehr Nebenläufigkeit
- Hardware-Unterstützung für die Sperren notwendig
- Semaphor ist zu kostspielig
- Spinlock ist das wichtigste Synchronisationskonzept
- Scheduling von Threads ist sehr komplex

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

163



## Betriebssystemaspekte

### Die wichtigsten Fragen

- Wie funktioniert Synchronisation im traditionellen UNIX-Kern?
- Was versteht man unter Wiedereintrittsfähigkeit?
- Wie werden Maschinenbefehle in Mehrprozessor/Mehrkern-Systemen abgearbeitet?
- Was benötige ich für ein Mehrprozessor-System?
- Was kennzeichnet ein SMP-Betriebssystem?
- Welche Varianten von Sperren finden wir im SMP-Betriebssystem?
- Wie wird ein Betriebssystem SMP-fähig?
- Wie funktioniert Synchronisation mittels Semaphor?
- Wie funktioniert Synchronisation mittels Spinlock?
- Welche weiteren Funktionen muss ein SMP-Betriebssystem aufweisen?

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

164

# Programmierung mit OpenMP

1. Konzepte und `hello world`
2. Überblick
3. Parallelisierung einer Schleife
4. Eine komplexere Schleife
5. Parallele Bereiche
6. Lastausgleich
7. Parallele Abschnitte
8. Sequentielle Abschnitte
9. Bibliotheken und Compiler
10. Bewertung

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

165

Siehe:

- <http://www.openmp.org/>
- <https://en.wikipedia.org/wiki/OpenMP>
- <https://computing.llnl.gov/tutorials/openMP/> - sehr gutes Tutorial zum Thema

Bücher:

- B. Chapman et al.: Using OpenMP: Portable Shared Memory Parallel Programming. MIT Press, 2007, 353 Seiten.

Was bedeutet OpenMP?

- Kurze Version: Open Multi-Processing
- Lange Version: Open specifications for Multi-Processing via collaborative work between interested parties from the hardware and software industry, government and academia.

# 1. Konzepte und hello world

Automatische Parallelisierung durch Compiler immer noch nicht möglich

- Trotz langjähriger Forschung weder für Nachrichtenaustausch noch für gemeinsame Speicherbereiche

Aber: Compilergestützte Parallelisierung möglich

- Im Falle von OpenMP für gemeinsamen Speicher!

Alternativ: Bibliotheksbasierter Ansätze

- Message Passing Interface (MPI) für verteilten Speicher
- Pthreads für gemeinsamen Speicher

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

166

Der letzte Versuch für automatisch parallelisierende Compiler für Architekturen mit verteiltem Speicher, High Performance Fortran (HPF), wurde Ende der 90er erfolglos beendet.

Siehe:

- [https://en.wikipedia.org/wiki/High\\_Performance\\_Fortran](https://en.wikipedia.org/wiki/High_Performance_Fortran)

## Neuer Ansatz: OpenMP (Open Multi-Processing)

- Keine neue Programmiersprache
- Arbeitet mit Fortran und C/C++ zusammen
- Compiler-Direktiven steuern Übersetzung
- Zusätzliche (kleine) Bibliothek
- Direktiven+Bibliothek sind das API von OpenMP
- OpenMP-Compiler übersetzt in Programme mit Threads (nicht weiter spezifiziert)
  - Verwendung ausschließlich für gemeinsamen Speicher!

# OpenMPs Hello World

```
programm hello
    print *, "Hello world from thread:"
!$omp parallel
    print *, omp_get_thread_num()
 !$omp end parallel
    print *, "Back to the sequential world."
end
```

- Umgebungsvariable: **OMP\_NUM\_THREADS**
- Bibliotheksaufruf: **omp\_get\_thread\_num()**
- Compiler-Direktive: **!\$omp parallel**
  
- Thread-Nummern: 0...**OMP\_NUM\_THREADS-1**

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

168

Hier sieht man auch sofort alle drei Konstrukte, die OpenMP beinhaltet:  
Als wichtigstes die Compiler-Direktive, dann ein OpenMP-Bibliotheksaufruf  
und eine Verwendung von OpenMP-Umgebungsvariablen.

## Warum ein Compiler-Ansatz?

Zunächst reiner Compiler-Ansatz geplant

Vorteil gegenüber Bibliotheken

- Nicht-OpenMP-Compiler ignorieren parallele Konstrukte automatisch
- Compiler können zusätzlich optimieren
- Inkrementelle Parallelisierung möglich

Reiner Compiler-Ansatz zu schwierig

- Erweiterung durch einige einfache Bibliotheksaufrufe

## Geschichte von OpenMP

- OpenMP sehr neu: seit 1997
  - Version 3.0, Mai 2008: Neu ist das Konzept der *tasks*
  - Version 3.1, Juli 2011
  - Version 4.0, Juli 2013: Unterstützung für Beschleunigerkarten, für Fortran 2003 use.
- Hat aber lange Vorgeschichte
  - Ehemaliger ANSI X3H5-Standard zur Programmierung von gemeinsamem Speicher  
Früher auf parallelen Maschinen verbreitet
  - OpenMP jetzt von allen Herstellern akzeptiert
  - Von unabhängiger Organisation gefördert

## **2. Überblick**

Portabilität: Neuübersetzung reicht aus

Kategorien der Spracherweiterungen

- Kontrollstrukturen, um Parallelismus auszudrücken
- Datenumgebungskonstrukte zur Kommunikation zwischen Threads
- Synchronisationskonstrukte zur Ablaufsteuerung von Threads

## Überblick (2)

### Compiler-Direktiven

- in Fortran

```
!$OMP <directive> <clauses>
```

- In C/C++

```
#pragma omp <directive> <clauses>
```

### Zusätzlich bedingte Übersetzung der OpenMP-Bibliotheksaufrufe

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

172

Die Compiler-Direktiven sind für einen nicht OpenMP-Compiler nicht wirksam.

## Überblick (3)

### Parallele Kontrollstrukturen

- Ausführungsmodell genannt fork/join-Modell
- Parallelle Kontrollstrukturen starten neue Threads und übergeben ihnen die Kontrolle

### Zwei Varianten

- **parallel**-Direktive: umschließt Block und erzeugt Menge von Threads, die den Block nebenläufig abarbeiten
- **do**-Direktive: verteilt Instanzen von Schleifendurchläufen auf Threads

## Überblick (4)

Kommunikation und Datenumgebung

- Regelt, wer wann wo welche Daten sehen kann

Programm beginnt immer mit einem Thread  
(master-Thread)

Bei **parallel** werden neue Threads gestartet – jeweils mit eigenem Keller

Variablen können von folgenden Typen sein

- **shared** – allen Threads gemeinsam zugängliche Variable
- **private** – thread-lokale Variable
- **reduction** – Mischform zur Ergebniszusammenführung
- ...

## Überblick (5)

### Synchronisation

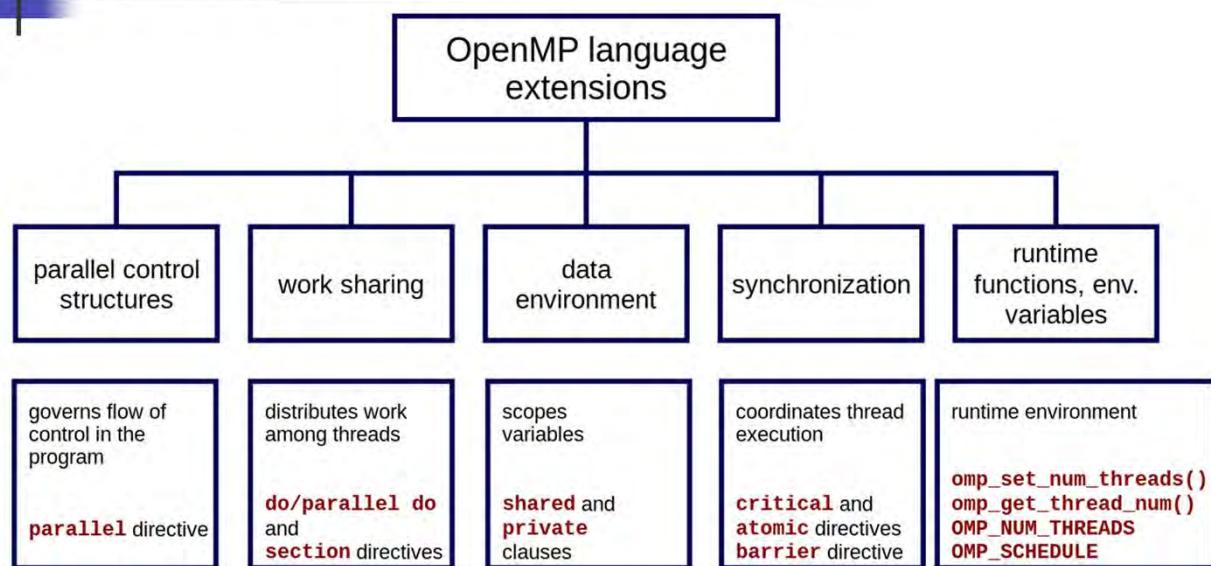
- Regelt den Ablauf der Threads

### Zwei Hauptformen

- Wechselseitiger Ausschluß mittels **critical**-Direktive
- Ereignis-Synchronisation mittels **barrier**-Direktive

### Weitere Konstrukte zur Bequemlichkeit oder Leistungsoptimierung

# Überblick (6)



25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

176

Quelle: Wikipedia, [en:Image:Omp lang ext.jpg](#)

### 3. Parallelisierung einer Schleife

```
subroutine saxpy(z,a,x,y,n)
integer i,n
real z(n),a,x(n),y

do i=1,n
    z(i)=a*x(i)+y
enddo

return
end
```

Keine Datenabhängigkeiten in der Schleife

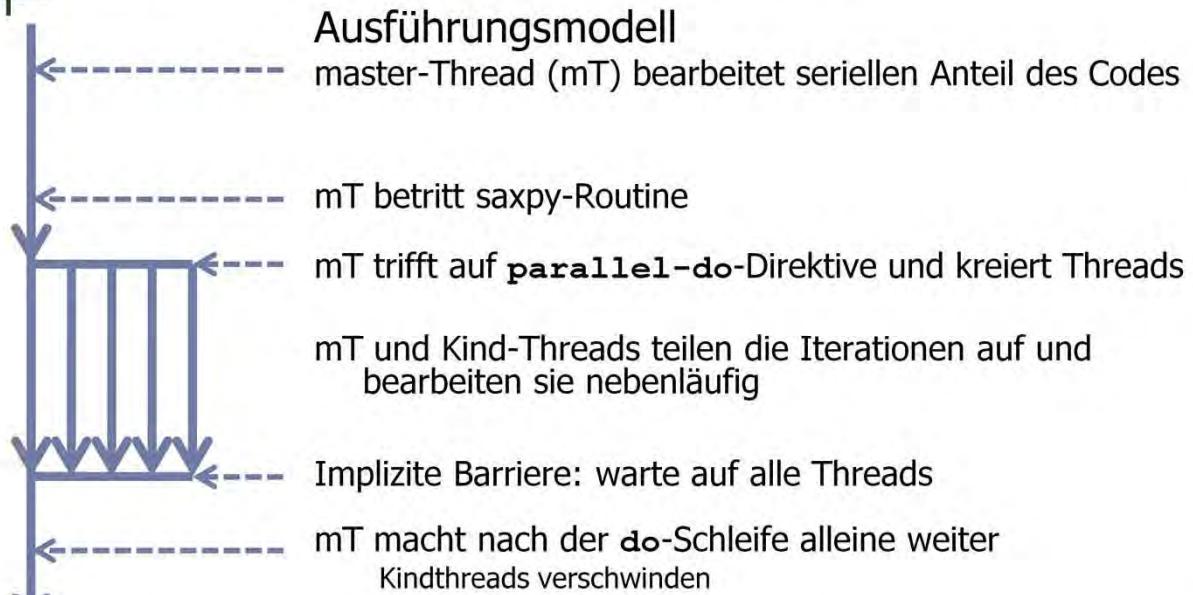
## Parallelisierung einer Schleife (2)

```
subroutine saxpy(z,a,x,y,n)
integer i,n
real z(n),a,x(n),y
 !$omp parallel do
 do i=1,n
   z(i)=a*x(i)+y
 enddo
 !$omp end parallel do
 return
end
```

Hier Parallelisierung alleine auf Schleifenindexebene

- Andere Ebenen auch möglich

## Parallelisierung einer Schleife (3)



25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

179

## Parallelisierung einer Schleife (6)

### Kommunikation und Datengültigkeit

- Außerhalb des **parallel-do**-Blocks
  - Die Variablen z, a, x, y, n, i sind nur einmal vorhanden
- Innerhalb des **parallel-do**-Blocks
  - Die Variablen z, a, x, y, n sind nur einmal vorhanden  
Vorsicht mit der Semantik beim Zugriff!
  - Die Schleifenvariable wird als thread-lokale Variable angelegt  
Aktualisierungen in einem Thread sind in anderen Threads nicht sichtbar

# Parallelisierung einer Schleife (7)

## Synchronisation

- Anforderungen
  - Variable z muß aktualisiert worden sein, wenn mit Anweisungen nach der Schleife fortgesetzt wird
- Realisierung
  - `parallel do`-Direktive hat implizite Barriere am Schleifenende

## 4. Eine komplexere Schleife

```
real*8 x,y
integer i,j,m,n,maxiter
integer depth(*,*)
integer mandel_val
...
maxiter=200

do i=1,m
    do j=1,n
        x=i/real(m)
        y=j/real(n)
        depth(j,i)=mandel_val(x,y,maxiter)
    enddo
enddo
```

## Eine komplexere Schleife (2)

### Zur Funktion `mandel_val`

- Darf nur von Eingabeparametern abhängen
- D.h. muß durch parallele Threads nutzbar sein (*thread-safe*)

### Zu den Variablen

- Variable `i` per default **private**  
(weil diese Schleife parallelisiert wird)
- Variablen `j,x,y` explizit auf **private** gesetzt  
(default wäre **shared**)

## Eine komplexere Schleife (3)

```
real*8 x,y
integer i,j,m,n,maxiter
integer depth(*,*)
integer mandel_val
...
maxiter=200
 !$omp parallel do private(j,x,y)
  do i=1,m
    do j=1,n
      x=i/real(m)
      y=j/real(n)
      depth(j,i)=mandel_val(x,y,maxiter)
    enddo
  enddo
 !$omp end parallel do
```

25.

184

## Eine Verkomplizierung

```
maxiter=200
do i=1,m
    do j=1,n
        x=i/real(m)
        y=j/real(n)
        depth(j,i)=mandel_val(x,y,maxiter)
        total_iters=total_iters+depth(j,i)
    enddo
enddo
```

Mitzählen der gesamten Iterationen

Variable **total\_iters** per default **shared**

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

185

## Eine Verkomplizierung (2)

Zugriff auf **total\_iters** in kritischem Bereich

```
!$omp critical
    total_iters=total_iters+depth(j,i)
 !$omp end critical
```

Verfahren wird beim Zugriff auf **total\_iters** serialisiert

- Zugriffszeit sollte prozentual kleiner Anteil sein!

# Reduktion

```
maxiter=200
total_iters=0
!$omp parallel do private(j,x,y)
!$omp+ reduction(+:total_iters)
do i=1,m
  do j=1,n
    x=i/real(m)
    y=j/real(n)
    depth(j,i)=mandel_val(x,y,maxiter)
    total_iters=total_iters+depth(j,i)
  enddo
enddo
 !$omp end parallel do
```

25.

