

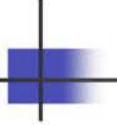
Hochleistungsrechnen

Vorlesung im Wintersemester 2021/22

Skriptversion 12.10.2021

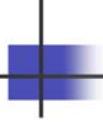
Prof. Dr. Thomas Ludwig

Universität Hamburg – Informatik – Wissenschaftliches Rechnen



Einleitung

1. Definition, Anwendungsbereiche, Systeme
2. Themenüberblick
3. Literatur und weitere Informationen
4. Leistungsnachweis
5. Arbeitsbereich Wissenschaftliches Rechnen



1. Definition, Anwendungsbereiche, Systeme

Definition in Wikipedia:

- **Hochleistungsrechnen** (englisch: *high-performance computing – HPC*) ist ein Bereich des computergestützten Rechnens. Er umfasst alle Rechenarbeiten, deren Bearbeitung einer hohen Rechenleistung oder Speicherkapazität bedarf.
- **Hochleistungsrechner** sind Rechnersysteme, die geeignet sind, Aufgaben des Hochleistungsrechnens zu bearbeiten.

Ein wichtiges Gebiet der Informatik mit sehr vielen Facetten und Bezügen zur Mathematik

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

3

Siehe: https://en.wikipedia.org/wiki/High_performance_computing

Der starke Bezug zur Mathematik ist dadurch gegeben, dass auf Hochleistungsrechnern größtenteils rechenintensive numerische Verfahren zum Einsatz kommen. Die computergestützte Simulation von Systemen beliebiger Natur steht dabei im Vordergrund.

Anwendungsbereiche

Anwendungsbereiche sind in allen Wissenschaften mit einem hohen Bedarf an Rechenleistung und Speichervolumen

Klassisch:

- Physik (Astronomie, Teilchenphysik, ...)
- Erdsystemforschung (Klima, Ozeanographie, ...)
- Bioinformatik (Stammbaumberechnungen, Pharmazie, ...)
- Ingenieurwissenschaften (Flugzeug-, Fahrzeugbau, ...)

Neu dazugekommen:

- Finanzwirtschaft
- Sozialwissenschaften (Simulation von Gesellschaften)



Ausprägungen

- Klein
 - Mehrere Prozessoren in einem Rechner und/oder Prozessorkerne in einem Prozessor
 - Einige hundert Euro
- Groß
 - Millionen von Prozessorkernen in einem Großrechner
 - Plattspeicher im Bereich dutzender Petabyte
 - Bandarchive im Bereich hunderter Petabyte
 - 5...500 Millionen Euro

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

5

- 1 Petabyte = 1024 Terabyte = 1024x1024 Gigabyte.
- Eine DVD fasst 4 Gigabyte.
- 2017: Eine aktuelle Festplatte speichert so 6 Terabyte, ein aktuelles LTO-6 Magnetband speichert 2,5 Terabyte.

Rechnersystem am DKRZ: bullx DLC 720



3.300+ Knoten, 100.000+ Prozessorkerne Intel Haswell/Broadwell

3,6 PFLOPS Rechenleistung, 240 TB Hauptspeicher

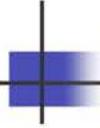
54 PB Festplatten, 400 PB Bandarchiv im Endausbau

Heißflüssigkeitskühlung für hohe Effizienz

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

6



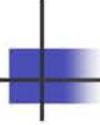
2. Themenüberblick

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 (20-63)
- Hardware-Architekturen (64-100)
- Die TOP500-Liste (101-157)
- Vernetzungskonzepte (158-192)
- Hochleistungs-Eingabe/Ausgabe (193-229)
- Betriebssystemaspekte (230-262)

Teil II: Programmierung

- Parallele Programmierung (263-308)
- Programmiermodell Nachrichtenaustausch (309-344)
- Parallele Eingabe/Ausgabe (345-376)
- Programmierung mit OpenMP (377-417)
- Programmierung mit Threads (418-453)
- Hybride Programmierung (454-505)
- Mathematische Bibliotheken (506-572)
- Reproduzierbarkeit (573-587)

Teil III: Programmierwerkzeuge

- Optimierung sequentieller Programme (588-629)
- Werkzeugarchitekturen (630-656)
- Fehlersuche (657-697)
- Leistungsanalyse (698-729)
- Leistungsoptimierung (730-757)
- Fehlertoleranz (758-800)

Teil IV: Allgemeine Fragestellungen

- Grid- und Cloud Computing (801-836)
- Kosten-Nutzen-Analyse (837-883)
- Rechnerbeschaffung (884-928)
- Visualisierung (Gastvortrag DKRZ)
- Maschinelles Lernen (Gastvortrag DKRZ)
- Die Geschichte des parallelen Rechnens (ohne Folien)
- Die Zukunft des parallelen Rechnens (ohne Folien)



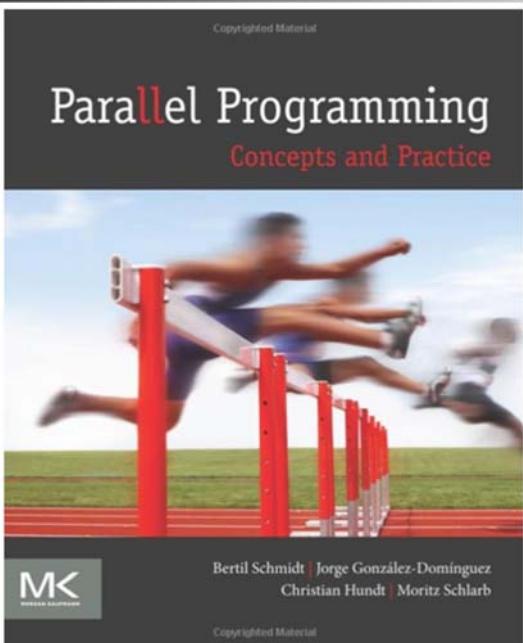
Themenabfolge

- Im Skript: nach Abstraktionsebenen
 - Wir starten bei der Hardware-Ebene
- In den Übungen: nur ausgewählte Themen!
 - Schwerpunkt parallele Programmierung mit den drei relevanten Programmiermodellen, Evaluierung der Leistung, Nutzung verschiedener Werkzeuge
- In der Vorlesung: nach Übungsthemen
 - Reihenfolge der Einzelthemen so, dass immer der Stoff besprochen wurde, bevor die Übungen dazu an die Reihe kommen

3. Literatur und weitere Informationen

- Materialienseite
 - [http://wr.informatik.uni-hamburg.de/teaching/wintersemester 2021 2022/hochleistungsrechnen](http://wr.informatik.uni-hamburg.de/teaching/wintersemester_2021_2022/hochleistungsrechnen)
 - Foliensätze, Übungsblätter usw.
- Mailingliste
 - <http://wr.informatik.uni-hamburg.de/listinfo/hr-2122>
 - Wichtige Neuigkeiten und Diskussionen zwischen den Teilnehmern

Literatur



Bertil Schmidt u.a.
Parallel Programming
Morgan Kaufmann, 2017
416 Seiten
Amazon: ca. 70 Euro
ISBN-10: 0128498900

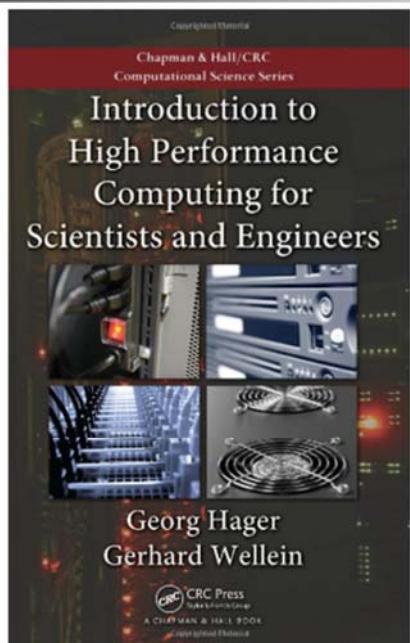
12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

14

Für Einsteiger und Fortgeschrittene.

Literatur...



**Georg Hager &
Gerhard Wellein
Introduction to High
Performance Computing
for Scientists and
Engineers**
Chapman & Hall, 2010
330 Seiten
Amazon: ca. 80 Euro
ISBN-10: 143981192X

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

15

Für Anfänger und Fortgeschrittene, die Hochleistungsrechnen einsetzen
wollen/müssen.

4. Leistungsnachweis

- Für alle Studierenden aus den verschiedenen Studiengängen:
 - 1. Termin: FR, 04. Feb. 2022, 09:30 - 11:30
 - 2. Termin: FR, 11. Mär. 2022, 09:30 - 11:30
- Übungen
 - Organisiert durch Anna Fuchs, Jannek Squar
 - Und: Niklas Schroeter, Georg von Bismarck
 - Termine
 - Online, DI und DO, noch festzulegen

5. Arbeitsbereich Wissensch. Rechnen

- Im Fachbereich für Informatik der Universität Hamburg seit Herbst 2009
- Leiter: Prof. Dr. Thomas Ludwig
 - Gleichzeitig Geschäftsführer des Deutschen Klimarechenzentrums
- Räumliche Unterbringung
 - Bundesstraße 45a
 - Keine Räume im Informatikum/Stellingen



Lehrveranstaltungen im WS 2021/22

Werden ggf. WS 2022/23 wiederholt!