187

Die Variable `total_iters` wird hierbei erst als private-Variable behandelt. Ihre Aktualisierung ist somit unkritisch. Nach Abschluß der Schleife erfolgt die Reduktion, bei der allen privaten Einzelvariablen zur gemeinsam genutzten außerhalb der Schleife aufaddiert werden.

Das `'+'` bei `'!$omp+` kennzeichnet die Fortsetzung der vorhergehenden Zeile. Fortran-Notation.

# Schleifenparallelisierung

Hauptproblem der Praxis:

Datenabhängigkeiten zwischen Schleifenindizes

Bspiel:

```
do i=2,n  
  a(i)=a(i)+a(i-1)  
enddo
```

Lösung des Problems

- Komplizierte Methoden zum Finden der Abhängigkeiten
  - Großes Forschungsgebiet seit >20 Jahren
- Verschiedene Methoden zu ihrer Beseitigung
  - Zusätzliche Variable
  - Zugriffskoordination mit kritischem Bereich

## 5. Parallelle Bereiche

Bisher nur parallele Schleifen (feingranular)

Jetzt auch grobgranularer Parallelismus

Konstrukt: **parallel / end parallel**

- eingeschlossener Code wird mit mehreren Threads nebenläufig bearbeitet

## Parallele Bereiche (2)

```
maxiter=200
do i=1,m
  do j=1,n
    x=i/real(m)
    y=j/real(n)
    depth(j,i)=mandel_val(x,y,maxiter)
  enddo
enddo
do i=1,m
  do j=1,n
    dith_depth(j,i)=0.5*depth(j,i) +
$           0.25*(depth(j-1,i)+depth(j+1,i))
  enddo
enddo
```

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

190

Zwei Einzelberechnungen in Schleifen nacheinander ausgeführt.

## Parallele Bereiche (3)

```
maxiter=200
!$omp parallel
!$omp+ private(i,j,x,y)
!$omp+ private(my_width,my_thread,i_start,i_end)
    my_width=m/2
    my_thread=omp_get_thread_num()
    i_start=1+my_thread*my_width
    i_end=i_start+my_width-1
    do i=i_start,i_end
        do j=1,n
            x=i/real(m)
            y=j/real(n)
            depth(j,i)=mandel_val(x,y,maxiter)
        enddo
    enddo
```

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

191

Das Ganze wird hier auf 2 Threads aufgeteilt. Jeder Thread ermittelt seine Nummer und bestimmt daraus den Block von Indizes, die er durchlaufen muss.

## Parallele Bereiche (4)

```
do i=i_start,i_end
    do j=1,n
        dith_depth(j,i)=0.5*depth(j,i) +
$           0.25*(depth(j-1,i)+depth(j+1,i))
    enddo
enddo
!$omp end parallel
```

Diese Parallelisierung ist für genau 2 Threads  
programmiert  
Keine Compiler-Parallelisierung auf Schleifenebene

## 6. Lastausgleich

Standard: jeder Thread erledigt gleich viele Iterationen einer Schleife

Aber: falls Schleifenrumpf in der Bearbeitungszeit variiert, führt das zu Lastungleichheit

Mechanismus: **schedule**-Klausel

- **static**: Zuteilung der Indizes zu Schleifenbeginn
- **dynamic**: Indizes werden zur Laufzeit zugeteilt

## Lastausgleich (2)

**schedule (type [ , chunk ] )**

- **static**: etwa Gleichverteilung
- **static chunk**: Rundumverteilung von Blöcken der Größe **chunk**
- **dynamic**: dynamische Rundumverteilung von Blöcken der Größe **chunk** (default=1)
- **guided**: Die Blockgröße sinkt exponentiell bis auf **chunk** ab; dynamische Rundumverteilung
- **runtime**: Verfahren wird durch die Umgebungsvariable **OMP\_SCHEDULE** bestimmt

## 7. Parallelle Abschnitte

Bei nichtiterativen Arbeitslasten:  
Zuteilung von Code zu Threads

```
!$omp section [clause [,] [clause...]]
[ !$omp section]
    code for the first section
[ !$omp section
    code for the second section
    ...
]
!$omp end sections [nowait]
```

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

195

Mit Klauseln kann man wieder z.B. die Verwaltung der Variablen oder das Scheduling steuern.

## Parallele Abschnitte (2)

- Die Anzahl der gewählten Threads bearbeitet unabhängig die Abschnitte
- Jeder Abschnitt genau einmal durchlaufen
- Nicht bestimmbar, welcher Thread welchen Abschnitt bearbeitet
- Nicht bestimmbar, wann welcher Abschnitt an die Reihe kommt
- Deshalb: Ausgabe eines Abschnittes sollte nicht Eingabe für einen anderen sein

## 8. Sequentielle Abschnitte

Parallele Abarbeitung manchmal zeitweilig nicht erwünscht

```
!$omp single [clause [,] [clause...]]
    Anweisungsblock der nur von einem
    Thread bearbeitet wird
!$omp end single [nowait]
```

Keine Barriere zu Beginn des sequentiellen Abschnitts

Mittels **nowait** warten Threads nicht auf das Ende des sequentiellen Abschnitts

## Sequentielle Abschnitte (2)

Beispiel: Ausgabe

```
!$omp parallel shared (out,len)
...
 !$omp single
   call write_array(out,len)
 !$omp end single nowait
 ...
 !$omp end parallel
```

## Ereignissynchronisierung

Barrieren: alle Threads warten an der Barriere, bis alle parallelen Threads eingetroffen sind – dann erfolgt die Fortsetzung der Arbeit

**`!$omp barrier`**

Ordnung: erzwingt ein Durchlaufen von Anweisungen in der ursprünglichen Reihenfolge der Indexwerte (d.h. wie in einem sequentiellen Programm)

**`!$omp ordered  
block  
 !$omp end ordered`**

## 9. Bibliotheken und Compiler

### Bibliotheken

Zusammenbinden von OpenMP-Programmen mit Bibliotheken von Dritten

- Es muss sichergestellt sein, dass die Bibliotheksaufrufe *thread-sicher* sind
- Andernfalls Bibliothek nicht in parallelen Bereichen verwenden
- Oder z.B. als kritischen Bereich kennzeichnen
  
- Heutzutage sind allerdings die meisten Bibliotheken schon *thread-sicher*



## Bibliotheken und Compiler (2)

### Compiler

Früher: spezielle Präprozessoren und Compiler

Heute: in alle gängigen Compiler integriert

- Unterstützung variiert aber!

Beispiele für OpenMP 3.1

- GCC 4.7
- Intel Fortran and C/C++ compilers 12.1
- LLVM/Clang 3.7

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

201

Siehe:

- <http://openmp.org/wp/openmp-compilers/>

## 10. Bewertung

### Vorteile

- Portabler Multithread-Code
- Einfach zu programmieren
- Inkrementelle Parallelisierung: beginne mit kleinen Änderungen
- Einheitlicher Code für sequentielle und parallele Programmversion
- Sequentieller Code muss nicht modifiziert werden, dadurch keine versehentlichen neuen Fehler
- Parallelismus auf verschiedenen Ebenen möglich
- Kann mit verschiedenen Beschleunigern verwendet werden
- Kann mit Nachrichtenaustausch gemischt werden

## Bewertung (2)

### Nachteile

- Risiko für schwer erkennbare Fehler bei Synchronisationen
- Risiko für unerkannte Datenabhängigkeiten in Schleifen
- Läuft nur auf Architekturen mit gemeinsamem Speicher
- Skalierbarkeit und Leistungsausbeute durch die Architektur begrenzt
- Zunächst vermeintlich einfach programmierbar – hohe Leistungsausbeute dann aber nicht einfach erzielbar

## **Programmierung mit OpenMP**

### **Zusammenfassung**

- OpenMP wird ausschließlich für Architekturen mit gemeinsamem Speicher verwendet
- OpenMP ist ein Ansatz, der auf Compiler-Direktiven aufbaut
- OpenMP-Programme mit regulärem Compiler problemlos übersetzbbar
- Konstrukte zur Parallelisierung von Schleifen (feingranular) und anderen Code-Bereichen (grobgranular)
- Konstrukte zur Kontrolle der Variableninstanzen in den Threads
- Konstrukte zur Instanziierung von Threads und zu deren Beendigung
- Konstrukte zur Synchronisation der Threads untereinander
- OpenMP bildet mit MPI den Standard der parallelen Programmierung für alle modernen Maschinen

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

204

# **Programmierung mit OpenMP**

## **Die wichtigsten Fragen**

- Was charakterisiert den Ansatz von OpenMP?
- Welche Konzeptklassen beinhaltet OpenMP?
- Welche Konstrukte zur Parallelarbeit gibt es?
- Wie werden Variablen verwaltet?
- Welche Synchronisationskonzepte gibt es?
- Wie werden Schleifen parallelisiert?
- Was ist das Hauptproblem der parallelen Schleifen?
- Wofür verwendet man sequentielle Abschnitte?
- Wie programmiert man allgemeine Parallelarbeit?
- Welche Konzepte zum Lastausgleich gibt es?

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

205



# Optimierung sequentieller Programme

1. Motivation und Ansätze
2. Ebenen und Potentiale
3. Anwendungsklassen
4. Programmiersprachen
5. Leistungsabschätzungen
5. Optimierung der Mathematik
6. Optimierung der Programmierung
7. Optimierung mit dem Compiler
8. Fazit

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

206

Siehe:

- [https://de.wikipedia.org/wiki/Effizienz\\_%28Informatik%29](https://de.wikipedia.org/wiki/Effizienz_%28Informatik%29)
- <https://de.wikipedia.org/wiki/Komplexit%C3%A4stheorie>
- <https://de.wikipedia.org/wiki/Sortierverfahren>
- <http://www.arsTechnica.de/computer/prog.html>
- <http://www.dorn.org/uni/sls/>
- <http://wwwseidl.in.tum.de/lehre/vorlesungen/WS06/optimierung/>



Programmierung ist eine Krücke, die wir nur benutzen,  
weil wir noch nicht weit genug sind, mathematische  
Darstellungen direkt in Ergebnisse zu transformieren

Mathematik  
Numerik  
Programm  
Ergebnisdaten

# 1. Motivation und Ansätze

„Gefühlte Programmgeschwindigkeit“

- Wichtig für den eigenen Arbeitsablauf
- Stark situationsabhängig
- Psychologische Effekte beachten

## Beispiele

- Reaktionszeit beim Anklicken eines Knopfes
  - Nach max. 2 Sekunden sollte eine Rückmeldung kommen
- Ausführungszeit der angeklickten Operation
  - Beliebig, aber durch Benutzererwartungen beschränkt
  - Vorhersagbarkeit wäre gut
  - Falls nicht das, dann wenigstens Fortschrittsanzeige

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

208

Je nach Arbeitsablauf darf das Programm auch langsam sein – ich muss ja vielleicht erst wieder nachdenken.

1ms-100ms mag egal sein, 1s-100s ist ein großer Unterschied

Zur Reaktionszeit siehe GNOME Human Interface Guideline und „Acceptable Response Times“

## Motivation (2)

Programme mit GUI (Geschäftsprogramme)

- Schnelle Reaktion der GUI
- Beliebige Reaktion des Programms

Programme auf Kommandozeile (technisch/wissensch.)

- Meist wünscht man sich kürzere Programmlaufzeiten

Programme in Maschinensteuerungen

- Vorgegebene maximale Laufzeit (Echtzeitprogramme)  
Z.B. Auslösung eines Airbags

# Ansätze

## Wir warten auf schnellere Hardware

- Ging von 1941 bis ca. 2005
  - Permanente Leistungssteigerung der Prozessoren unter Beibehaltung des Nutzungskonzepts
    - Compiler erzeugt Programm für den Prozessor (1bit – 64bit)
  - Seit ca. 2005 Mehrkernprozessoren auch im PC
    - D.h. weitere Leistungssteigerungen nur noch durch manuelles Parallelisieren des Codes
      - Leider kann der Compiler das nicht (bzw. nur unzureichend)

## Wir parallelisieren den Code

- Konzept seit den 1970ern genutzt
- Seit den 1990ern Rechnercluster, dann auch mit Linux
- Wird später besprochen

## 2. Ebenen und Potentiale

### Mathematisch/algoritmische Ebene

- Welche alternativen mathematischen Verfahren sind in der Ausführung schneller?

### Programmiersprachliche Ebene

- Geeignete Verfahren
  - Welcher Algorithmus läuft am besten?
  - Welche Datenstrukturen sind am geeignetsten?
- Optimale Anpassung an die Architektur
  - Wieviel Hauptspeicher hat mein Rechner?
- Optimale Anpassung an den Compiler
  - Wie legt der Compiler die Daten im Speicher ab?

## Ebenen der Optimierung (2)

### Compilerebene

- Welche Optimierungen führt der Compiler durch?
- Wie kann ich Optimierungen gezielt auswählen?

### Hardware-Ebene

- Kann ich meinen Rechner an das Problem anpassen?
  - Z.B. Einbau von mehr Hauptspeicher
  - Z.B. Einbau von speziellen Beschleunigerkarten (GPGPU, FPGA)

## Benötigte Methodenkenntnisse

- Wissen über die Anwendung
  - Wissen zu mathematischen Verfahren
  - Wissen zu Programmiertechniken
  - Wissen über Compilerkonzepte
  - Wissen über Rechnersysteme
- + Wissen über das Zusammenwirken aller fünf Ebenen**

Wunschdenken des Naturwissenschaftlers

Mich kümmert nur meine Naturwissenschaft!

- Geht klar, aber dann wird das Werkzeug Computer nicht optimal eingebunden werden können

Disziplinabhängige Unterschiede: Physiker vs. alle anderen

# Optimierungspotentiale

## Mathematik

- Sehr hoch
  - Komplexitätsverringerung der Verfahren bringt Größenordnungen

## Programmiertechnik

- Sehr hoch
  - Komplexitätsverringerung der Algorithmen bringt Größenordnungen
  - Effiziente Datenstrukturen bringen vielleicht noch eine Größenordnung
  - Optimale Anpassung an eingesetzte Hardware bringt vielleicht auch noch eine Größenordnung

## Compiler

- Mittel
  - Optimierungen des Maschinencodes werden durchgeführt

## Optimierungspotentiale (2)

Hardware-Umbauten im normalen Rechner

- Mittel bis hoch, aber schwierig in der Umsetzung

Hardware-Umbau: Erwerb eines Hochleistungsrechners

- Sehr hoch: Faktor 10 bis 1.000.000
  - Schwierig in der Realisierung

Wahl einer optimalen Programmiersprache

- Gering
- Komfort vs. Geschwindigkeit
- Mathematische und programmiertechnische Optimierungen mit jeder Sprache möglich
  - Pfusch ebenso!

### **3. Anwendungsklassen**

#### Geschäftsssoftware (als Gegenbeispiel)

- Oft nicht zeitkritisch in der Ausführung, weil eher kurz
- Ggf. Einmaloptimierung und dann langer Produktionsbetrieb

#### Wissenschaftliche Software

- Typischerweise oft zeitkritisch, weil komplexe Berechnungen
- Probleme
  - Ständiger Wandel des Codes, der z.B. ein mathematisches Modell realisiert
  - Wenig Produktionsbetrieb mit unverändertem Code
  - Wissenschaftler hat keine Zeit zur Codeoptimierung
  - Wissenschaftler hat keine Kenntnis über Möglichkeiten

## (Traurige) Tatsachen

### Kein systematisches Leistungs-Engineering

- Kenntnisse über Optimierungen auf allen Ebenen sind nur bruchstückhaft bei den Anwendern vorhanden
- Keine Lehre zu diesem Thema
- Nahezu keine schriftlichen Unterlagen
- Niemand weiß, wie schnell das Programm sein müsste

### Ausnutzung der nominellen Prozessorleistung gering

- In vielen Fällen werden nur 5-10% der **Rechenleistung** genutzt
- Gründe beispielhaft:
  - Sprünge im Code, nicht genügend Mathematik im Code

## (Traurige) Tatsachen (2)

Wissenschaftliche Software meist schlecht optimiert

- Ergebnisse kommen zu langsam
- Ressourcen werden nicht optimal genutzt
  - Klimacodes für IPCC AR5 brauchen am DKRZ 30 MCPUh und das kostet 1 MEuro für Strom

Energiekosten sind jetzt ein wichtiger Faktor

- Nicht mehr alleine interessant: time-to-solution  
Sondern auch: kWh-to-solution

# Kosten/Nutzen-Analyse

## Kosten

- Einführung neuer Mathematik
- Verbesserung der Programmstruktur
- Installation optimierter Bibliotheken

## Nutzen

- Verkürzte Programmlaufzeit

## Sinnvolles Vorgehen

- Aufgewandte Zeit und gesparte Zeit in Relation setzen
- Unterm Strich sollte eine Ersparnis herauskommen
- Aufwand für Optimierung an das Einsparpotential anpassen

# Die Wahl der Programmiersprache

C

- Gute Anpassung an Hardware möglich
- Programmierung vergleichsweise maschinennah
- Effizienter Maschinencode

C++

- Vorteile/Nachteile von C
- Zusätzliche objektorientierte Programmierung

Fortran

- Gute Anpassung an Mathematik
- Programmierung nicht so maschinennah wie C
- Trotzdem effizienter Maschinencode

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

220

## Die Wahl der Programmiersprache (2)

### Java

- Gute Programmierkonzepte und hoher Programmierkomfort
- Keine optimale Anpassung an die Hardware
- Keine optimale Anpassung an die Mathematik
- Einigermaßen effizienter Maschinencode

### Skriptsprachen / Matlab etc.

- Schnelle Programmerstellung möglich
- Komfort geht auf Kosten der Laufzeitoptimierung

### Zusammenfassung

- Leistungsausbeute bei Sprachen hängt eher vom Vermögen des Programmierers ab
- Objektorientierung kostet Leistung und bringt Komfort

## Ideale Vorgehensweise

- Sauberer Entwurf der Mathematik und der Implementierung
  - Nicht nur wegen Laufzeit sondern auch wegen
    - Fehlerfreiheit, Wartbarkeit, Erweiterbarkeit
- Messen der Programmlaufzeiten
- Bewerten
  - Kann ich mit dieser Laufzeit meine wissenschaftliche Arbeit durchführen?
- Aufdecken von Leistungsengpässen
  - Welche Werkzeuge gibt es?
- Beseitigung von Leistungsengpässen
  - Die wichtigsten zuerst

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

222

## 4. Leistungsabschätzungen

### Theoretisch

- Komplexitätsmaße für Zeit- und Speicherbedarf ermitteln
  - Sehr schwierig – am besten ggf. Literatur heranziehen

### Praktisch

- Laufzeiten und Speichernutzungen messen
  - Das kann jeder

# Theoretische Leistungsabschätzungen

Lernt der Informatiker in der Komplexitätstheorie

In aller Kürze:

- Zeitbedarf und Speicherbedarf haben eine funktionale Abhängigkeit von der Anzahl und Größe der Eingabedaten
- Bezeichnet durch  $O(X)$ , wobei X eine Funktion von n ist

Beispiele (für Laufzeitkomplexität)

- $O(n)$ : Das Programm ist linear von n abhängig
  - Beispiel: Durchlaufen aller Eingabewerte und Maximum finden
- $O(n^2)$ : Das Programm ist quadratisch von n abhängig
  - Beispiel: Schlechte Sortierverfahren, die alle Werte mit allen vergleichen

# Theoretische Leistungsabschätzung (2)

## Auszug aus Sortierverfahren bei Wikipedia

Sortierverfahren	Best-Case	Average-Case	Worst-Case
AVL Tree Sort (höhen-balanciert)	$\mathcal{O}(n)$	$\mathcal{O}(n \cdot \log(n))$	$\mathcal{O}(n \cdot \log(n))$
Binary Tree Sort	$\mathcal{O}(n \cdot \log(n))$	$\mathcal{O}(n \cdot \log(n))$	$\mathcal{O}(n^2)$
Bogosort	$\mathcal{O}(n)$	$\mathcal{O}(n \cdot n!)$	$\infty$
Bubblesort (Vergleiche) (Kopieraktionen)	$\mathcal{O}(n)$ $(n - 1)$ $(0)$	$\mathcal{O}(n^2)$ $\approx \left(\frac{n^2}{4}\right)$ $\approx \left(\frac{n^2}{4}\right)$	$\mathcal{O}(n^2)$ $\approx \left(\frac{n^2}{2}\right)$ $\approx \left(\frac{n^2}{2}\right)$
Combsort	$\mathcal{O}(n \cdot \log(n))$	$\mathcal{O}(n \cdot \log(n))$	$\mathcal{O}(n^2)$
Gnomesort	$\mathcal{O}(n)$		$\mathcal{O}(n^2)$
Heapsort	$\mathcal{O}(n \cdot \log(n))$	$\mathcal{O}(n \cdot \log(n))$	$\mathcal{O}(n \cdot \log(n))$

Wichtig: welches ist die minimale Komplexität?

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

225

## Theoretische Leistungsabschätzung (3)

### Weitere Beispiele

- O(1): Die Laufzeit ist fest und hängt nicht von n ab
  - Beispiel: Ein Programm, das immer gleich abstürzt ☺
- O(log n): Die Laufzeit hängt logarithmisch von n ab
  - Beispiel: Suche in einem Binärbaum
- O( $n^k$ ): Die Laufzeit hängt polynomial von n ab
  - Beispiele: kaum welche mit Praxisrelevanz für  $k > 2$
- O( $2^n$ ): Die Laufzeit hängt exponentiell von n ab
  - Beispiele: viele theoretische, die praktisch nicht berechnet werden können

### Beachte

- Bei sehr kleinen n ist manchmal auch eine höhere Komplexität noch akzeptabel (z.B. O( $n^2$ ) beim Sortieren statt O( $n \log n$ ))
- Die Bestimmung der Komplexität ist sehr komplex ☺

# Praktische Leistungsabschätzung

Es gibt verschiedene Messwerkzeuge

- Wenige für sequentielle Programme
- Einige komplexe für parallele Programme

Unter Linux

- Kommandos **time** (der Shell) und **/usr/bin/time**
  - Letzteres zeigt auch den Speicherverbrauch und andere Daten
- Einfach auf der Kommandozeile dem Programmaufruf voranstellen
  - Ermittelt die Gesamtlaufzeit des Programms
- Kommando **gprof**
  - Programm mit Compileroption für Profiling übersetzen (gcc: -pg)
  - Laufenlassen des Programms erzeugt Datei gmon.out
  - Kann mit gprof angesehen werden
- Kommando **perf**

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

227

Siehe:

- <http://www.brendangregg.com/perf.html>
- <https://perf.wiki.kernel.org/index.php/Tutorial>

## Praktische Leistungsabschätzung (2)

### Beispiel (Hager/Wellein):

%	cummulative	self		self	total	
Time	seconds	seconds	calls	ms/call	ms/call	name
70.45	5.14	5.14	26074562	0.00	0.00	intersect
26.01	7.03	1.90	4000000	0.00	0.00	shade
3.72	7.30	0.27	100	2.71	73.03	calc_tile

### Erläuterung

- self seconds ist die Laufzeit in der Funktion
- cummulative seconds ist die aufsummierte Laufzeit, wenn nach self seconds sortiert wird

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

228

Kleine Vorschau auf Optimierung bei parallelen Programmen: wenn wir die Funktion intersect auf immer mehr Prozessoren verteilen, dann sinkt die Programmalaufzeit um theoretisch 5,14 Sekunden und geht gegen 2,15 Sekunden. Man erreicht also maximale eine etwa Verdreifachung der Geschwindigkeit, egal, wieviele Prozessoren man einsetzt. ☺

## Praktische Leistungsabschätzung (3)

### Vorgehensweise

- Wir optimieren die Funktionen mit dem höchsten Zeitanteil
  - Hier zunächst intersect
- Eine Abschätzung des Optimierungspotentials der einzelnen Funktionen gibt Aufschluss über das Gesamtpotential

### Aspekte von gprof

- Funktionsbasiert – d.h., wer sein Programm nicht in Funktionen unterteilt, kann nichts messen ☺
- Inlining von Funktionen durch den Compiler muss korrekt behandelt werden, sonst sind die Messwerte falsch
  - Inlining: der Compiler ersetzt im Maschinencode einen Funktionsaufruf durch die Funktion selber

## 5. Optimierung der Mathematik

- Am besten zusammen mit den Mathematikern
  - Kooperationen mit Numerikern/Optimierern
- Bessere mathematische Verfahren brauchen Zeit für die Entwicklung und Evaluation
- Kann oft nicht vom Naturwissenschaftler geleistet werden
  - Muß aber in Zusammenarbeit mit ihm erfolgen, da meist die Kenntnis der Anwendung von Nöten ist

## Optimierung der Mathematik (2)

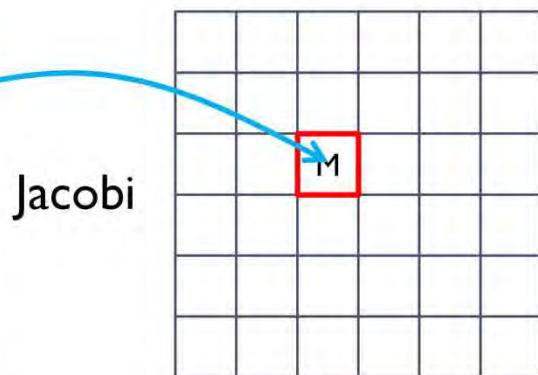
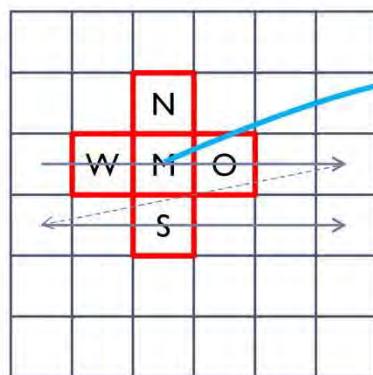
Beispiel: partielle Differentialgleichungen mittels Jacobi- oder Gauß/Seidel-Verfahren

- Löst ein lineares Gleichungssystem
- Z.B. für folgende Anwendung: wir erwärmen eine Platte an den Ecken auf eine bestimmte Temperatur – wie ist dann die Verteilung der Temperatur über die Platte hinweg?

Bewertung:

- Gauß-Seidel-Verfahren konvergiert schneller
- Seit neuestem aber: Jacobi lässt sich für hohe Anzahl von Prozessoren besser parallelisieren

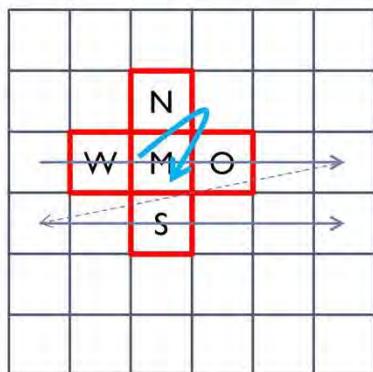
# Optimierung der Mathematik (3)



Jacobi

- Es werden zwei Matrizen verwandt: eine mit den aktuellen Werten und eine für die Werte der nächsten Iteration
- Der neue Wert M wird aus den alten Nachbarwerten von M (N, W, S und O) ermittelt
- Wenn alle neuen Werte bestimmt sind, werden die beiden Matrizen getauscht und die nächste Iteration beginnt
- Das Verfahren endet, wenn für alle neuen M die Änderung kleiner einer unteren Schranke ist

# Optimierung der Mathematik (4)



## Gauß-Seidel

- Nur eine Matrix verwandt, also weniger Speicher *und* auch schneller
- Der neue Wert M wird aus den Nachbarwerten von M (N, W, S und O) ermittelt
- Hier jetzt: N und W wurden bereits aktualisiert, S und O noch nicht. Das mathematische Verfahren läuft somit anders ab
- Das Verfahren endet, wenn für alle neuen M die Änderung kleiner einer unteren Schranke ist

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

233

# 6. Optimierung der Programmierung

Am besten zusammen mit den Informatikern

## Drei Ebenen

- Pure Programmierung ohne Berücksichtigung von Compiler und Hardware
- Programmoptimierung im Zusammenspiel mit dem Compiler
- Programmoptimierung im Zusammenspiel mit der Hardware

## Allgemeine Probleme

- Bei einem Wechsel der Zielarchitektur müssen die Optimierungen erneut evaluiert und dann angepasst oder ausgetauscht werden
- Dasselbe gilt bei einem Wechsel auf parallele Architekturen

# Optimierung der Programmierung (2)

## Unabhängig von Compiler und Hardware

- Effiziente Algorithmen
  - Effizientes Sortieren, effizientes Suchen
- Effiziente Datenstrukturen
  - Listen, Bäume, Hashtabellen, dünn besetzte Matrizen

## Findet man in Büchern und Vorlesungen

- Informatikergrundvorlesung „Algorithmen & Datenstrukturen“
  - Das Minimum dessen, was der Naturwissenschaftler wissen sollte!
- Amazon: „Algorithmen und Datenstrukturen“

# Optimierung der Programmierung (3)

## Beispiel: dünnbesetzte Matrizen

- NxN Einträge, aber nur 0,1% sind ungleich von null

## Speicherung

- Ablage in einer verketteten Liste (einfach oder doppelt) mit Angabe der x,y-Koordinate
- Zusätzlich noch ein Feld mit Zeigern auf z.B. jedes 1000ste Element

## Zugriff

- Einstieg an einem der Zeiger, Ablaufen der Liste bis zur gewünschten Koordinate

## Matrizenmultiplikation

- Wird jetzt ganz neu implementiert

# Optimierung der Programmierung (4)

Abhängig vom Compiler

Beispiel:

- Abbildung von logischen Datenstrukturen in den Hauptspeicher
- Hier: zweidimensionale Felder
- Z.B. wird zeilenweise in den Hauptspeicher abgebildet
- Programm lese z.B. die Werte zeilenweise oder spaltenweise
  - Was passiert mit der Zugriffszeit?
- Man würde meinen: gar nichts – wäre da nicht der Cache
  - Der holt sich nicht nur den fehlenden Wert sondern noch mehrere andere

# Optimierung der Programmierung (5)

1	2	3
4	5	6
7	8	9

logische  
Datenstruktur

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---

Speicherabbild

Cache

a	b	c
1	2	3
x	y	z

Cacheinhalt  
nach Zugriff auf  
Element '1'

- Nach Zugriff auf '1' ist eine Cacheline geladen, gerade eben die Elemente '1', '2' und '3'
- Greift das Programm auf '2' und '3' zu, so geht dies schnell (Cache-Hits)
- Greift das Programm nach '1' auf '4' zu, so muß eine zweite Cacheline '4', '5' und '6' geladen werden
  - Das kostet Zeit! (Cache-Miss)
  - Dasselbe Problem wiederholt sich, wenn dann auf '7' zugegriffen wird