- Seminar „Supercomputer: Forschung und Innovation“ (Anna Fuchs)
- Projekt „Parallele Systeme“ (Anna Fuchs)
- Projekt mit Seminar „Parallele Systeme für Fortgeschrittene“ (Anna Fuchs)

Beachten Sie auch das Angebot für SS 2021!

- Student Cluster Competition auf der Konferenz ISC 2022

Mitarbeit in der Arbeitsgruppe

Forschungsthemen

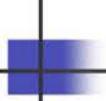
- Speicherung großer Datenmengen & Speichersysteme
- Anwendung in den Geowissenschaften

Wir betreuen:

- Bachelor-, Master-, Diplomarbeiten, Promotionen

Wir stellen potentiell ein:

- Studentische Hilfskräfte in der Arbeitsgruppe / am DKRZ
- Mitarbeiter in der Arbeitsgruppe / am DKRZ



Klimaforschung und Hochleistungsrechnen

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

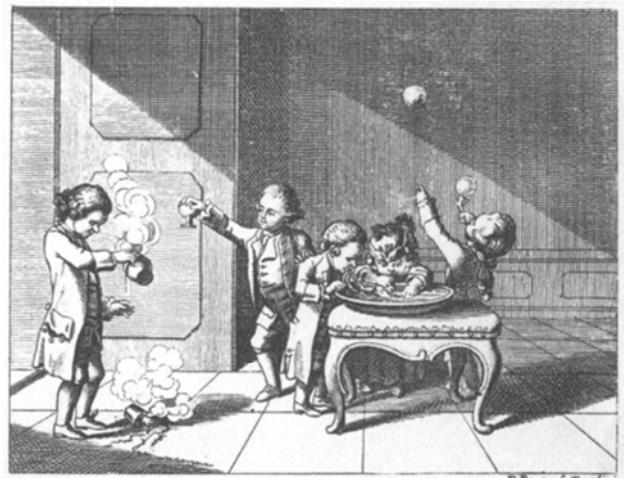


1. Der Erkenntnisgewinnungsprozess

- 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)



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

22

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)

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

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

23

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

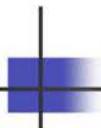
12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

24

Siehe

- http://de.wikipedia.org/wiki/Numerische_Simulation
- 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“

Wissen – Erkenntnis

Natürliche Sprache

Messdaten

Bildsprache

Formale Sprache

Mathematik

Ergebnisdaten aus
Nachverarbeitungen

Programmiersprache

Ergebnisdaten aus

Formale Sprache

Berechnungen

(Spezial)Rechner

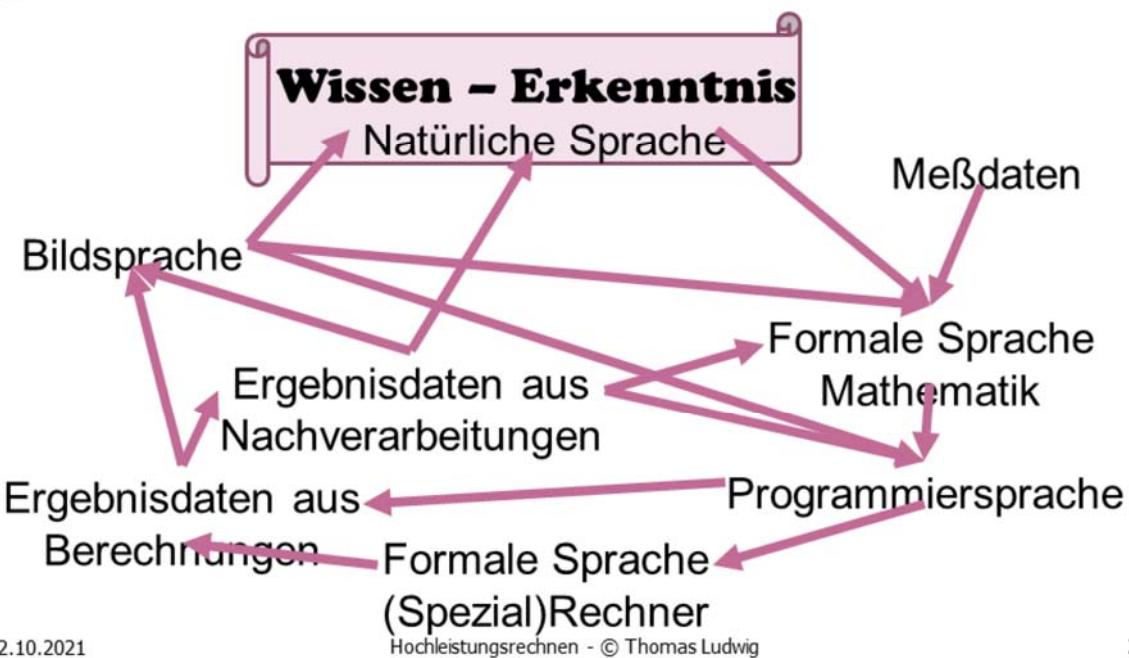
Hochleistungsrechnen - © Thomas Ludwig

12.10.2021



AUSBLENDEN

„Die Dialekte der Klimaforschung“



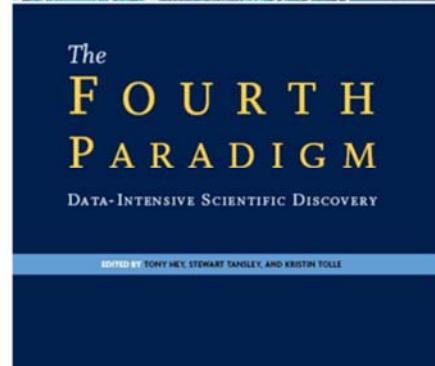
Siehe Unterlagen zum Workshop „Die Dialekte der Klimaforschung“
<https://www.dkrz.de/Klimaforschung/dkrz-und-klimaforschung/theorie/dialekte>

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."

Tony Hey et al., Microsoft Research, 2009



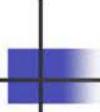
12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

29

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

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

12.10.2021

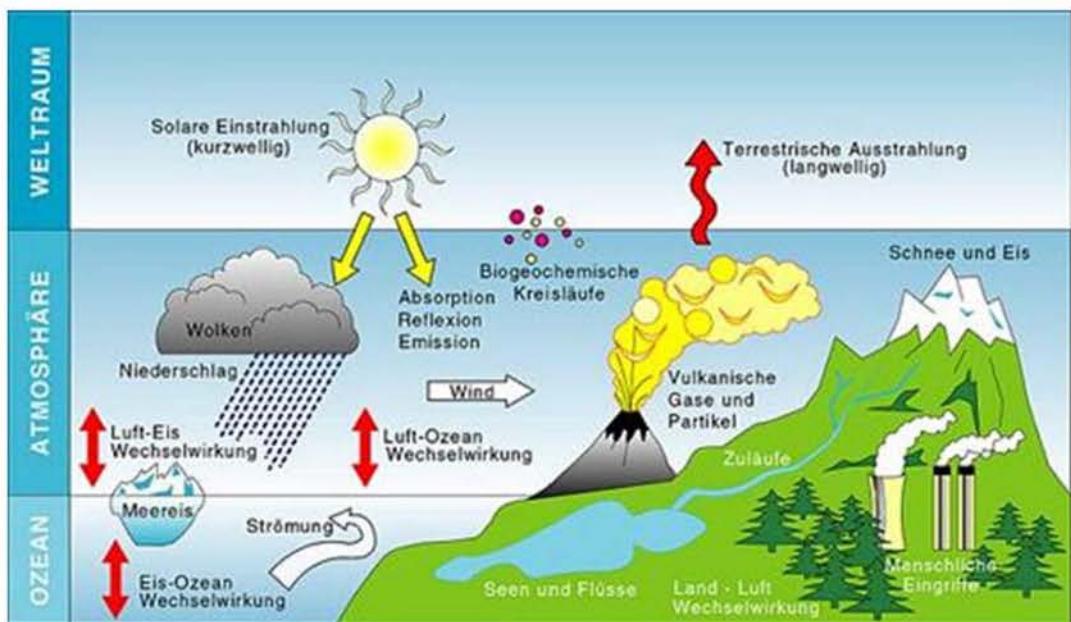
Hochleistungsrechnen - © Thomas Ludwig

33

Siehe:

- http://de.wikipedia.org/wiki/John_von_Neumann
- <http://de.wikipedia.org/wiki/ENIAC>

Komponenten im Klimasystem



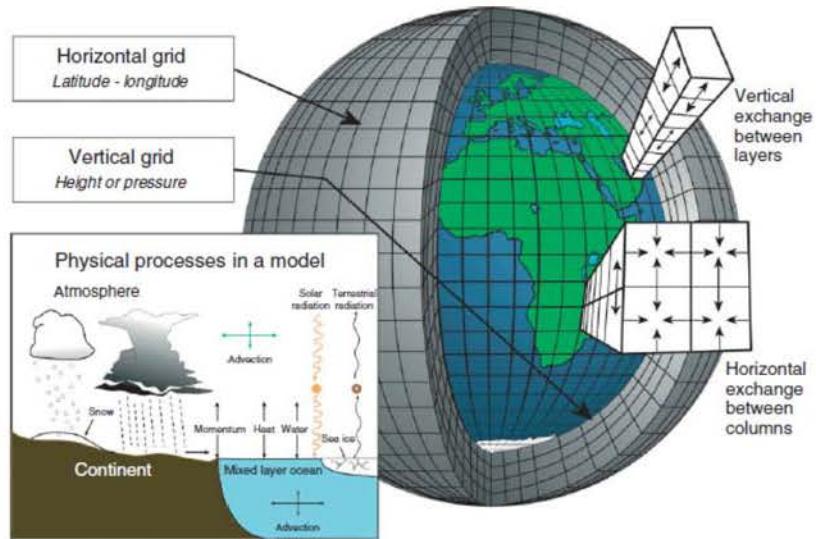
12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

34

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 !!