# Optimierung der Programmierung (6)

Abhängig von der Hardware

Beispiel 1:

- Wir haben 2 GB Hauptspeicher
- Das Programm hat aber viele GB virtuellen Adressraum
- Daten, die nicht im Hauptspeicher gehalten werden können, werden auf die Platte ausgelagert (Swapping)
- Das kostet Zeit!
- Also: Datenstrukturen des Programms an freien Platz anpassen

Beispiel 2:

- Im Rechner ist eine zusätzliche Grafikkarte verbaut
- Wir könnten diese zur Beschleunigung von Berechnungen verwenden

Beispiel 3:

- Das Programm wurde für einen 32bit-Prozessor entwickelt
- Es soll jetzt auf einem 64-bit Prozessor laufen
- Im einfachsten Fall Neuübersetzung für den neuen Zielprozessor (der kommt nie vor ☺)

## 7. Optimierung mit dem Compiler

Alle Compiler haben aufwendig Codeoptimierungen eingebaut

- Manche sind unabhängig vom Zielprozessor
- Manche sind genau auf Befehlssätze, Register usw. zugeschnitten

Der Programmierer kann per Optionen verschiedene Optimierungsstufen auswählen

- Weiß dann mehr oder weniger, was passiert. Eher weniger

## Optimierung mit dem Compiler (2)

### Optimierungsoptionen für den GNU-C-Compiler `gcc`

-O0

führt keine Optimierungen durch

-O1

der Compiler versucht, Codegröße und Programmlaufzeit zu verringern, ohne die Übersetzungszeit wesentlich zu erhöhen

-O2

mehr Optimierungen aber kein Inlining, kein Loop Unrolling  
Code wird schneller, Übersetzungszeit steigt

-O3

Inlining wird auch aktiviert

-Os

Wie -O2, aber ohne Optimierungen, die den Code vergrößern

# Optimierung mit dem Compiler (3)

## Zwei Beispiele für Optimierungsverfahren

- Inlining von Funktionen
  - Der Sourcecode einer Funktion wird übersetzt und überall da direkt eingebaut, wo die Funktion aufgerufen wird
  - Zeitaufwendige Sprünge entfallen
  - Maschinencode wird länger
- Loop Unrolling
  - Wenn eine Schleife z.B. 10x durchlaufen wird, dann wird der Code 10x hintereinander abgelegt
  - Zeitaufwendige Sprünge entfallen
  - Maschinencode wird länger

## Optimierung mit dem Compiler (4)

### Wann wähle ich welche Stufe?

- Phase der Fehlersuche: immer mit -O0
  - Ansonsten sind die Umbauten im Code für die Fehlersuche hinderlich, weil Code umgestellt, zum Teil eliminiert wird usw.
  - Eine eindeutige Zuordnung zu den Zeilen des Quellcodes ist dann nicht mehr möglich
- Phase des Profiling: ohne Funktionen-Inlining
  - Nicht alle Level sind mit der Funktionsweise des gewählten Profiler kompatibel
  - Im Einzelfall das Handbuch lesen
- Phase des Produktionsbetriebs z.B. mit -O3
  - Manchmal treten aber Fehler auf, die aus einem komplexen Zusammenspiel zwischen maschinennaher Programmierung und Compileroptimierung entstehen
  - Dann den Optimierungslevel heruntersetzen

## **8. Fazit**

Es kann nur in Zusammenarbeit der Disziplinen eine Verbesserung erzielt werden

Anwendungswissenschaftler – Informatiker – Mathematiker

Viel Wissen für eine optimale Optimierung nötig

- Der Einzelne weiß dazu meist zu wenig
- Ist aber keine Entschuldigung, jegliches Wissen abzuweisen

Aufwandsabschätzung

- Was nützt es mir bei meiner Abschlussarbeit?
- Was kostet mich das?
- Was kostet es in der Folge Dritte?

## To-Do-Liste

### Was muss ich wissen?

- Grundlagen der Komplexität bzgl. Zeit- und Speicherbedarf
- Wichtige Datenstrukturen und wichtige Algorithmen
- Kenntnis über das Ausmessen von Programmen
- Kenntnis über wichtige Aspekte der Compileroptimierung

### Was muss ich tun?

- Ein Buch zu „Algorithmen und Datenstrukturen“ besorgen und nach nützlichen Konzepten durchsehen
- `time` und `gprof` ausprobieren und einüben
- Programme regelmäßig bzgl. Ihrer Leistung analysieren und wichtige Einsichten aufschreiben
- Diskussion mit anderen Entwicklern über dieses Thema

# Optimierung sequentieller Programme

## Zusammenfassung

- Es gibt keine systematischen Ansätze zur Optimierung von sequentiellem Code
- Die Optimierungen finden auf der Ebene der Mathematik, der Programmierung und des Compilers statt
- Die Optimierungspotentiale sind da durchwegs sehr hoch
- Die konkrete Wahl der Programmiersprache ist weniger wichtig (solange es eine Übersetzer-Sprache ist)
- Mit der Komplexitätstheorie schätzt man Laufzeiten und Speicherbedürfnisse von Algorithmen ab
- Zur konkreten Programmanalyse gibt es bei Linux verschiedene Werkzeuge
- Optimierungen der Mathematik können die Laufzeit deutlich verbessern
- Optimierungen bei der Programmierung können die Laufzeit deutlich verbessern
- Optimierungen mit dem Compiler können die Laufzeit deutlich verbessern

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

246



## **Optimierung sequentieller Programme**

### **Die wichtigsten Fragen**

- In welchen Fällen versuche ich, die Leistung zu steigern?
- Auf welchen Ebenen setzt eine Optimierung an?
- Wie sind hier die Optimierungspotentiale?
- Welche Kenntnisse muss ich haben?
- Wie schätze ich die theoretische Leistungsfähigkeit ab?
- Wie schätze ich die praktische Leistungsfähigkeit ab?
- Wie optimiere ich die Mathematik?
- Wie optimiere ich auf der Programmebene?
- Wie nutze ich die Compileroptimierungen?
- Was muss ich alles lernen?



# Fehlersuche

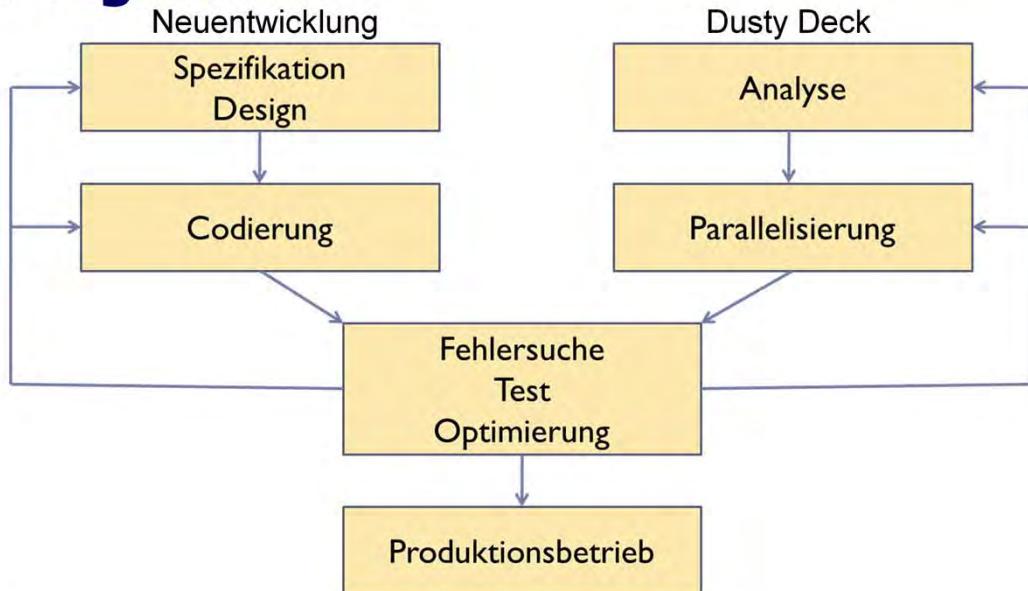
1. Entwicklungszyklus paralleler Programme
2. Fehlersuche
3. Häufige Fehlerquellen
4. Problemstellungen
5. Werkzeugunterstützung
6. Offline-Werkzeuge
7. Laufzeit-Debugger
8. Konzepte paralleler Debugger
9. Deterministische Ablaufkontrolle

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

248

# 1. Entwicklungszyklus paralleler Programme



25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

249

Siehe:

- [https://en.wikipedia.org/wiki/Legacy\\_code](https://en.wikipedia.org/wiki/Legacy_code)

Als Dusty-Deck (von 'deck', Lochkartenstapel), oder Legacy-Programme bezeichnet man solche, die schon sehr alt sind, aber immer noch verwendet werden. Die Programmierer sind längst woanders und niemand kennt sich mit dem Code aus. Bei Parallelisierungen gehören die allermeisten Projekte in diese Kategorie.

## **2. Fehlersuche (Debugging)**

Aufspüren von Fehlerzuständen und die Beseitigung ihrer Ursachen

### 4 Schritte

- Test, Regressionstest
- Erkennen der Fehlerwirkung
- Schließen auf die Fehlerursache
- Beseitigen der Fehlerursache

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

250

Siehe:

- [https://en.wikipedia.org/wiki/Regression\\_test](https://en.wikipedia.org/wiki/Regression_test)

# Debugging?

9/9

0800 Aikam started  
1000 " stopped - aikam ✓  
13'00 (032) MP - MC { 1.2700 9.037 847 025  
033 PRO 2 1.3047645,5 (23) 9.037 846 995 convert  
convert 2.13067645  
Relays 6-2 in 033 failed several speed test  
in relay " 11.00 test.  
Relays changed  
1100 Started Cosine Tape (Sine check)  
1525 Started Multi Adder Test.  
1545 Relay #70 Panel F  
  
First actual case of bug being found.  
1600 aikam started.  
1700 closed down.

Relay 3145  
Relay 3370

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

251

Quelle:

- <https://en.wikipedia.org/wiki/File:H96566k.jpg>

**The First "Computer Bug"** Moth found trapped between points at Relay # 70, Panel F, of the Mark II Aiken Relay Calculator while it was being tested at Harvard University, 9 September 1947. The operators affixed the moth to the computer log, with the entry: "First actual case of bug being found". They put out the word that they had "debugged" the machine, thus introducing the term "debugging a computer program".

## Beispiel

```
foo (a, b, x, &result);
/* Ursache: `&' vergessen
Compiler-Warning: pointer from
integer without a cast */

void foo (int a, int b, int *x,
          int *result)
{  *x = a+b;
/* segmentation violation */
}
```

Der Fehler liegt in der falschen Übergabe des dritten Parameters: es müsste die Adresse von x übergeben werden, nicht der Wert von x.

### **3. Häufigste Fehlerquellen**

#### Sequentielle Programmierung

- Schnittstellenprobleme (Typen, Zeiger auf Parameter, ...)
- Zeiger und dynamische Speicherverwaltung
- Logische und arithmetische Fehler

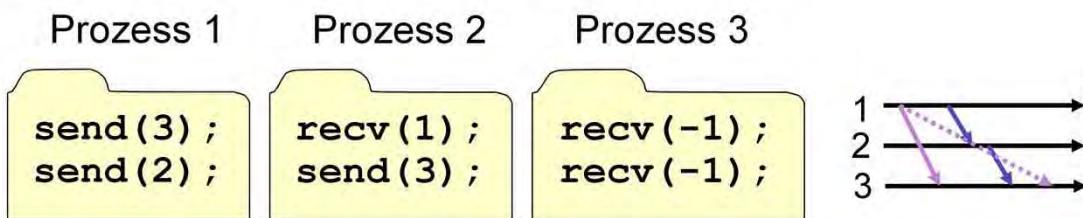
#### Parallele Programme

- Kommunikationsfehler (Protokolle)
- Überholvorgänge (*races*)
- Verklemmungen (*deadlocks*)

## Häufigste Fehlerquelle: Überholtvorgänge

Definition: Ein Überholtvorgang entsteht durch unsynchronisierte, modifizierende Zugriffe auf gemeinsame Objekte (Adressbereiche, Nachrichtenpuffer)

Beispiel:

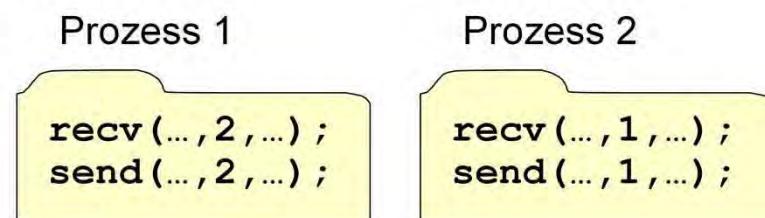


Konsequenz: Nichtdeterminismus,  
Nichtreproduzierbarkeit (schwer feststellbar)

## Häufigste Fehlerquelle: Verklemmung

Definition: Bei einer Verklemmung warten Prozesse blockierend auf Ereignisse anderer Prozesse, die auch blockiert sind

Beispiel:



Konsequenz: Programm bleibt hängen (leicht feststellbar)

## 4. Problemstellungen

Zusätzlich zu den normalen Problemen der Fehlersuche

- Erkennen einer Fehlerwirkung
- Suchen der Fehlerursache
  - Nichtreproduzierbarkeit der Fehlerwirkung
  - Nichtdeterminismus der Programmausführung
  - Ursache: Zeitabhängigkeit **nach** der Fehlerursache
- Unübersichtlichkeit: viele Prozesse
- Physische Verteiltheit
- Dynamik: Knoten- und Prozessmengen variieren potentiell

# Erkennen einer Fehlerwirkung

## Normalerweise

- Zustand des Programms entspricht nicht der Spezifikation  
(am Ende / mittendrin)
- Vergleich mit Testdaten

## Bei parallelen Programmen

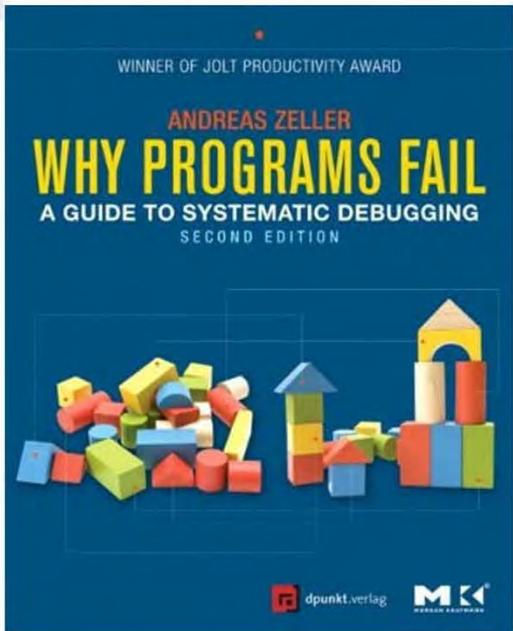
- Ergebnisse nicht nachrechenbar
- Ablauf abhängig von der Prozessorzahl
- Ablauf abhängig von zeitlichen Verhältnissen

→ Falsche Berechnungen schwer erkennbar

## Fehlertypen

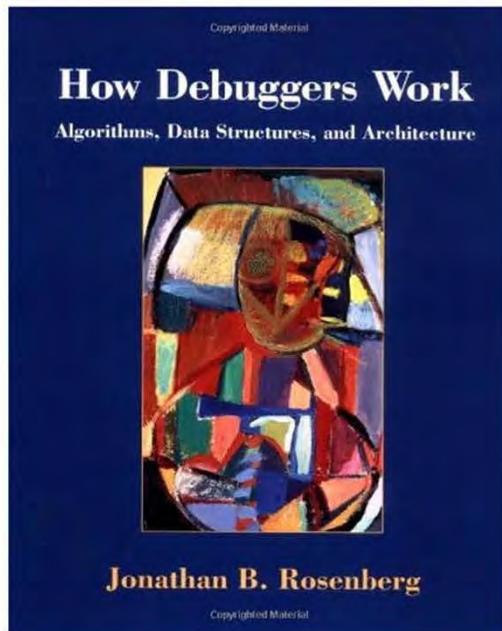
- Heisenbug
  - Verschwindet, wenn man versucht, ihn zu suchen
- Bohrbug
  - Beständiger, zuverlässiger Fehler (eher selten)
- Mandelbug
  - Hohe Komplexität lässt ihn chaotisch erscheinen
- Schroedinbug
  - Taucht erstmals auf, nachdem jemand den Programmcode gelesen hat und feststellte, dass das nie hat laufen können. Danach läuft es auch nicht mehr.

# Literatur



25.10.2016

Hochleistungsrechnen - © Thomas Ludwig



259

## 5. Werkzeugunterstützung

- Statische Analyse
- printf() und WRITE
- Debugger
  - Spurbasierte Werkzeuge (offline)
  - Laufzeit-Werkzeuge (online)
- Ablaufkontrolle und Sicherungspunkte

# Statische Analyse

Analyse des Programmtextes vor/zur Übersetzungszeit

- Sequentielle Aspekte
  - Strikte Typ- und Parameterprüfung
  - Erweiterte semantische Tests
  - Einsatz spezieller Werkzeuge (siehe Liste)
  - Gute ANSI-C-Compiler, Option –Wall (alle Warnungen)
- Parallele Aspekte
  - Erkennen möglicher Überholvorgänge
  - Prüfung auf Verklemmungsfreiheit
  - = Forschungsthemen (bisher ungelöst)

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

261

Siehe:

- [https://en.wikipedia.org/wiki/List\\_of\\_tools\\_for\\_static\\_code\\_analysis](https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis)

## printf() und WRITE

### **Die** Werkzeuge zur Fehlersuche schlechthin!

Bei parallelen Programmen aber:

- Zuordnung zu einzelnen Prozessen schwierig  
Zeichen-/zeilenweises Mischen möglich
- Bei Netzen: Umleitung in ein gemeinsames Fenster?!
- Sortierung oft unmöglich, da keine globale Zeit
- Korrekte kausale Ordnung der Ausgaben nicht gewährleistet

# Offline-/Online-Werkzeuge

Automatische Fehlerprüfung nach oder zur Laufzeit

Sequentielle Aspekte

- Dynamische Speicherverwaltung

Parallele Aspekte

- Parameterprüfung bei Programmierbibliothek
- *Race*-Erkennung (Forschungsthema)

Vorbereitung der Anwendung (Alternativen)

- Präprozessor und Neuübersetzung
- Binden mit speziell instrumentierter Bibliothek
- Instrumentierung der Binärdatei

## 6. Spurbasierte Werkzeuge (offline)

### Merkmale

- Aufzeichnung relevanter Ereignisse des Programmlaufs
- Betrachtung der Spur durch „Browser“  
offline und auch near-online möglich
- Im Prinzip: automatisiertes `printf()`

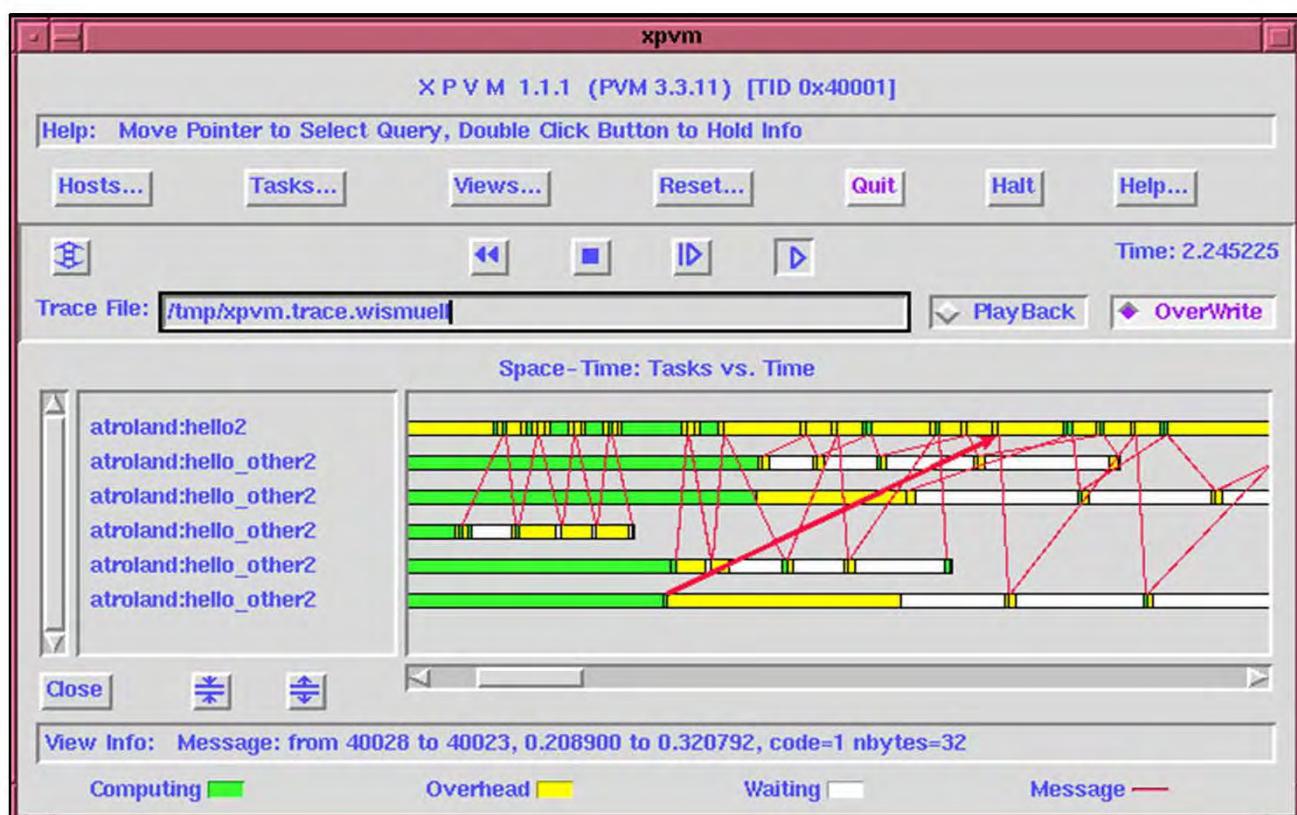
### Aufgezeichnete Ereignisse

- Aufruf und Rückkehr der Funktionen der Programmierbibliothek  
Lokale Zeit, Dauer, Parameter
- Zum Teil auch benutzerdefinierte Ereignisse möglich

# Spurbasierte Werkzeuge...

## Darstellungsarten

- Raum-Zeit-Diagramme, Gannt-Diagramme
  - Darstellung einzelner Prozeßzustände
  - Knoten- und/oder prozeßorientierte Darstellung
  - Gut: globaler Überblick
  - Schlecht: Globale Ordnung meist trügerisch
- Folge von Schnappschüssen
  - Darstellung des globalen Zustands zu bestimmten Zeiten



# Spurbasierte Werkzeuge...

## Steuerung

- VCR-ähnliche Elemente: Start, Stop, Vor, Zurück, Einzelschritt
- Meist Auswahl relevanter Knoten und Prozesse

## Vorbereitung

- Präprozessor und Neuübersetzung
- Binden mit instrumentierter Bibliothek
- Laufzeitoption der Programmierbibliothek

## Bewertung

- Für globalen Überblick und zur Überwachung der Kommunikation

## 7. Laufzeit-Debugger (online)

### Vorgehen

- Anhalten des Programms an interessanten Stellen
- Inspizieren des Programmzustandes
- Fortsetzen (oder Neustart) des Programms
- Schrittweise Programmabarbeitung

### Bei erkanntem Fehler

- Hypothese zur Fehlerursache
- Neustart des Programms und Überprüfen der Hypothese

# Laufzeit-Debugger...

## Typischer Funktionsumfang

- Anhalten des Programms  
Bedingt und/oder unbedingt
- Inspizieren des Programmzustandes  
Prozeduraufrufkeller, Parameter, Variablen
- Modifikation des Programmzustandes  
Setzen von Variablen, Veränderung des Codes(!)
- Ausführungskontrolle
  - Start und Stop
  - Einzelschritt (Anweisungen, Prozeduren)

## 8. Konzepte paralleler Debugger

### Eigenschaften paralleler Programme

- Mehrere Aktivitätsträger  
Prozesse, evtl. Threads; evtl. mehrere Binärformate
- Dynamik  
Zur Laufzeit Änderungen der Knoten, Prozesse, ...
- Interaktion  
Kommunikation und Synchronisation zwischen Prozessen
- Verteiltheit  
Verteilte Information; kein globaler Systemzustand

Berücksichtigung dieser Eigenschaften sehr unterschiedlich

Kein Standard auf dem Gebiet in Sicht

Es gibt zur Zeit zwei gebräuchliche parallele Debugger:

- The Distributed Debugging Tool DDT der Firma Allinea
- Totalview der Firma Rogue Wave.

# Umgang mit mehreren Prozessen/Threads

Zwei Methoden:

Fenstertechnik und Prozessmengen

- Pro Prozess ein Fenster
  - Unabhängiger sequentieller Debugger pro Fenster
  - Leicht zu entwickeln; schwierige Benutzung bei vielen Prozessen
  - => (v.a.) für funktionsparallele Programme
- Ein einziges Fenster für alle Prozesse
  - Auswahl eines Prozesses zur Fehlersuche
  - Kommandos für Prozessmengen
  - => (v.a) für datenparallele Programme
- Mehrere Fenster für beliebige Teilmengen von Prozessen
  - => für beliebige Programme (DETOP)

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

271

DETOP: Debugging Tool für Parallel Programs, Eigenentwicklung an der TU München.

## Skalierbarkeit und Dynamik

Problem: Umgang mit höheren Prozessanzahlen

- Kommandos für Gruppen von Prozessen
- Zusammenfassen identischer Ergebnisse verschiedener Prozesse
- Einsatz geeigneter graphischer Darstellungen

Problem: Debugging dynamisch generierter Prozesse

- Stoppen aller neu erzeugten Prozesse; manuelle Auswahl
- Automatische Auswahl über Mustervergleich mit Zusatzinformation

## Interaktion

### Überwachung von Kommunikation und Synchronisation

- Möglich durch Haltepunkte auf Bibliotheksfunktionen
- Meist keine weitergehende Unterstützung, wie z.B.
  - Ausgabe wartender Prozesse
  - Status von Nachrichtenwarteschlangen
  - Haltepunkte auf Nachrichten
- Ausweg

Gleichzeitige Benutzung spurbasierter Werkzeuge (i.a. nur lesender Zugriff) und von Spezialwerkzeugen (message queue manager, mqm)

# Verteiltheit

Daten der Anwendung sind verteilt

- Spezialwerkzeuge liefern globale Sicht

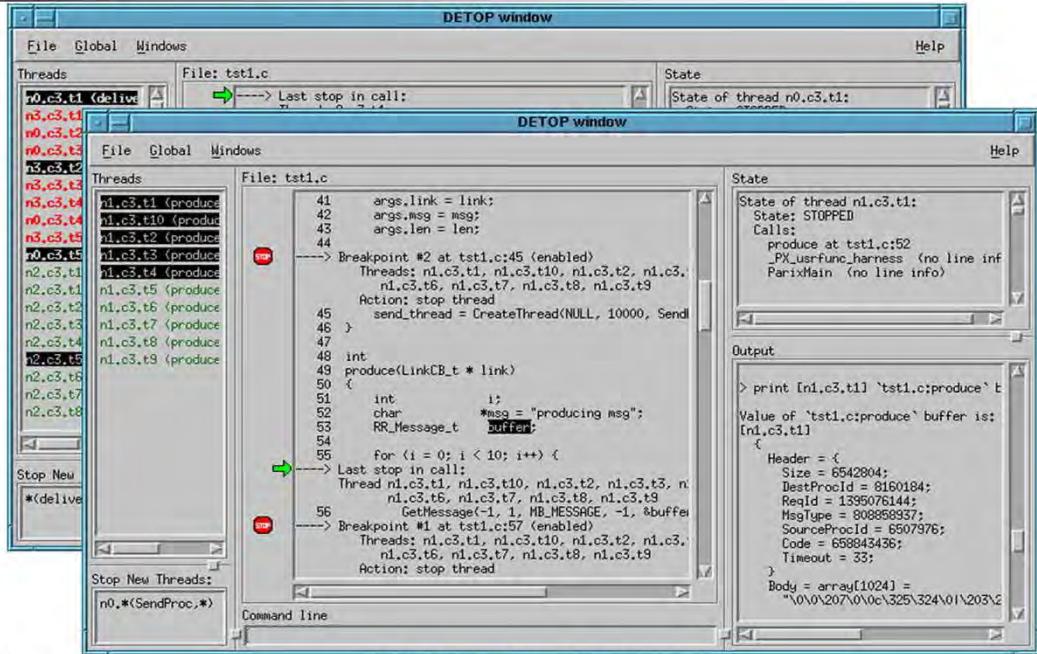
Gemeinsame Daten verteilter Prozesse

- Beispiele: MPI-Gruppen, gemeinsame Speichersegmente
- Problem: Einfrieren des Zustandes bei Erreichen des Haltepunktes
- Meist nur Anhalten eines Prozesses unterstützt
- Wenn globales Anhalten unterstützt, dann nie sofort(!)  
=> Zustandsveränderungen sind möglich

Globale Ereigniserkennung (Forschungsthema)

- Verknüpfung von Ereignissen in verschiedenen Prozessen
- Z.B. Ereignisse a und b sind kausal abhängig/unabhängig

# Beispiel DETOP (1993)



25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

275



## Beispiel Allinea DDT

### The Distributed Debugging Tool (DDT)

*"DDT, the Distributed Debugging Tool is a comprehensive graphical debugger for scalar, multi-threaded and large-scale parallel applications that are written in C, C++ and Fortran."* Allinea

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

276

Siehe: <http://www.allinea.com/>

"For multi-threaded or OpenMP development DDT allows threads to be controlled individually and collectively, with advanced capabilities to examine data across threads.

The Parallel Stack Viewer is a unique way to see the program state of all processes and threads at a glance - and developers can easily spot rogue processes or threads, and even define new control groups from it, meaning massively parallel programs are easy to manage.