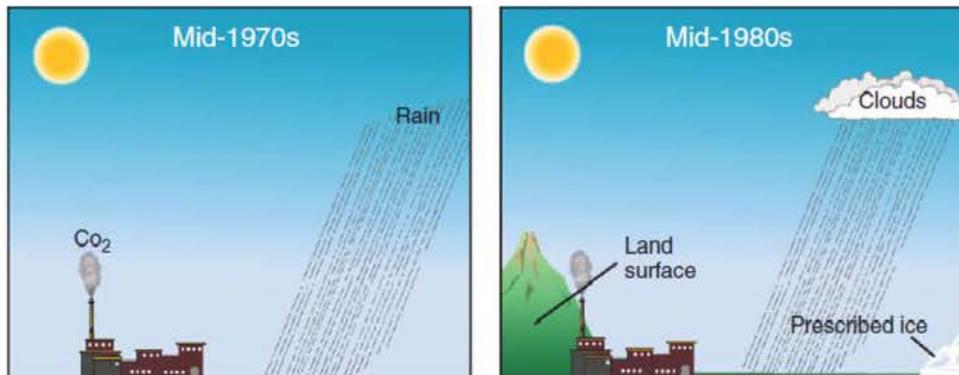


12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

35

Evolution der Klimamodelle



- Auflösungen:

- Horizontal 5°

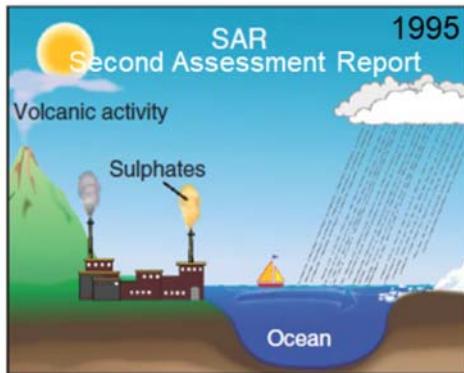
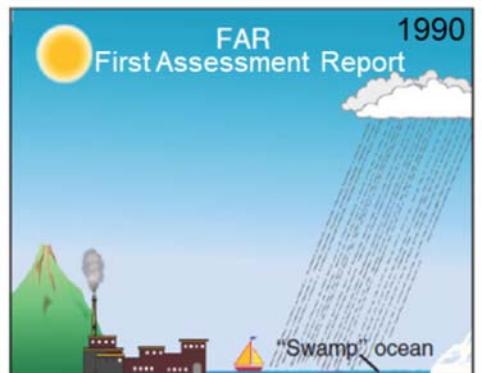
- Vertikal 12 Schichten

- 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

12.10.2021

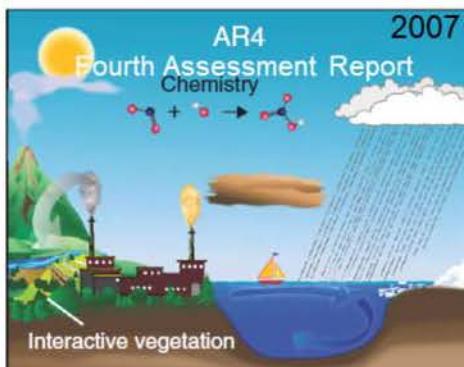
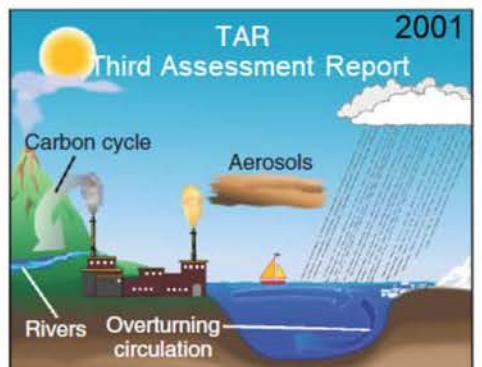
Hochleistungsrechnen - © Thomas Ludwig

37

Siehe:

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

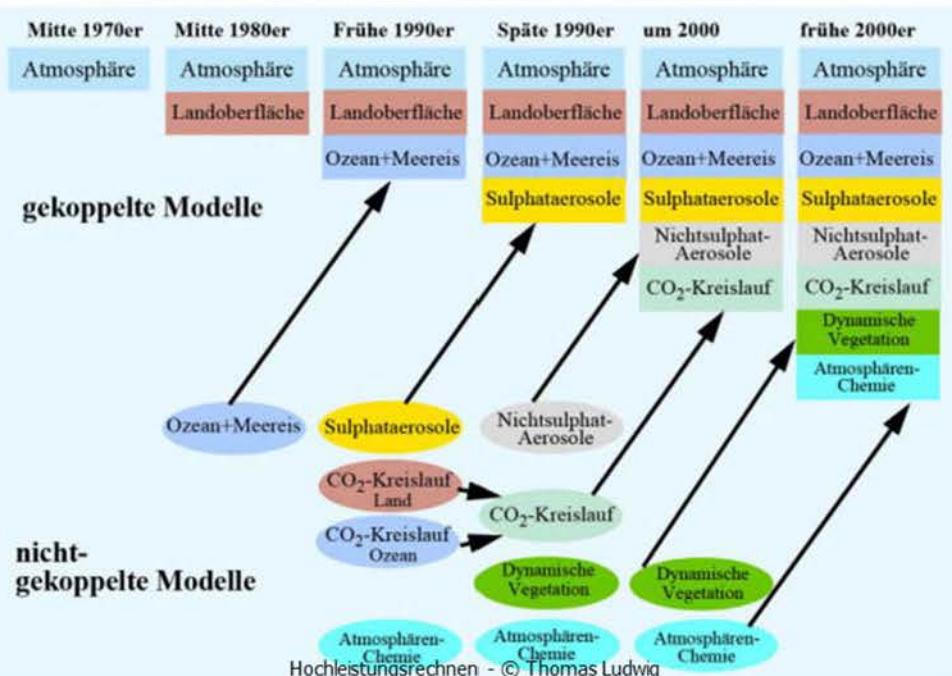
Evolution der Klimamodelle (3) (IPCC)



- Auflösungen:
Horizontal 140 km

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

Prozesse im Klimamodell



12.10.2021

39

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: 50(?) PB in 100(?) Mio. Dateien

12.10.2021

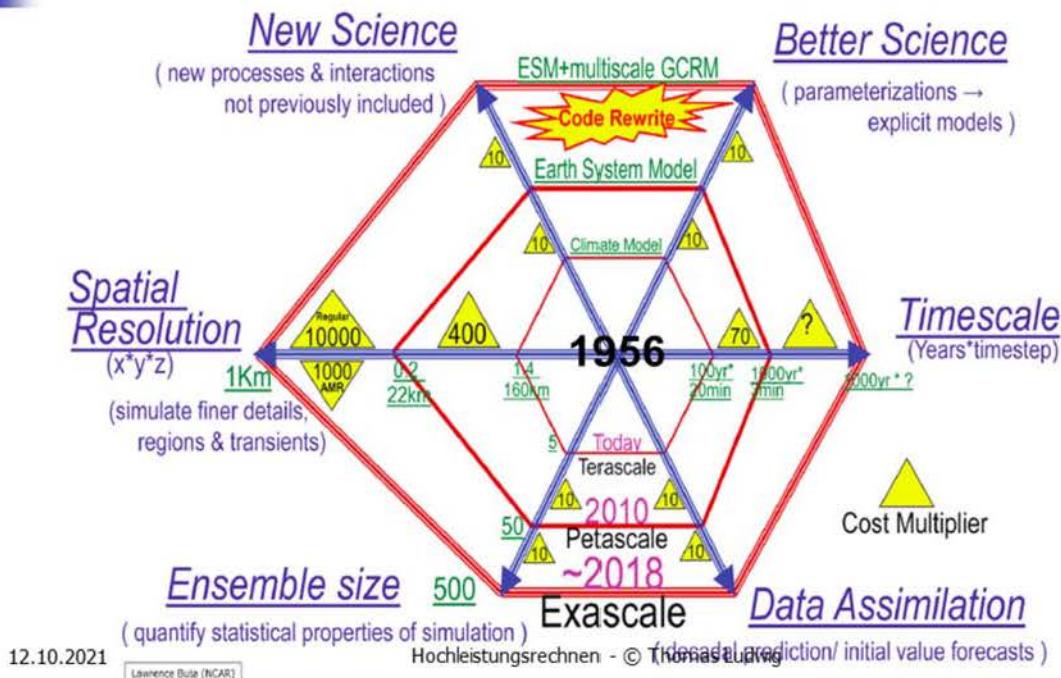
Hochleistungsrechnen - © Thomas Ludwig

40

Siehe:

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

Der Big Bang der Klimamodellierung



12.10.2021

Lawrence Buja (NCAR)

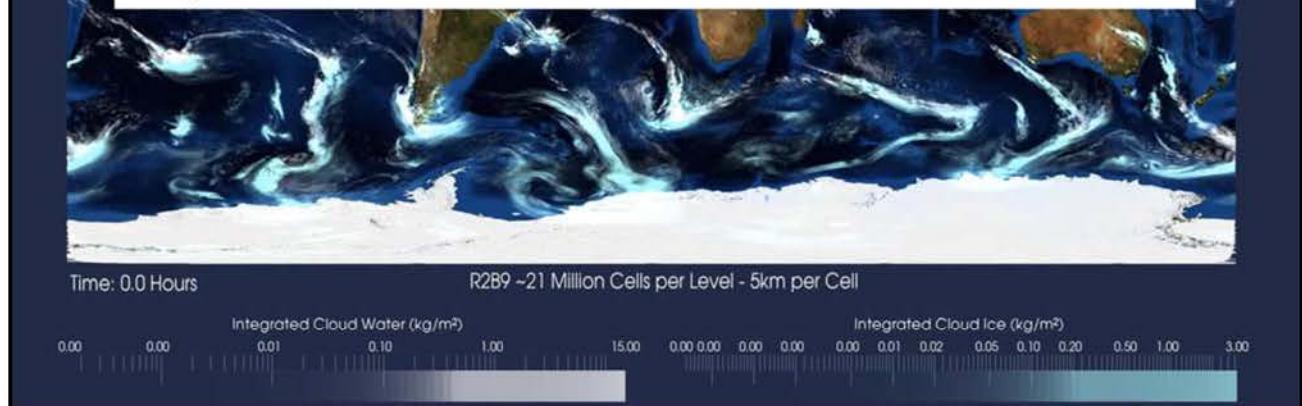
41

Neueste Entwicklungen: HP(CP)²-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

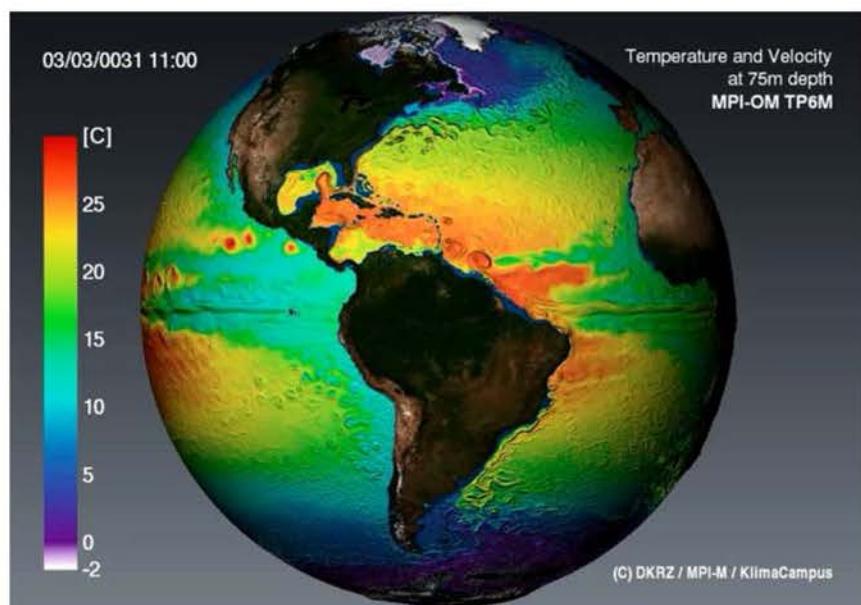
Beispiel: ICON R2B9 globales Atmosphärenmodell

- Horizontale Auflösung: 5 km
- 21 Millionen Zellen pro Level * 137 Level = 2,9 Milliarden Zellen
- 11,6 GByte pro 3D-Feld und Zeitschritt
- Simulationszeitraum: 3 Wochen
- Datengröße ca. 60 TByte (32 Bit)
- Rechenzeitbedarf: 120 Stunden auf 515 Knoten
- Speicherintervall: stündlich



Ergebnisvisualisierung

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



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

44

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
DKRZ – Partner der Klimaforschung

Höchste Rechenleistung.
Ausgereiftes Datenmanagement.
Kompetenter Service.

Vision

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

12.10.2021
FHH-BWF-UNI

Hochleistungsrechnen - © Thomas Ludwig
Deutsches Klimarechenzentrum Bundesstraße 45 Januar 2008

Lehmann + Partner Architekten

46

30 Jahre DKRZ (1987-2017)

Erster Computer: Control Data Cyber-205

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

200 MFLOPS hat ein altes 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



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

48

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

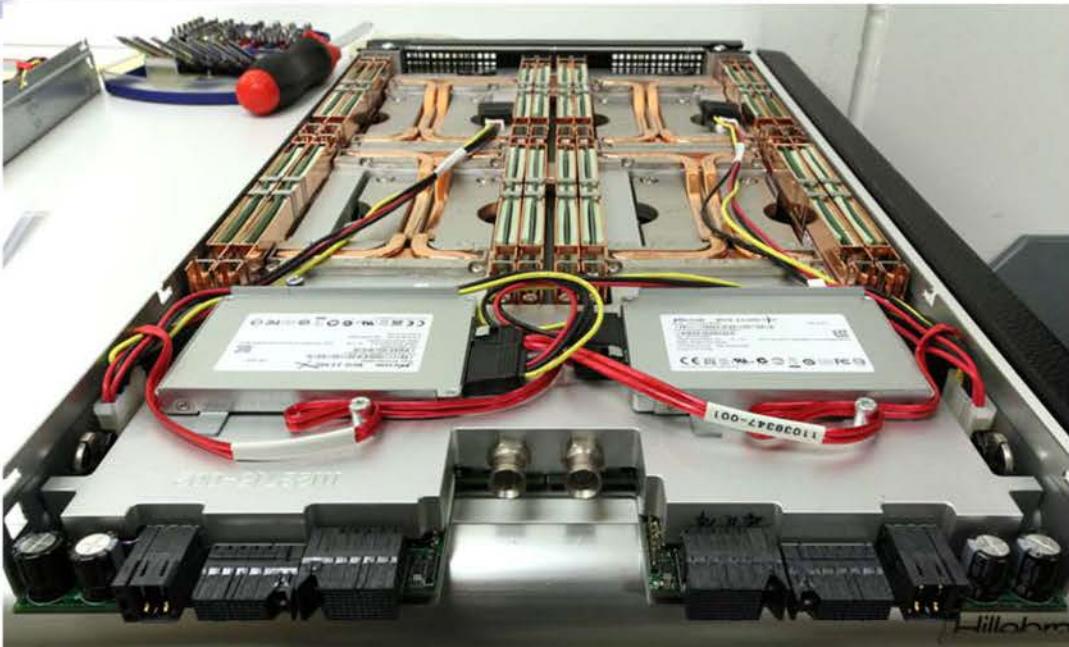


12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

49

Doppel-Rechnerknoten



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

50

Aktuelles Bandarchiv

- 7 Sun StorageTek SL8500 Bandbibliotheken
- 67.000+ Stellplätze für Bänder (+10.000 in Garching)
- Ca. 80 Bandlaufwerke
- 400 PB Kapazität mit LTO7-Bändern (ca. 120 PB belegt)



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

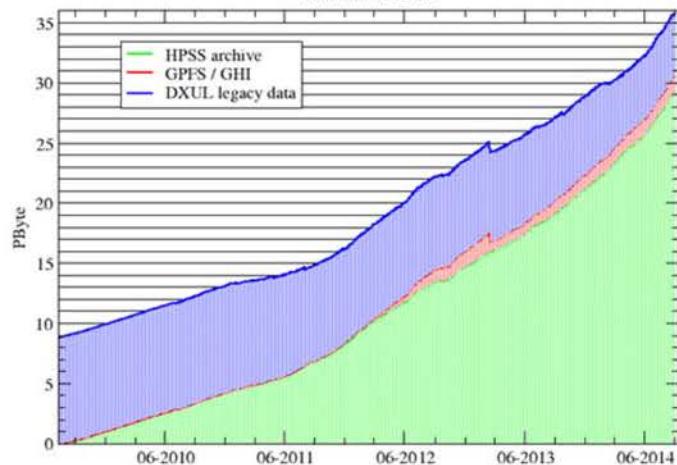
51

Datenvolumen

Aktuell ca. 8 PB/Jahr

DKRZ tape archive

total data in HPSS



12.10.2021

Wed Sep 10 14:00:00 2014

Hochleistungsrechnen - © Thomas Ludwig

52

Ausbau von HLRE-2 zu HLRE-3

Charakteristikum	2009	2015	Faktor
Leistung	150 TFLOPS	3,6 PFLOPS	24x
Rechnerknoten	264	3.000	12x
Hauptspeicher	20 TB	240 TB	12x
Festplattenkapazität	6 PB	54 PB	9x
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

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

53

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

4. Themengebiete der Informatik

Denkbar sind speziell Ausrichtungen auf Klimaforschung bei

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

Achtung: Brainware, Peopleware und Wetware existieren nicht in der englischen Sprache. ☺



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

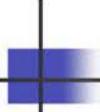
Software – von höchster Bedeutung!