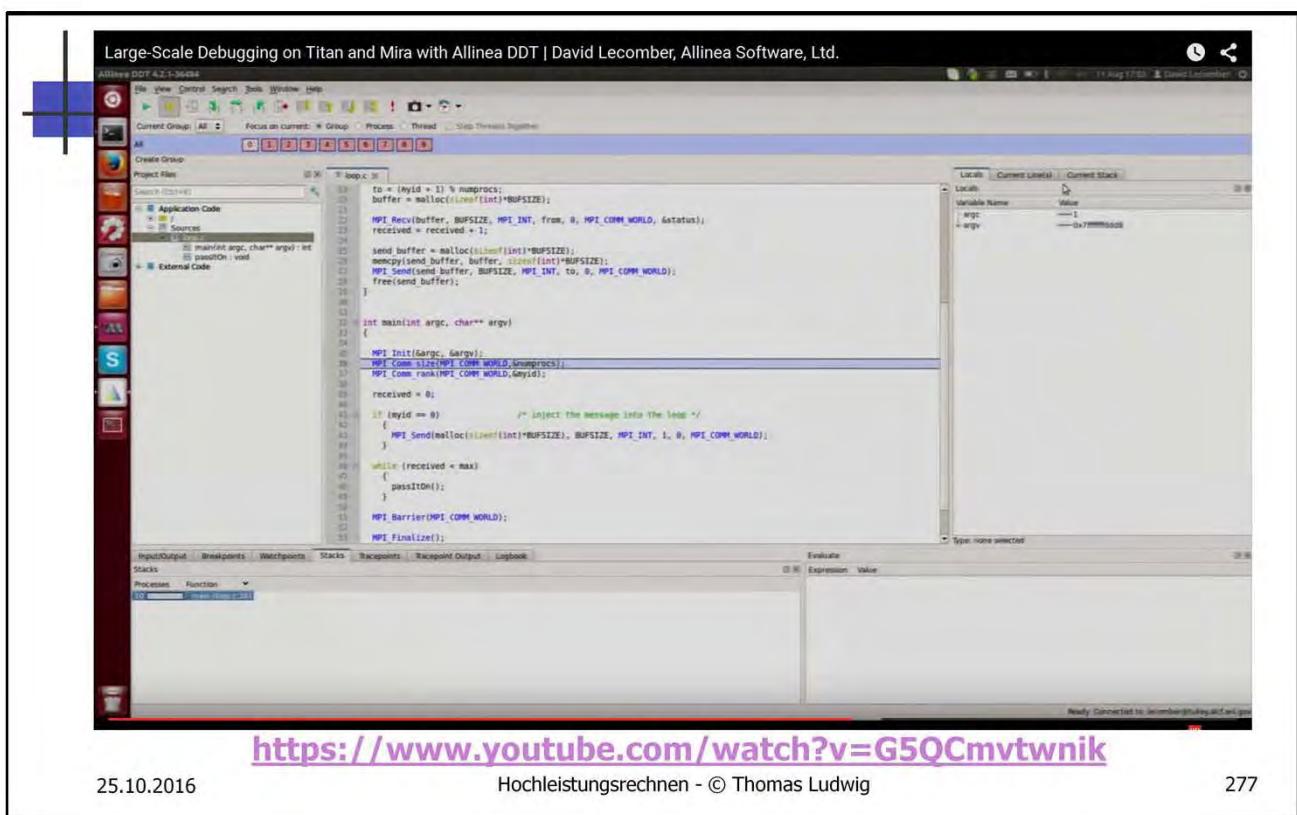
DDT's interface scales amazingly - providing the same clarity of information at thousands of processes as at a handful - highlighting commonality and differences with summary views and data comparison to focus your attention.

DDT is proven at scale on the most powerful systems - including debugging applications at over 200,000 cores simultaneously.

DDT's advanced memory debugging capability brings tremendous benefits to developers of scalar and parallel applications. DDT can find memory

leaks, and detect common memory usage errors before your program crashes.

With DDT, you can check a pointer is valid or find the stack when it was allocated - a fantastic boost for any developer. Reading or writing beyond the ends of allocated data can also be detected - instantly."



25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

277

# Übersicht

The screenshot shows a debugger interface with several windows:

- Top Bar:** Session, Control, Search, View, Help.
- Toolbar:** Includes icons for file operations, search, and execution control.
- Focus on current:** A dropdown menu with options: Current Group, All, Group, Process, Thread, Step Threads Together. The "All" option is selected.
- Task List:** Shows four tasks labeled 0, 1, 2, 3. Task 0 is highlighted with a red box.
- Code Editor:** Displays the MPI code for `mpi_hello.f90`. The code initializes MPI, prints the number of processes and rank, and finalizes MPI.
- Stack Arguments:** Shows the stack arguments for the current task, including `#1 mpi_hello () at mpi_hello.f90:4 (st)`.
- Stacks (All) Window:** Shows the call stack for all processes. It lists multiple entries for `-mpi_hello (mpi_hello.f90:4)`, indicating that each process runs the same MPI program.
- Evaluate Window:** An empty window for evaluating expressions.

Annotations in red text:

- Kontrolle aller Prozesse gemeinsam** (Control all processes together) points to the "Focus on current" dropdown.
- MPI-Job aus 4 Tasks** (MPI job consisting of 4 tasks) points to the task list.
- Alle Prozesse führen gleiches Programm aus** (All processes run the same program) points to the call stack window.
- MPI startet offenbar eigene Threads** (MPI apparently starts its own threads) points to the call stack window.

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

278

Erklärungsbedürftig sind noch: einzelne Kontrollelemente in zweiter Menüleiste (v.l.):

Fortfahren, Unterbrechen, Breakpoint setzen, Step-In, Step-Over, Until

# Breakpoint im Ozeanmodell MPIOM

Erste ausführbare Zeile in Funktion

Standardausgabe und -fehler aller Prozesse

Werte von Ausdrücken in aktueller Zeile

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

279

```

      91      INTEGER :: i_type
      92      INTEGER :: i_xup,iylo,j_size,j_size
      93
      94      REAL, INTENT(OUT) :: a0(:,:)
      95      CHARACTER (LEN=*) , INTENT(IN) :: ttt
      96      CHARACTER (LEN=*) , INTENT(IN) :: nm
      97
      98      INTEGER nm
      99
     100     INTEGER :: mu
     101     REAL :: mu
     102
     103     !-----#
     104     #ifdef _PROFILE
     105     CALL trace_start ('bounds_exch_2d_total', 2)
     106     CALL trace_start ('bounds_exch_2d_ov/ns', 3)
     107
     108 #endif
     109
     110 ! calculate field bounds incl. halos
     111 ! i_lo+1:i_hi+1
     112 ! j_lo+1:j_hi+1
     113 ! j_lo+1:j_hi+1
     114 ! i_lo+1:i_hi+1
     115 ! j_lo+1:j_hi+1
     116 ! j_lo+1:j_hi+1
     117
     118 !-----#
  
```

Gezeigt ist ein Lauf von MPIOM mit 16 Prozessen.

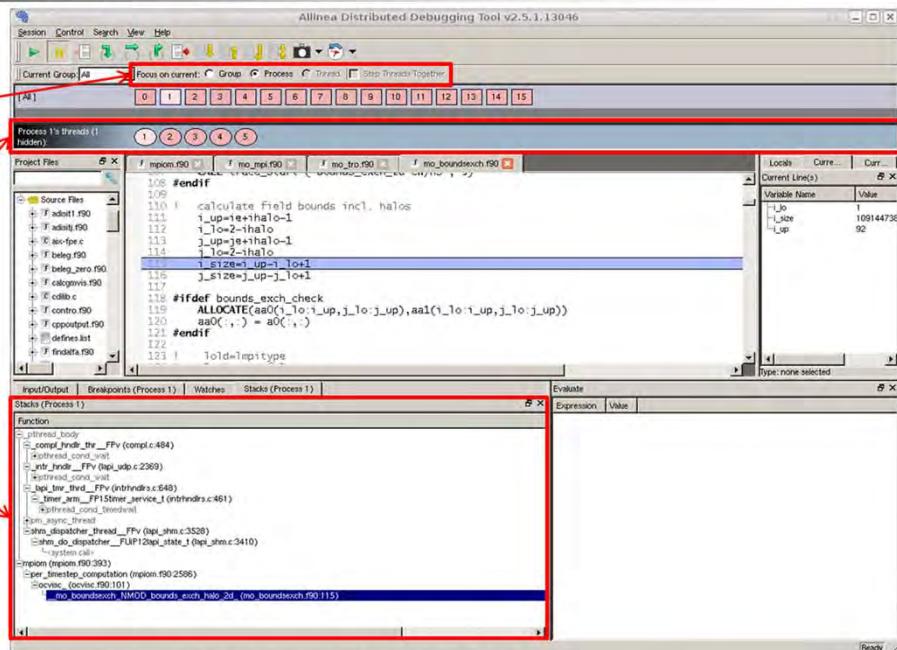
Gestoppt wurde durch einen Breakpoint auf die Kommunikationsfunktion boundsexch\_halo\_2d, es wird die erste ausführbare Zeile fokussiert, die erste Zeile der Funktionsdefinition ist Fortran-üblich bereits viele Zeilen vorher und deshalb nicht im Bild.

Wie am Dialog in der Bildmitte zu erkennen wartet der Debugger typischerweise einen kurzen Moment bis alle Prozesse in der fokussierten Gruppe am Breakpoint ankommen.

# Ausführung in einzelnen Prozess

Fokus auf  
einzelnen  
Prozess  
gewechselt

Thread- und  
Stackansicht  
wechseln mit



25.10.2016

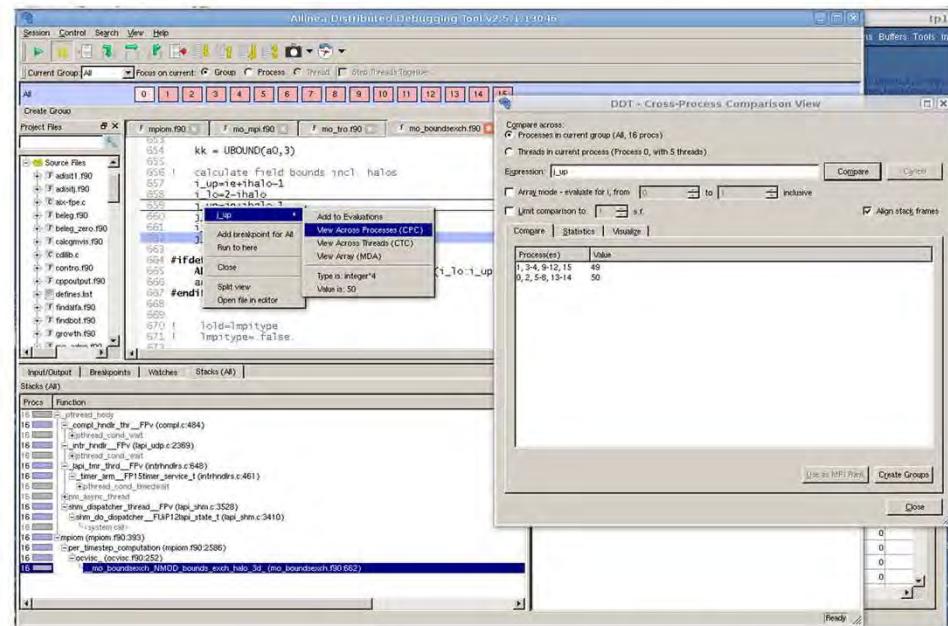
Hochleistungsrechnen - © Thomas Ludwig

280

## Nennenswert:

- Stackansicht wechselt auf den ausgewählten Prozess
- Fortlaufende Aktualisierung der Ausdrücke in aktueller Zeile
- Automatische Anzeige der auswählbaren Threads unter Prozessen in der Gruppe

# Variablenansicht



25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

281

## **8. Deterministische Ablaufkontrolle**

### Problem des Nichtdeterminismus

- Erzwingen einer deterministischen Abarbeitungsreihenfolge der Kommunikation  
Programm dadurch evtl. verlangsamt  
Varianten der Reihenfolgen systematisch testbar  
(Bei sequentiellen Programmen kein Problem!)

## Deterministische Ablaufkontrolle...

Funktionsweise eines Werkzeugs hierzu:

- Erster Programmlauf
  - Aufzeichnen der Reihenfolge des Eintreffens von Nachrichten bei Empfängern
- Weitere Programmläufe (*deterministic replay*)
  - Verwendung der Informationen aus dem ersten Programmlauf
  - Die Reihenfolge, wie Nachrichten beim Empfänger ankommen, wird gesteuert: zu früh eintreffende werden zurückgestellt

# Sicherungspunkte

## Problem der Zykluszeit

- Bei Fehler muss das Programm vom Anfang wiederholt werden, um nach der Fehlerursache zu suchen

## Vorgehensweise

- Zyklisches Erstellen von Sicherungspunkten
- Im Fehlerfall: Auswahl eines geeigneten Sicherungspunktes und Wiederanlauf des Programms von diesem Zeitpunkt aus.
- Evtl. gekoppelt mit Ablaufkontrolle



## **Fehlersuche**

### **Zusammenfassung**

- Unterscheide Fehlerursache und Fehlerwirkung
- Typische Fehler paralleler Programme
  - Überholvorgänge, Verklemmungen
- Probleme
  - Nichtreproduzierbarkeit, Nichtdeterminismus
  - Unübersichtlichkeit, physische Verteilheit, Dynamik
- Spurbasierte Werkzeuge für einen globalen Überblick und Prüfung der Kommunikation
- Haltepunktbasierte Debugger für Detailuntersuchungen
- Ablaufkontrolle beseitigt Nichtdeterminismus
- Sicherungspunkte verkürzen den Testzyklus

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

285



## **Fehlersuche**

### **Die wichtigsten Fragen**

- Welche Schritte umfasst die Fehlersuche?
- Was sind die häufigsten Fehlerquellen in Programmen?
- Was versteht man unter einer Verklemmung?
- Was versteht man unter einem Überholtvorgang?
- Welche Problemstellungen gibt es bei parallelen Programmen?
- Welche Kategorien der Werkzeugunterstützung unterscheiden wir?
- Wie stellen spurbasierte Werkzeuge typischerweise ihre Informationen dar?
- Welche Funktionen bietet ein Laufzeit-Debugger?
- Was ist deterministische Ablaufkontrolle und wie funktioniert sie?
- Was ist hierbei der Sinn von Sicherungspunkten?



# Kosten-Nutzen-Analyse

1. Kostenbetrachtungen
2. Nutzenbetrachtungen
3. Kosten-Nutzen-Analysen
4. Mögliche Optimierungen

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

287

# 1. Kostenbetrachtungen

TCO – total cost of ownership (Gesamtkosten)  
Summe aller Kosten über die Lebenszeit eines Systems

## Investitionskosten

- Computer-Hardware und -Software
- Rechenzentrumsgebäude
- ...

## Betriebskosten

- Wartungskosten
- Humanressourcen
- Stromkosten
- ...

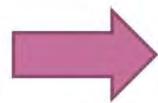
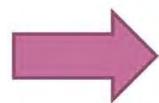
# Kosten in der Petascale-Ära

## Investitionskosten

- 2002: Earth Simulator (Yokohama): 600 M\$
- 2010: Tianhe-1A (Tjanin): 88 M\$
- 2011: K computer (Kobe): etwa 1 G\$
- 2011: Sequoia (Livermore): 250 M\$
- 2012: SuperMUC (Munich): 135 M€
  
- Beinhaltet zum Teil das RZ-Gebäude
- Beinhaltet zum Teil Stromkosten oder Kosten für ein Kraftwerk

# Kosten in der Petascale-Ära...

## Skalierbare Rechnersysteme



25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

290

## Kosten in der Petascale-Ära...

Betriebskosten für Strom

1 MW 24/7 für ein Jahr ergeben 8.760.000 kWh/a

0.11€/kWh ergeben 1.000.000 Euro pro Jahr

# Kosten in der Petascale-Ära...

## Betriebskosten für Strom

- 2002: Earth Simulator (Yokohama): 600 M\$
  - 3 MW → 2.5 M\$/a
- 2010: Tianhe-1A (Tianjin): 88 M\$
  - 4 MW → 3.5 M\$/a
- 2011: K computer (Kobe): etwa 1 G\$
  - 12 MW → 10 M\$/a
- 2011: Sequoia (Livermore): 250 M\$
  - 8 MW → 7 M\$/a
- 2012: SuperMUC (Munich): 135 M€
  - 3 MW → 5 M€/a

# Kosten in der Exascale-Ära

## Forschungs- und Entwicklungskosten

- Zahlreiche Exascale-Programme zum Bau eines Exaflops-Computers mit einem Exabyte-Speichersystem
- USA, Japan, Europa, China, Russland
  - Einige Milliarden Investitionen in F&E

## Geschätzte Investitionskosten

- Erster EFLOPS-Computer: 500-1000 M€

## Geforderte Betriebskostengrenze Strom

- 20 MW → 20 M€/a



## TCO für Klimaforschung am DKRZ

Insgesamt ca. 16 M€/a

- 8 M€ für Hardware
- 2,5 M€ für elektrischen Strom
- 3 M€ für Humanressourcen (Brainware)
- 1 M€ für das Gebäude (bei 25 Jahren Abschreibung)
- 0,5 M€ für Magnetbänder
- ...

# Energiekosten für DKRZ-Klimaforschung

## 5. IPCC-Statusbericht

- Deutscher Anteil nutzt ca. 30 Mio. Prozessorkernstunden
- DKRZ stellte 60 Mio. Prozessorkernstunden pro Jahr bereit
  - IBM Power6 „Blizzard“ mit 8.500 Rechnerkernen
- D.h. Energiekosten des deutschen Beitrags zum  
5. IPCC-Bericht **ca. 1 M€**
  - **9.000.000 kWh zur Lösung** mit DKRZs Rechnersystem
  - 4.500 Tonnen CO<sub>2</sub> mit normalem deutschen Strom

Klimaforscher sollten den Klimawandel prognostizieren ...  
... nicht produzieren!



## Kollateralschäden durch HPC

### Stromverbrauch

1 MW 24/7 für ein Jahr ergibt 8.760.000 kWh/a

20 MW 24/7 für ein Jahr ergibt 175.200.000 kWh/a



## Clean Energy



[Contact Us](#)

Search:  All EPA  This Area

You are here: [EPA Home](#) » [Climate Change](#) » [Clean Energy](#) » [Clean Energy Resources](#) » Greenhouse Gas Equivalencies Calculator

[Clean Energy Home](#)

[Basic Information](#)

[Energy and You](#)

[Clean Energy Programs](#)

[Clean Energy Resources](#)

[Site Map](#)

## Greenhouse Gas Equivalencies Calculator

UPDATED May 2011. New NYUP sub region and national average non-baselload emissions rates updated. See the [revision history page](#) for more details.

Did you ever wonder what reducing carbon dioxide (CO<sub>2</sub>) emissions by 1 million metric tons means in everyday terms? The greenhouse gas equivalencies calculator can help you understand just that, translating abstract measurements into concrete terms you can understand, such as "equivalent to avoiding the carbon dioxide emissions of 183,000 cars annually."

This calculator may be useful in communicating your greenhouse gas reduction strategy, reduction targets, or other initiatives aimed at reducing greenhouse gas emissions.

### Other Calculators

There are a number of other web-based calculators that can estimate greenhouse gas emission reductions for

- individuals and households
- waste, and
- transportation.

For basic information and details on greenhouse gas emissions, visit the Emissions section of [EPA's climate change site](#). 297

175200000

kilowatt-hours of electricity ▾

Calculate Equivalent

[? Click Here for Calculations and References](#)

## Option 2: If You Already Know the Quantity of Emissions

If you have already estimated the quantity of emissions (e.g., metric tons of carbon dioxide or CO<sub>2</sub> equivalent), input the amount of emissions and select the appropriate units for the corresponding gas.

Amount

Unit

Gas

120,810

Metric Tons ▾

CO<sub>2</sub>

- Carbon Dioxide or CO<sub>2</sub> Equivalent\*

1 kWh entspricht 0,69 kg CO<sub>2</sub>

## Equivalency Results

Click on the question mark ? link to read the explanation of that particular calculation

The information you entered above is equivalent to one of the following statements:

Annual greenhouse gas emissions from  passenger vehicles [?](#) ([click calculation](#))

CO<sub>2</sub> emissions from  gallons of gasoline consumed [?](#)

CO<sub>2</sub> emissions from  barrels of oil consumed [?](#)

CO<sub>2</sub> emissions from  tanker trucks' worth of gasoline [?](#)

CO<sub>2</sub> emissions from the *electricity* use of  homes for one year [?](#)

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

299

## 2. Nutzenbetrachtungen

Hochleistungsrechnen erweitert Experiment und Theorie

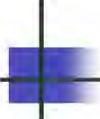
Numerische Simulation – die Dritte Säule

- Numerische Simulation als Mittel der Erkenntnisgewinnung
- Unabdingbar für moderne Wissenschaft und Technik

HPC ermöglicht **wettbewerbsfähige** Wissenschaft und Technik

## HPC und Wissenschaft

- Klima-/Erdsystemforschung
  - Verständnis der Wolkenbildung und Niederschläge
- Lebenswissenschaften
  - Verständnis des Gehirns und seine Simulation
  - Verständnis der Gene usw.
- Physik
  - Verständnis des Universums
  - Verständnis der kleinsten Bausteine
- vieles mehr ...

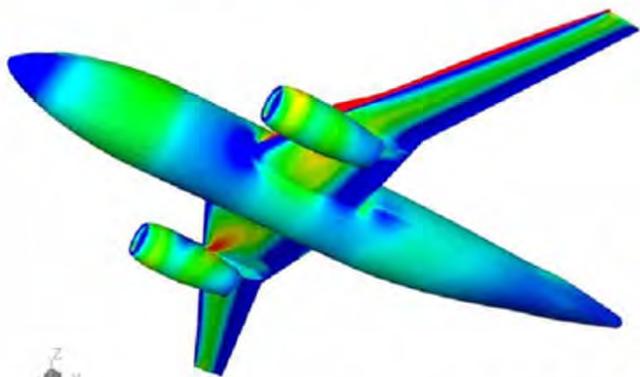


## HPC und Technik

- Automobilbau
  - Entwicklung effizienter Motoren
  - Optimierung von Reifen
- Flugzeugbau
  - Entwicklung sicherer und effizienter Flugzeuge
- Öl- und Gasindustrie
  - Erschließung neuer Reservoirs
- vieles mehr ...

# HPC und Technik...

Zusammenarbeit von Boing und ORNL  
(cf. <http://hpc4energy.org/hpc-road-map/success-stories/boeing/>)



25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

303

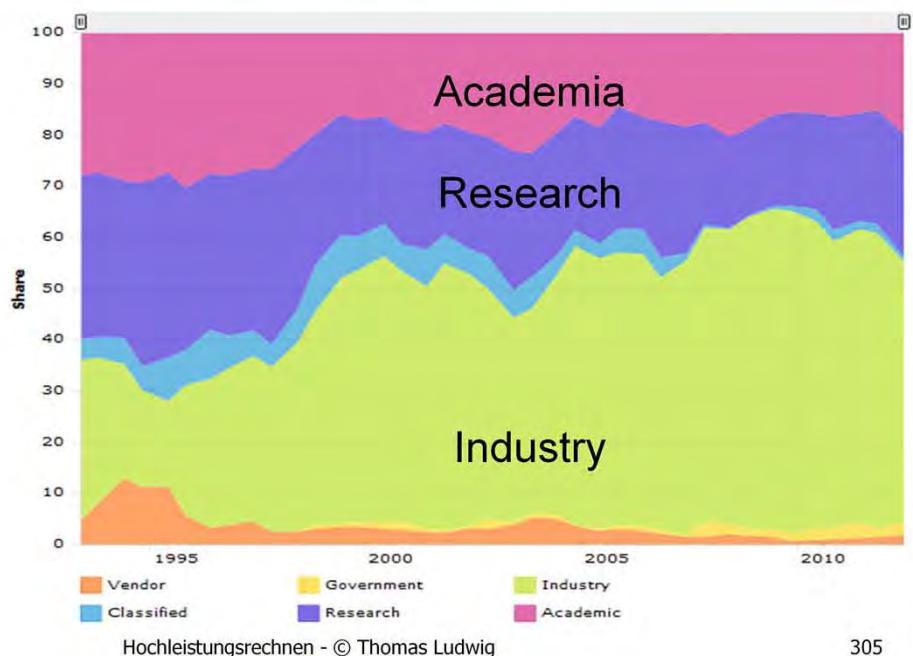
### Flugzeugdesign bei Boeing

- Modelliere Aeroelastizität
- Leichtere Verbundmaterialien für bessere Flügel
- 11 physikalische Flügeldesigns für 787 Dreamliner
  - Anstelle von 77 Flügeldesigns für 767
  - Konstruktion realer Testflügel deutlich verringert
  - Erhebliche Kosteneinsparungen!

# HPC in Wissenschaft und Technik

TOP500  
June 2012

system  
share



25.10.2016

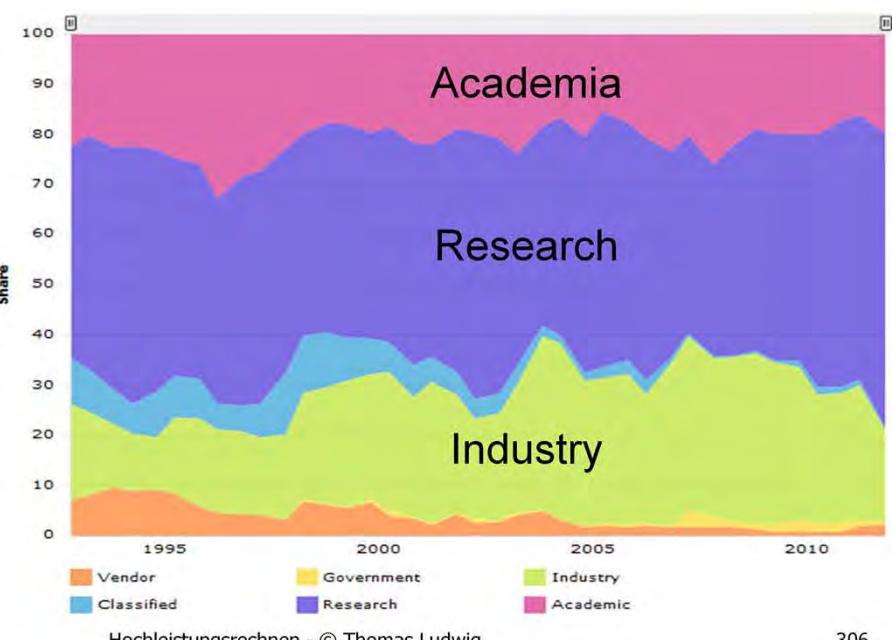
Hochleistungsrechnen - © Thomas Ludwig

305

# HPC in Wissenschaft und Technik...

TOP500  
June 2012

performance  
share



25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

306

### **3. Kosten-Nutzen-Analyse**

- Wie können wir die Kosten quantifizieren?
- Wie können wir den Nutzen quantifizieren?
- Wie definiert man eine Kosten-Nutzen-Relation?
  
- Was sind die möglichen Konsequenzen
  - ... für die Wissenschaft?
  - ... für die Industrie?
  - ... für die Gesellschaft?

## Beobachtung

Es gibt nicht viel systematische Forschung

Zur Beantwortung dieser Frage

tatsächlich: **nahezu keine Forschung**

Unser Ansatz hier:

- Betrachte praktische Beispiele
- Betrachte analytische Ansätze
- Betrachte mehr Beispiele ☺

## Kosten-Nutzen-Modell von Google



25.10.2016



Hochleistungsrechnen - © Thomas Ludwig

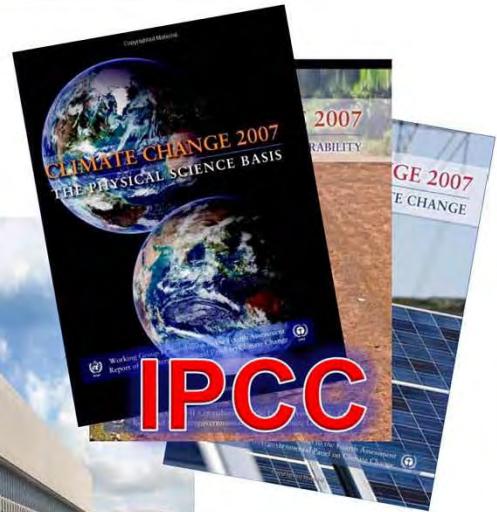
309

Verbrauch von 260MW im Jahr 2010. Der Profit war sehr hoch!

## Kosten-Nutzen-Modell des DKRZ



25.10.2016



310

## Kosten-Nutzen-Modell des DKRZ...

TCO des DKRZ pro Jahr: etwa **16 M€**

Veröffentlichungen pro Jahr: Annahme: 100

Mittlere Kosten pro Veröffentlichung: 160.000 €

+ Kosten für Wissenschaftler ☺

Das sind Steuergelder –  
die Gesellschaft hätte gerne einen Nutzen davon

# Erster analytischer Ansatz

Suzy Tichenor (Council of Competitiveness) and  
Albert Reuther (MIT Lincoln Laboratory)

Making the Business Case for High Performance Computing: A  
Benefit-Cost Analysis Methodology  
CTWatchQuarterly, November 2006

- Aufsichtsräte der U.S. Industrie sehen HPC nur als Kostenfaktor
- Versuche also, Nutzen und Kosten in Wissenschaft und Technik zu quantifizieren
- Überzeuge die Entscheidungsträger

## Erster analytischer Ansatz...

### Maße

- benefit-cost ratio BCR (bcr = benefit / cost)  
[also: BCR = ROI / TCO]
- internal rate of return IRR (IRR=BCR-1)
  
- Benötigt eine akkurate Datenbasis
- Auswertung lief über ein Jahr

ROI – return on investment, TCO – total cost of ownership

## Erster analytischer Ansatz...

For research oriented organizations

$$\text{productivity} = \frac{\text{(time saved by users on system)}}{\text{( time to parallelize )} + \text{( time to train )} + \text{( time to teach )} + \text{( time to administrate )} + \text{( system cost )}}$$

For industry environments

$$\text{productivity} = \frac{\sum \text{( Profit gained or maintained by project )}}{\text{( Cost of software )} + \text{( Training cost )} + \text{( Admin cost )} + \text{( System cost )}}$$

(cf. Jeremy Kepner, MIT Lincoln Laboratory, HPCS Productivity Team member)

## Beispielfall

MIT Lincoln Laboratory: 600-Prozessoren-Cluster, 200 Nutzer, Vollkostenjahresverdienst 200.000 \$

- 36.000 Stunden Benutzerzeit eingespart
- Zeitaufwand zur Parallelisierung von 200 Benutzerprogrammen: 6.200 Stunden
- Zeit für Training: 800 Stunden
- Systemverwaltung benötigt 2.000 Stunden pro Jahr
- HPC-System kostet 500.000 \$ (äquivalent zu 5.000 Mitarbeiterstunden)