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

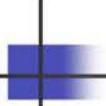
- Hohe Investitionen in Hardware (schön! ☺)
- Aber
 - Nicht effizient nutzbare Hardware ist verschwendetes Geld
- Stattdessen
 - Angemessene Anteile in Personal investieren
 - Ausbildung in Bereichen wie z.B. Programmierung, Fehlersuche, Leistungsoptimierung
 - Könnte wissenschaftliche Produktivität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

5. Herausforderungen

- Der technische Fortschritt in der Hardware der Informationssysteme hat ein exponentielles Wachstum
 - 2020: das Ende für Halbleitertechnik ist wohl erreicht
- 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 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 schnell an
 - Es wird schwieriger, die Systeme in vollem Umfang effizient zu nutzen
 - Klimaforschung hat aber sehr langlaufende Modelle



Klimaforschung und Hochleistungsrechnen

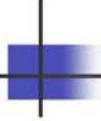
Zusammenfassung

- 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

1. Parallelismus

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

Beispiel

- Ein Arbeiterin schaufelt ein 1m*1m*1m großes Loch in einer Stunde
- Zwei Arbeiterinnen schaufeln ein 1m*1m*1m großes Loch in einer halben Stunde
- Tausend Arbeiterinnen? ☺
- Ein 1m*1m*1000m langer Graben? ☺
- Ein 1m*1m*1000m **tiefes Loch?** ☺ ☺

Gadara-Aquädukt

- 155km Länge
- x Römerinnen treiben den Tunnel am Tag um y Meter voran
- Probleme
 - ...
 - ...



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

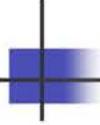
66

Siehe: <https://de.wikipedia.org/wiki/Gadara-Aqu%C3%A4dukt> – wie wurde das gebaut?



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 (in den Prozessoren)
 - Prozessoren
 - Vollständige Rechner (auch: Rechnerknoten)
 - 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

Prozessortechnologie ab ca. 2015

- Miniaturisierung gerät in die Krise

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

69

Siehe:

- https://en.wikipedia.org/wiki/Microprocessor#Multi-core_designs
- https://en.wikipedia.org/wiki/Clock_rate
- 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.

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

70

Siehe:

- https://en.wikipedia.org/wiki/Connection_Machine
- 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

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

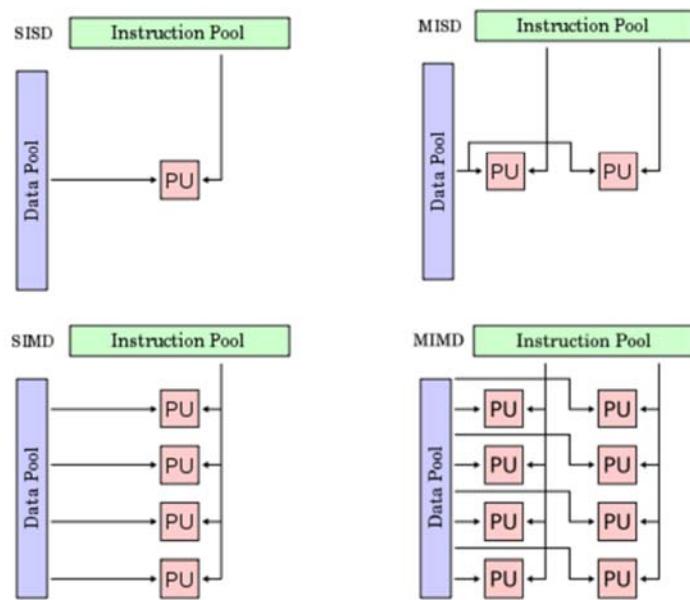
71

Siehe:

- 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)



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

72

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?

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

74

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

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

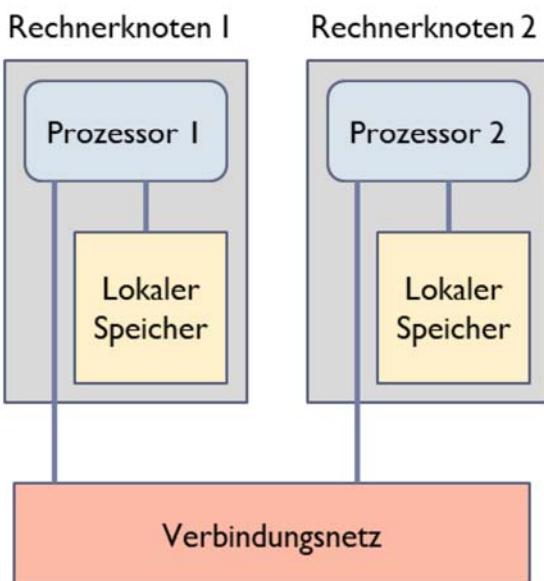
Neue Klassen von Rechnerarchitekturen

- 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
 - Heute immer Mehrkernprozessoren

12.10.2021

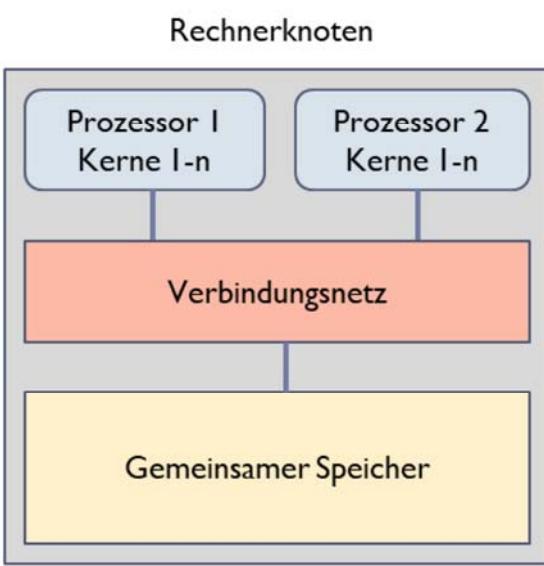
Hochleistungsrechnen - © Thomas Ludwig

76

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

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

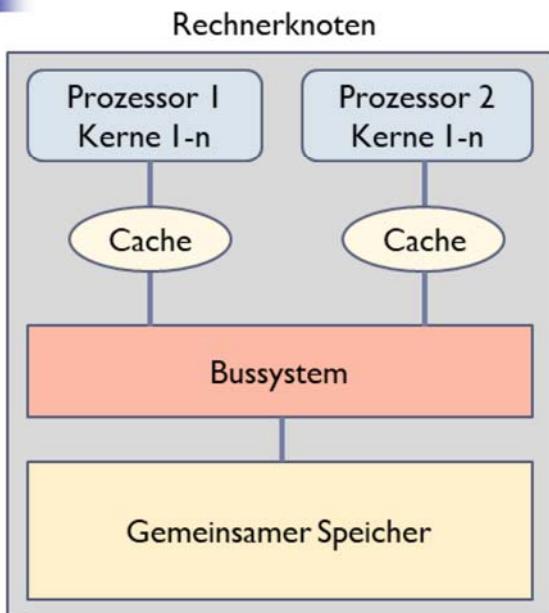
77

Siehe:

- 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)

12.10.2021

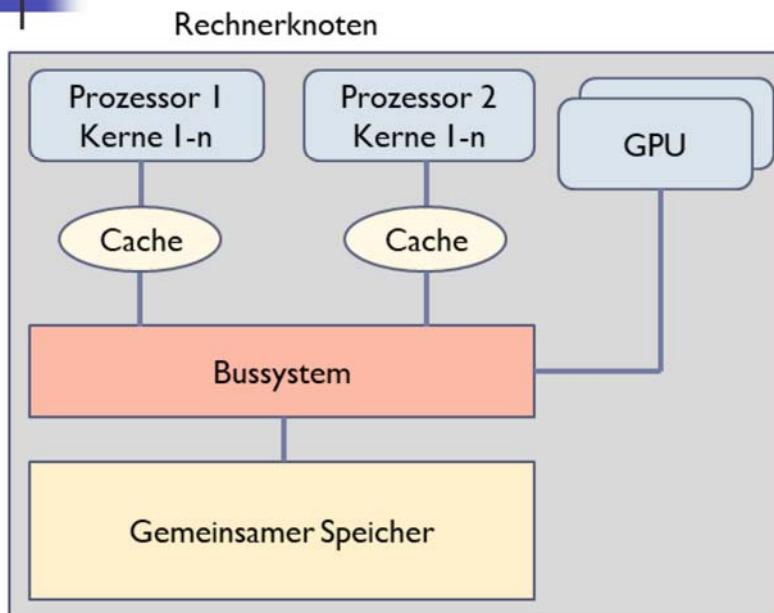
Hochleistungsrechnen - © Thomas Ludwig

78

Siehe:

- 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)
 - Oder auch : FPGA, Vektor

12.10.2021

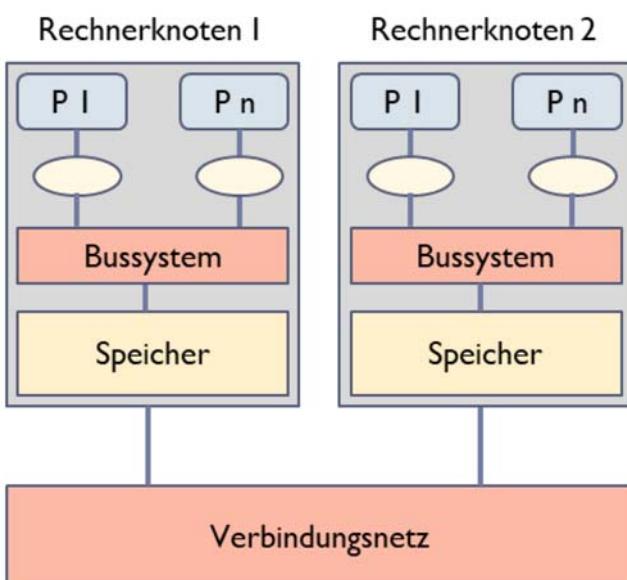
Hochleistungsrechnen - © Thomas Ludwig

79

Siehe:

- https://en.wikipedia.org/wiki/General-purpose_computing_on_graphics_processing_units
- 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

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

80

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)

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

81

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 Prozess- oder Threadzahl noch hohe Leistung bringt
- Ein Netz skaliert gut, wenn beim Ausbau die Leistung mit dem investierten Geld korreliert

Siehe:

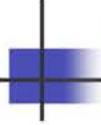
- <https://en.wikipedia.org/wiki/Scalability>

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

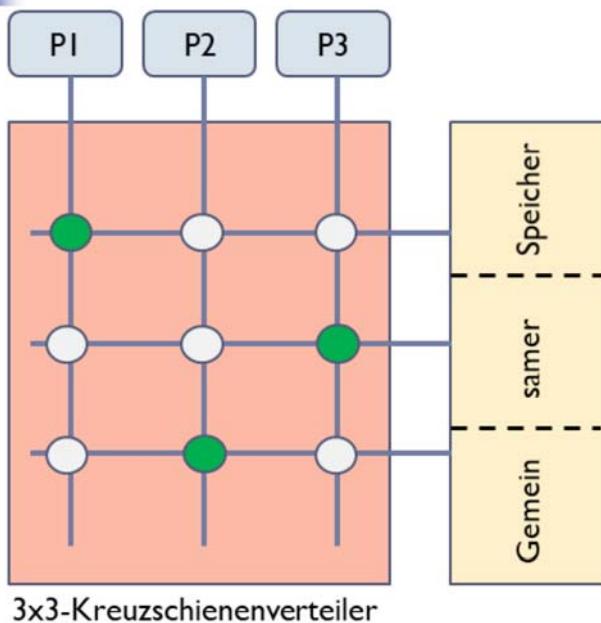
Es gibt hier eine Vielzahl von Konzepten !

- 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

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

87

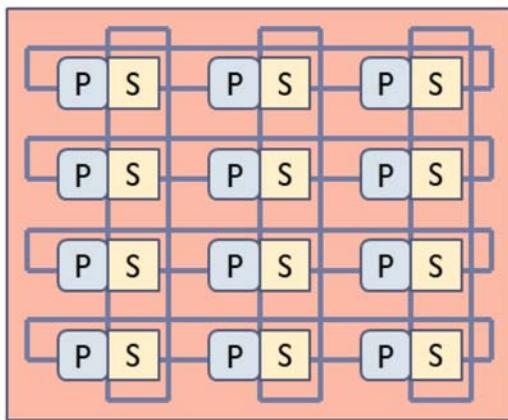
Engl.: crossbar switch

Siehe:

- 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

12.10.2021

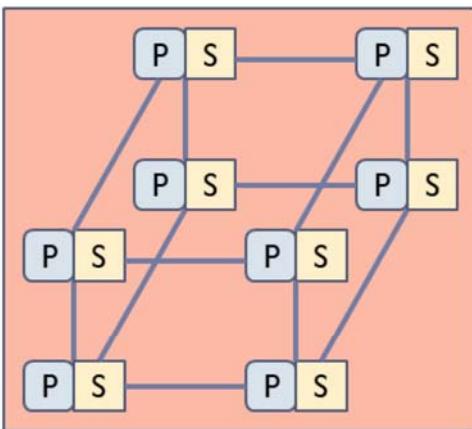
Hochleistungsrechnen - © Thomas Ludwig

88

Siehe:

- 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 Entfernung
- Hoher Grad der Vernetzung
- Knotenzahl vervierfacht, maximaler Pfad wächst um zwei

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

89

Siehe:

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

6. Hintergrundspeicher

- Lokale Platte an jedem Rechnerknoten
 - Heute meist nur für Servicezwecke auf dem Rechnerknoten
- Dateiserver ins Netz eingebunden
 - Persistente Datenhaltung
 - Engpass bei Datenzugriff
- Storage Area Network (SAN)
 - Speicherkomponenten mit eigenem Netz an die Komponenten des Clusters angehängt
- Speziallösungen für Hochleistungsrechnen
 - Spezielle Eingabe/Ausgabe-Knoten sind an Plattensysteme angeschlossen
- Hierarchical Storage Management (HSM) und Bandarchive

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

7. Spezialkonzepte

Historische Architekturen

- Workstationcluster
- Gridcomputing

Aktuelle Architekturen

- Cloud-Computing
- HPC in der Cloud

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

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

92

Siehe:

- https://en.wikipedia.org/wiki/Beowulf_cluster

Helics-Cluster Uni Heidelberg 2001

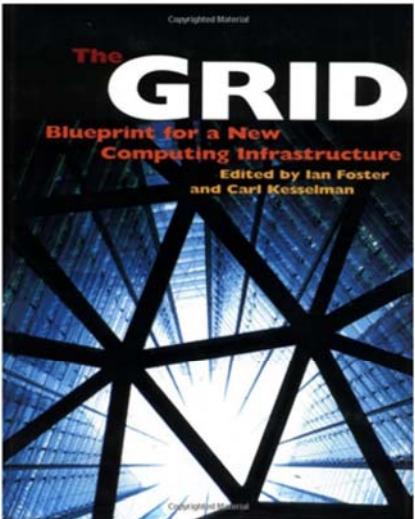


12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

93

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

12.10.2021

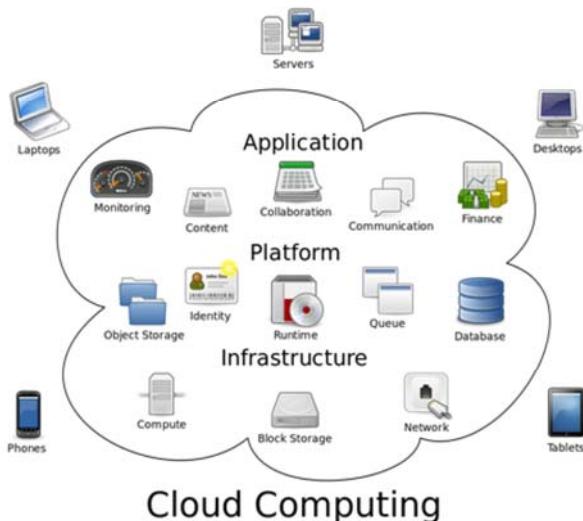
Hochleistungsrechnen - © Thomas Ludwig

94

Siehe:

- https://en.wikipedia.org/wiki/Grid_computing

Cloud-Computing



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

Der Rechner/Speicher für die Zukunft ?

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

95

Siehe:

- 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

HPC in der Cloud

- Cray-Systeme nutzen eine Mischung von Intel-Xeon-Prozessoren, Nvidia Tesla P100 GPUs, Xeon-Phi-Koprozessoren und FPGAs mit verschiedenen Netzwerken wie InfiniBand und Crays Aries-Verbindungsnetz
- ClusterStor-Speichersystem mit bis zu 80 PByte und 1,7 TByte/s

Cray supercomputers coming to Azure cloud

New offering is aimed at simulation, modeling, and other HPC tasks.