## Beispielfall...

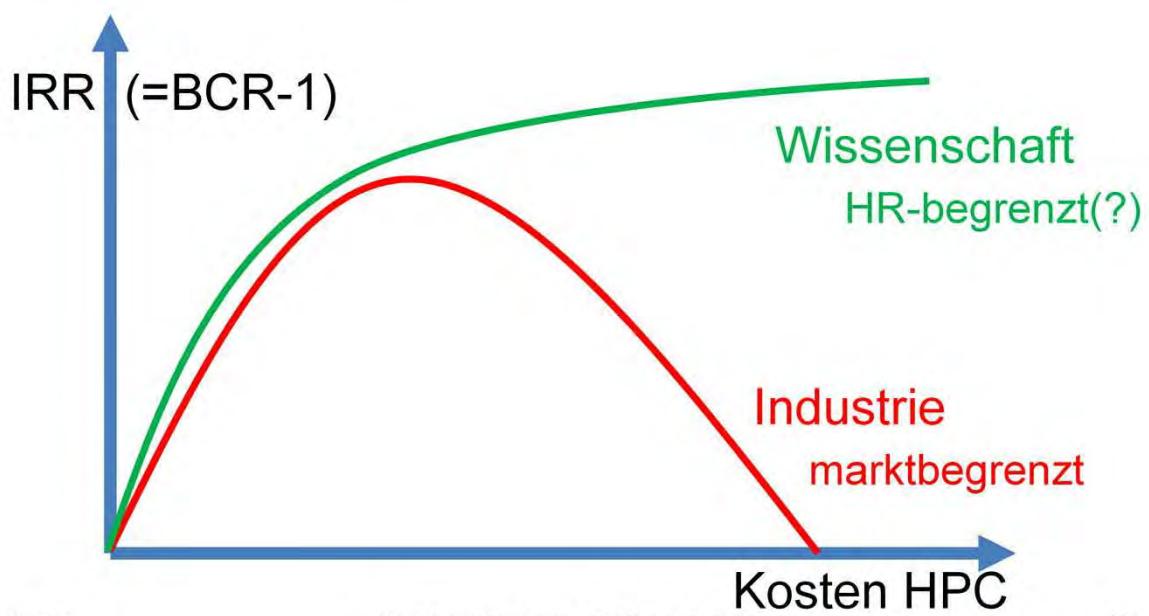
$$BCR = \frac{[Salary] \times 36000}{[Salary] \times (6200 + 800 + \cancel{22.8} + 2000 + 5000)} = \frac{36000}{14028} = 2.6,$$

$$IRR_{\text{year}} = BCR - 1 = 1.6 = 160\%.$$

Erspart Zeit für alle 200 Nutzer

Typischer Spruch eines Universitätskanzlers:  
*"Warum Zeit sparen für Wissenschaftler – die sind ja eh da"*

## BCR-Überlegung [Ludwig]



## Zweiter analytischer Ansatz

Amy Apon (University of Arkansas),  
Stan Ahalt (University of North Carolina) et al.

High Performance Computing Instrumentation and Research  
Productivity in U.S. Universities

Journal of Information Technology Impact, Vol. 10/2, 2010

- Forschungsinstitute mit leistungsfähigen HPC-Systemen sind erfolgreicher mit ihrer Forschung
- Ergebnisse sind statistisch und ökonomisch signifikant

## Zweiter analytischer Ansatz...

Apon/Ahalt studieren die folgenden Variablen

- dRankSum      Summe der abgeleiteten Ränge (500...1)
- Counts          #Listen mit dieser Institution
- NSF            Summe NSF-Förderung
- Pubs           Summe der Veröffentlichungen
- FF              Summe der Bundesförderung
- DOE            Summe der DOE-Förderung
- DOD            Summe der DOD-Förderung
- NIH            Summe der NIH-Förderung
- USNews        Rang in US News und World Report

# Korrelationsanalyse

	Counts	NSF	Pubs	All Fed	DOE	DOD	NIH	USNews
dRankSum	0.8198	0.6545	0.2643	0.2566	0.2339	0.1418	0.1194	-0.243
Counts		0.6746	0.4088	0.3601	0.3486	0.1931	0.2022	-0.339
NSF			0.7123	0.6542	0.5439	0.2685	0.4830	-0.540
Pubs				0.8665	0.4846	0.3960	0.8218	-0.588
All Fed					0.4695	0.6836	0.9149	-0.543
DOE						0.1959	0.3763	-0.384
DOD							0.4691	-0.252
NIH								-0.500

cf. slides by Apon, Ahalt on “Investment in High Performance Computing”

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

320

dRankSum und Counts haben starke Korrelation mit Höhe der NSF-Förderung (0,6545 and 0,6746), d.h. dies belegt die Hypothese

NSF-Förderung und Publikationen haben höhere Korrelationen mit Counts als mit dRankSum, d.h. eine konstante Investition bringt mehr als ein einzelner hoher Rang

Hohe negative Korrelation mit USNews, weil “1” der beste Platz ist; zeigt dass Publikation sehr wichtig sind

## Nebenbemerkung zur wissensch. Methode

Die Forschung von Apon/Ahal ist ein typisches Beispiel für datengetriebene Wissenschaft – noch nicht datenintensiv

- „The Fourth Paradigm“
- Kombiniere existierende Daten und gelange zu neuen Einsichten
- Würde ich Forschung auf zweiter Ebene nennen
- Wir werden viel mehr davon zu sehen bekommen



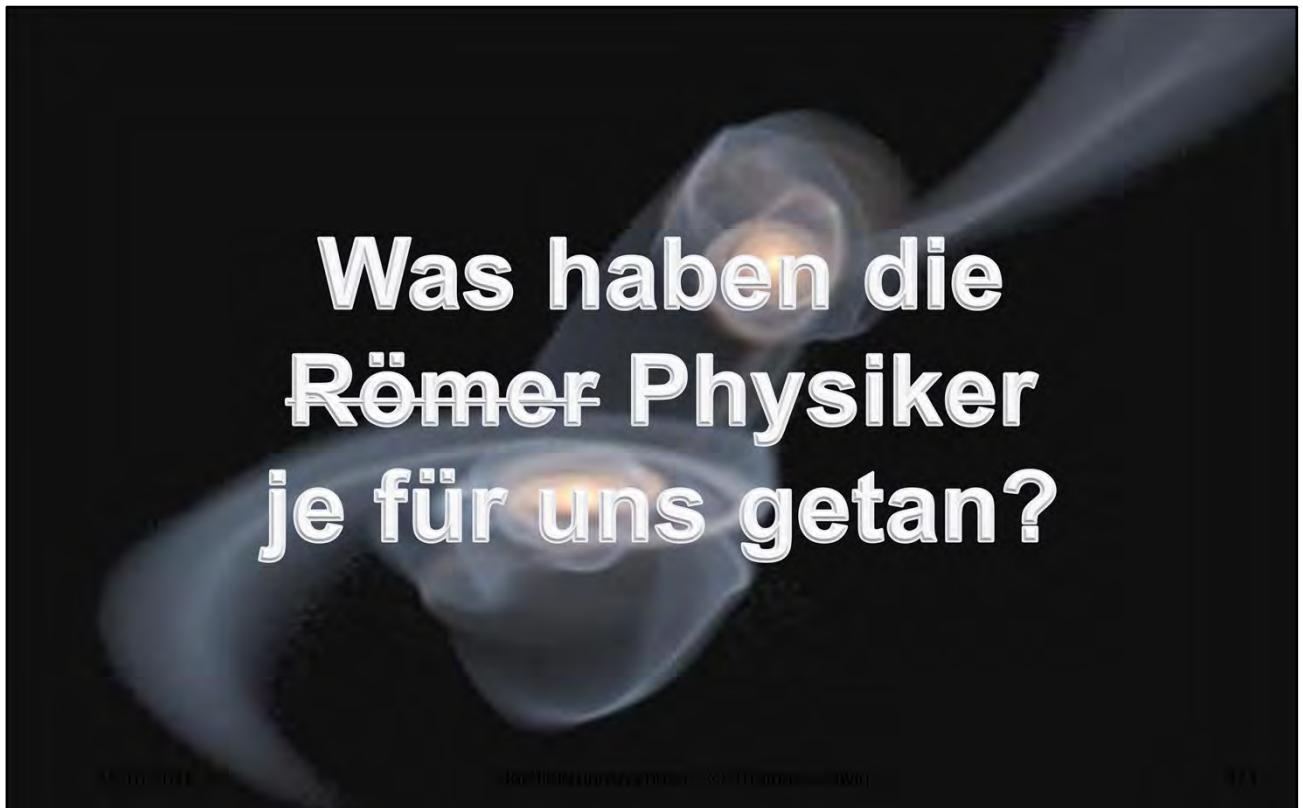
## Kosten-Nutzen-Analyse

Zwei weitere Beispiele...

## Higgs Boson alias Gottesteilchen

- Der Bau des Large Hadron Collider (LHC) kostete ca. 4 Milliarden Euro
- Stromkosten pro Jahr ca. 18 Millionen Euro
- Gesamtbudget zum Betrieb pro Jahr ca. 800 Millionen Euro

**Gesamtkosten zum Finden des Higgs Boson  
11 Milliarden Euro**



**Was haben die  
Römer Physiker  
je für uns getan?**

Galaxiekollisionen ☺

## 4. Mögliche Optimierungen

### Beobachtungen

- Meist entscheiden Politiker über Geldmittel
- Nutzen des Hochleistungsrechnens ist immer sehr hoch
  - Fast schon so nötig, wie ein Computer überhaupt

### Frage

- Können wir die finanziellen Ressourcen effizienter einsetzen, um einen höheren Nutzen zu haben?

# Wie erhöhen wir den BCR-Wert?

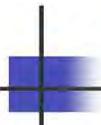
## Genereller Ansatz

- Erhöhe Nutzen und/oder **verringere Kosten**

## Im Detail

- Investiere in Humanressourcen
- Optimiere Programme (sequentielle und parallele)
- Verbessere die Leistung der Anwendungen
- Erhöhe somit die wissenschaftliche Produktivität

Hardware, Software, Brainware



## Wie messen wir das?

Im Detail: **verschiebe Ausgaben, verringere Kosten**

- Investiere in Humanressourcen
- Optimiere Programme (sequentielle und parallele)
  - Kosten gemessen in Gehalt pro Personenmonat
- Verbessere die Leistung der Anwendungen
  - Einsparungen durch Zeitgewinn und Energieeinsparung
- Erhöhe somit die wissenschaftliche Produktivität
  - Mehr Wissenschaft bei selbem Budget

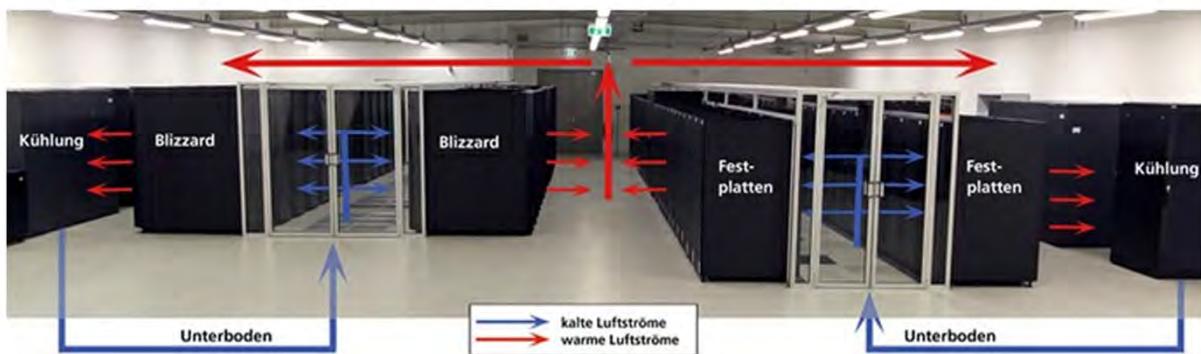
## Fiktives Beispiel Klimaforschung

### Beispiel: Produktionsläufe IPCC AR5

- Zur Erinnerung
  - Energiekosten des deutschen IPCC-Beitrags: ca. 1 M€
  - **9.000.000 kWh zur Lösung** mit DKRZ-System
  - 4.500.000 kg CO<sub>2</sub> mit regulärer deutscher Elektrizität
- Ansatz: optimiere Programm um 10% Laufzeit
  - Erspart 900.000 kWh
  - Erspart 100.000 € (ist ein Personenjahr)
  - Erspart 450 t CO<sub>2</sub>

## Reales Beispiel DKRZ (2012)

- Blizzard+Plattensystem 1,5 MW
- Kühlungssystem früher 0,45 MW
- Einhausung für ca. 100 T€ gekauft



- Spart im Jahr 100 T€ Strom und 425 t CO<sub>2</sub>

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

329

Siehe:

- <https://www.dkrz.de/about/kontakt/presse/aktuell/archiv-2013/energieeffizienz>

## Reales Beispiel HECToR

HECToR ist der UK National Supercomputing Service

- dCSE hat die Aufgabe, den Nutzern bei der Verbesserung ihrer Codes zu helfen
- Veröffentlichen viele Erfolgsgeschichten mit Quantifizierungen

Z.B.

- Ozeancode NEMO: höhere Geschwindigkeit und E/A
  - 6 PM Aufwand, spart 96 K£ pro Jahr
- Wichtiger Code der Materialwissenschaften CASTEP:  
4x schneller, 4x besser skalierbar
  - 8 PM Aufwand, spart 320 K£ - 480 K£ pro Jahr
- Plus: weniger Energieverbrauch pro Anwendung



## Schlussfolgerungen

### **Investiere in Personal !**

Wir benötigen mehr Spezialisten für HPC

- Code-Entwicklung
  - Höhere Qualität
- Code-Optimierung
  - Schneller, skalierbarer, energieeffizienter
- Rechenzentren
  - Energieeffizienter
- Viele andere Dinge...

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

331

Andere Dinge wie z.B. Visualisierungen, Datenmanagement, Langzeitarchivierung usw.



## Kosten-Nutzen-Analyse

### Zusammenfassung

- Die Kosten für HPC sind sehr hoch
- Der Nutzen von HPC in Wissenschaft und Technik ist gewaltig!
- Kosten lassen sich relativ einfach quantifizieren
- Nutzen lässt sich sehr schwer quantifizieren
- Nutzen-Kosten-Relation lässt sich optimieren, indem man die Kosten verringert und/oder den Nutzen steigert
- Die Investition in Humanressourcen (Brainware) kann nachweislich die Nutzen-Kosten-Relation verbessern

25.10.2016

Hochleistungsrechnen - © Thomas Ludwig

332

## **Kosten-Nutzen-Analyse**

### **Die wichtigsten Fragen**

- Benennen Sie ungefähre Kosten großer Rechnersysteme bzgl. Investitionen und Energieverbrauch
- Wie ist der ökologische Fussabdruck solcher Systeme?
- Mit welchen Maßen erfasst man wissenschaftliche Leistungen?
- Welche Zusammenhänge gibt es zwischen der Verfügbarkeit von HPC-Systemen und solcherart gemessenen Leistungswerten?
- Nennen sie Beispiele zur Verbesserung der Nutzen-Kosten-Relation bei HPC-basierter Wissenschaft
- Welche Änderung beim Einsatz finanzieller Mittel ist erfolgversprechend?