PETER BRIGHT - 10/23/2017, 6:57 PM



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

96

Siehe:

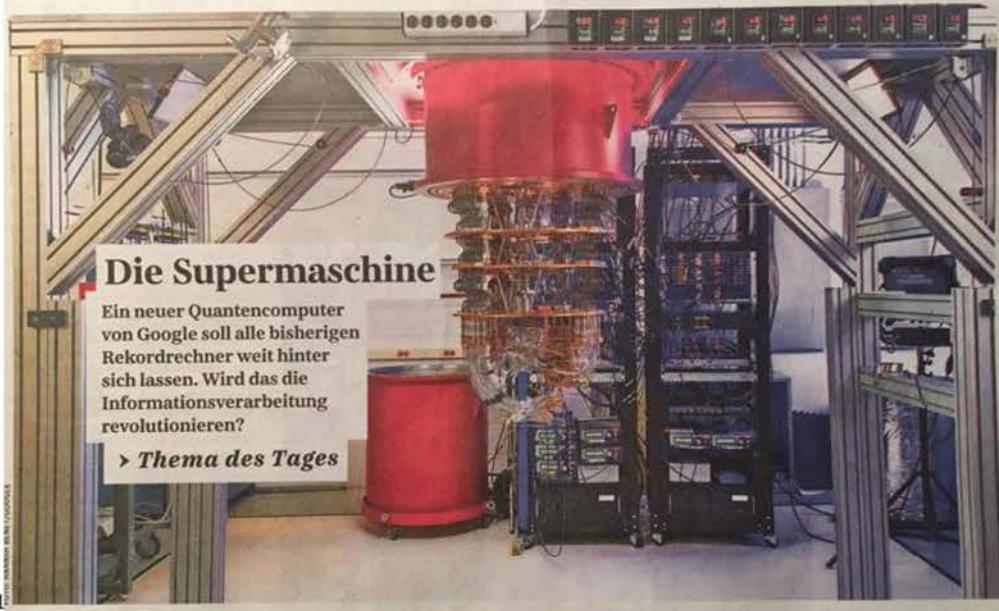
- <https://arstechnica.com/gadgets/2017/10/cray-supercomputers-coming-to-azure-cloud/>
- <https://www.heise.de/ix/meldung/HPC-in-der-Cloud-Cray-Supercomputer-aus-Azure-beziehen-3868737.html>

Quantencomputer

HF3

MÜNCHEN, DONNERSTAG, 24. OKTOBER 2019

75. JAHRGANG



Die Supermaschine

Ein neuer Quantencomputer von Google soll alle bisherigen Rekordrechner weit hinter sich lassen. Wird das die Informationsverarbeitung revolutionieren?

› Thema des Tages

12.1

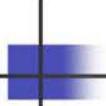
97

Quantencomputer...

- 2,5 Tage : 10.000 Jahren =
1 : 1.500.000



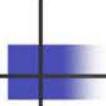
Schon im September war Googles Papier zu seiner Forschung – wohl unabsichtlich – auf einer Webseite der Nasa veröffentlicht worden. Danach gab es Streit darüber, ob Googles mathematisches Problem wirklich so schwer zu lösen war, wie der Konzern behauptet. IBM, ein Konkurrent auf dem Gebiet der Quantencomputer, hatte Googles Behauptungen als „Hype“ kritisiert. Um die Aufgabe aus dem Experiment zu lösen, braucht ein aktueller Supercomputer nicht 10 000 Jahre, wie von Google behauptet, sondern nur zweieinhalb Tage. Google habe die Fähigkeiten moderner Supercomputer nicht voll ausgenutzt. Das Schlagwort „Überlegenheit“ (supremacy) werde nur falsch verstanden werden, wie es zuvor schon vielen Begriffen aus dem Bereich der künstlichen Intelligenz ergangen sei. John Martinis, Forscher in Googles Quanten-Team, sagt nun: Er werde Daten und Programmcode des Experiments öffentlich machen, dann könnten sich die Kollegen von IBM selbst von dem Durchbruch überzeugen.



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



Hardware-Architekturen

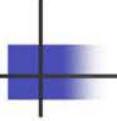
Die wichtigsten Fragen

- 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?

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

100



Die TOP500-Liste

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

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 und der Taktfrequenz des Prozessors

Bewertung durch sogenannte Benchmark-Programme

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

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

102

Siehe:

- <http://en.wikipedia.org/wiki/Flops>
- http://en.wikipedia.org/wiki/Benchmark_%28computing%29

Zum Vergleich: Prozessoren

LINPACK 1kx1k (DP)	Höchstleistung (in GFLOPS)	Durchschnittsleistung (in GFLOPS)	Effizienz (in %)
Nvidia Tesla GP100, 1,48 GHz		10600	
Nvidia Quadro P6000	19553	12901	
Xeon Skylake SP 6148	1536		
AMD Ryzen 1800X, 8K/16T, bislang unoptimiert	221 ^[14]		
Intel Core i7-7700K, 4K/8T	241 ^[14]		
Intel i7-5960X, 8K/16T	375 ^[14]		
Intel Core i7 5820k, 6K/12T, 3,3 GHz	273,1	265	

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

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

103

Siehe:

- https://de.wikipedia.org/wiki/Floating_Point_Operations_Per_Second
- <https://www.pugetsystems.com/blog/2014/09/08/Xeon-E5-v3-Haswell-EP-Performance-Linpack-595/>

Zum Vergleich: Anwendungen

- Rechnersystem Mistral am DKRZ 2016
 - 3.000 TFLOPS LINPACK-Leistung
 - Bei ca. 100.000 Prozessorkernen macht das ca. 30 GFLOPS/Prozessorkern (gemittelt Haswell/Broadwell)
- Rechnersystem Blizzard am DKRZ 2010
 - 110 TFLOPS LINPACK-Leistung
 - Bei ca. 8.500 Prozessorkernen macht das ca. 13 GFLOPS/Prozessorkern
- Klimaberechnung IPCC AR5 auf Blizzard
 - Geschätzter Bedarf: 30 Millionen Prozessorkernstunden
 - DKRZ stellte 60 Millionen pro Jahr mit Power6/IBM bereit
 - Ca. 50 TFLOP pro Prozessorkernstunde
 - 13 GFLOPS * 3.600s/h
 - 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_{peak} ist die theoretische Maximalleistung

Bezeichnet als HPL – High Performance Linpack

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

105

Siehe:

- 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 das somit 380 T€ für den Linpack
- HPL am DKRZ 2017
 - Unerwünscht wegen Stromverbrauchsspitze
- HPL repräsentiert nur wenige parallele Programme
- Manche Rechnern werden auf guten LINPACK hin entworfen
- In der Praxis deshalb Anwendungsbenchmarks

2. Die TOP500-Liste

Website 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

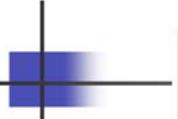
12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

107

Siehe:

- <http://top500.org/>



Rank	System	Cores	Rmax [TFlop/s]	Rpeak [TFlop/s]	Power [kW]
1	Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442,010.0	537,212.0	29,899
2	Summit - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
3	Sierra - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
4	Sunway TaihuLight - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
5	Selene - NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband, Nvidia NVIDIA Corporation United States	555,520	63,460.0	79,215.0	2,646
6	Tianhe-2A - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000, NUDT National Super Computer Center in Guangzhou China	4,981,760	61,444.5	100,678.7	18,482
7	JUWELS Booster Module - Bull Sequana XH2000 , AMD EPYC 7402 24C 2.8GHz, NVIDIA A100, Mellanox HDR InfiniBand/Par-Tec ParaStation ClusterSuite, Atos Forschungszentrum Juelich (FZJ) Germany	449,280	44,120.0	70,980.0	1,764

12.10.2021

TOP500 Nov 2019 1-8 / Ein paar wichtige Daten sollten Sie kennen:

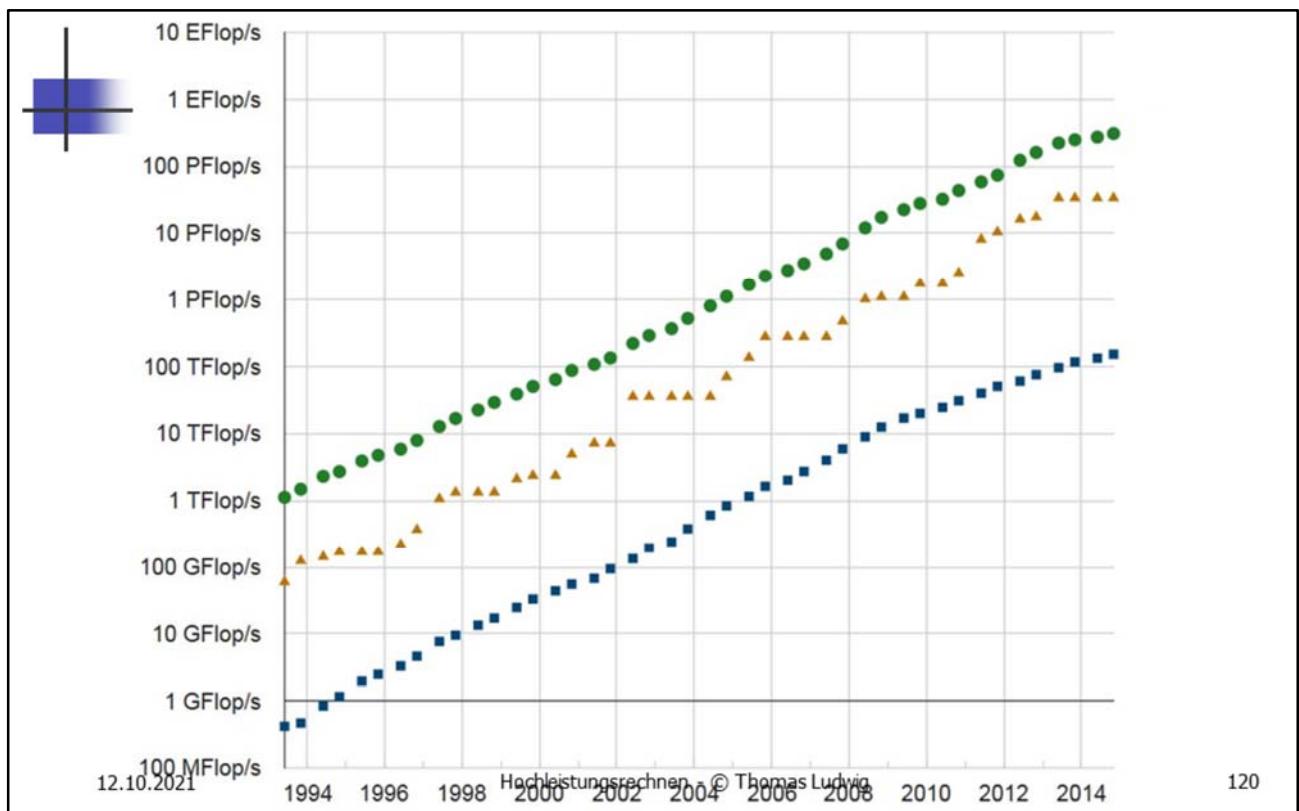
- Nr 1: Mit 7,6 Millionen Prozessorkernen kommt Fugaku auf knapp ein halbes EFLOPS bei knapp 30 MW mit ARM-Prozessoren.
- Nr 2: Mit knapp 2,4 Millionen Prozessorkernen kommt Summit auf 148 PFLOPS bei 10 MW mit IBMs Power9-Prozessor und Nvidia GV 100.
- Nr 3: Mit etwa 1,5 Millionen Prozessorkernen kommt Sierra auf 94 PFLOPS bei 7 MW mit derselben Architektur.
- Nr 4: Mit über 10 Millionen Prozessorkernen kommt Sunway Taihu-Light auf 93 PFLOPS bei 15 MW mit Prozessor-Eigenentwicklung mit 260 Cores.
- Nr 7: das schnellste deutsche System steht in Jülich mit ca. 450 Tausend Prozessorkernen und 44 PFLOPS mit vielen Beschleunigerkarten.



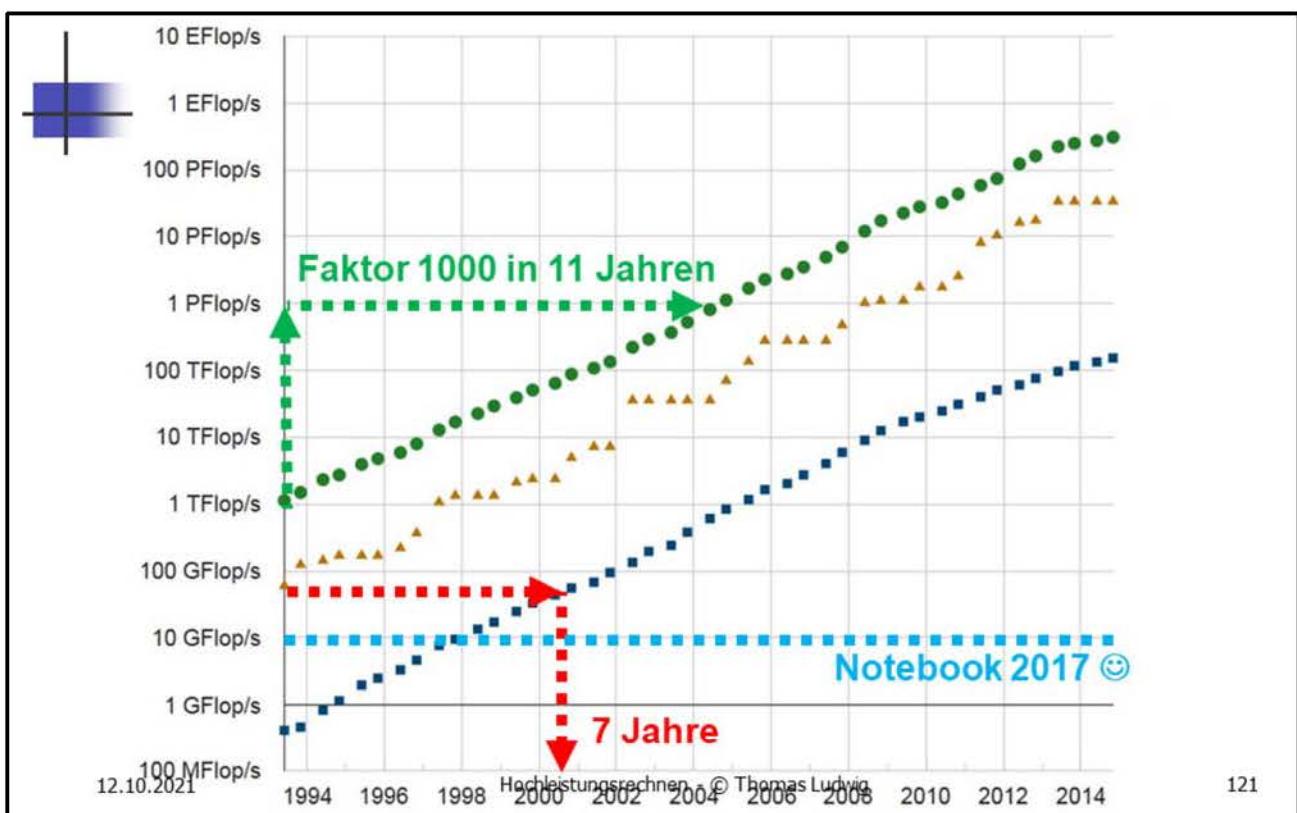
Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
7	JUWELS Booster Module - Bull Sequana XH2000 , AMD EPYC 7742 24C 2.8GHz, NVIDIA A100, Mellanox HDR InfiniBand/ParTec ParaStation ClusterSuite, Atos Forschungszentrum Juelich (FZJ), Germany	449,280	44,120.0	70,980.0	1,764
15	SuperMUC-NG - ThinkSystem SD650, Xeon Platinum 8174 24C 3.1GHz, Intel Omni-Path, Lenovo Leibniz Rechenzentrum, Germany	305,856	19,476.6	26,873.9	
16	Hawk - Apollo 9000, AMD EPYC 7742 64C 2.25GHz, Mellanox HDR InfiniBand, HPE HLRS - Hochleistungsrechenzentrum Stuttgart, Germany	698,880	19,334.0	25,159.7	3,906
44	JUWELS Module 1 - Bull Sequana X1000, Xeon Platinum 8160 24C 2.7GHz, Mellanox EDR InfiniBand/ParTec ParaStation ClusterSuite, Atos Forschungszentrum Juelich (FZJ), Germany	114,480	6,177.7	9,891.1	1,361
47	Emmy+ - BullSequana X, Intel Xeon Platinum 9242/Gold 6148, Intel Omni-Path, Atos HLRN+ at GWDG/University of Göttingen, Germany	120,296	5,948.8	8,911.3	
51	COBRA - Intel Compute Module HNS2600BP, Xeon Gold 6148 20C 2.4GHz, Intel Omni-Path, Intel Max-Planck-Gesellschaft MPG/IPPC, Germany	127,520	5,612.8	9,793.5	1,635
55	Eise - Bull Intel Cluster, Intel Xeon Platinum 9242 480 2.3GHz, Intel Omni-Path, Atos HLRN at ZIB/Konrad-Zuse-Zentrum Berlin, Germany	103,680	5,355.9	7,630.8	1,258
76	JURECA - T-Platforms V-Class/Dell C6320P, ES-2680v3/Phi 7250-F, EDR/Intel Omni-Path/ParTec ParaStation, Tesla K80/K40, T-Platforms , Intel , Dell Forschungszentrum Juelich (FZJ), Germany	155,150	3,782.6	6,563.8	1,345
100	Lichtenberg II (Phase 1) - MEGWARE MiriQuid, Intel Xeon Platinum 9242 48C 2.3GHz, Mellanox InfiniBand HDR100, MEGWARE Technische Universität Darmstadt, Germany	59,136	3,148.4	4,352.4	690
109	Mistral - bullx DLC 720, Xeon ES-2680v3 12C 2.5GHz/ES-2695V4 18C 2.1GHz, Infiniband FDR, Atos DKRZ - Deutsches Klimarechenzentrum, Germany	99,072	3,010.7	3,962.9	1,116
12.10.2021 (80)					114

Nov 2020 D

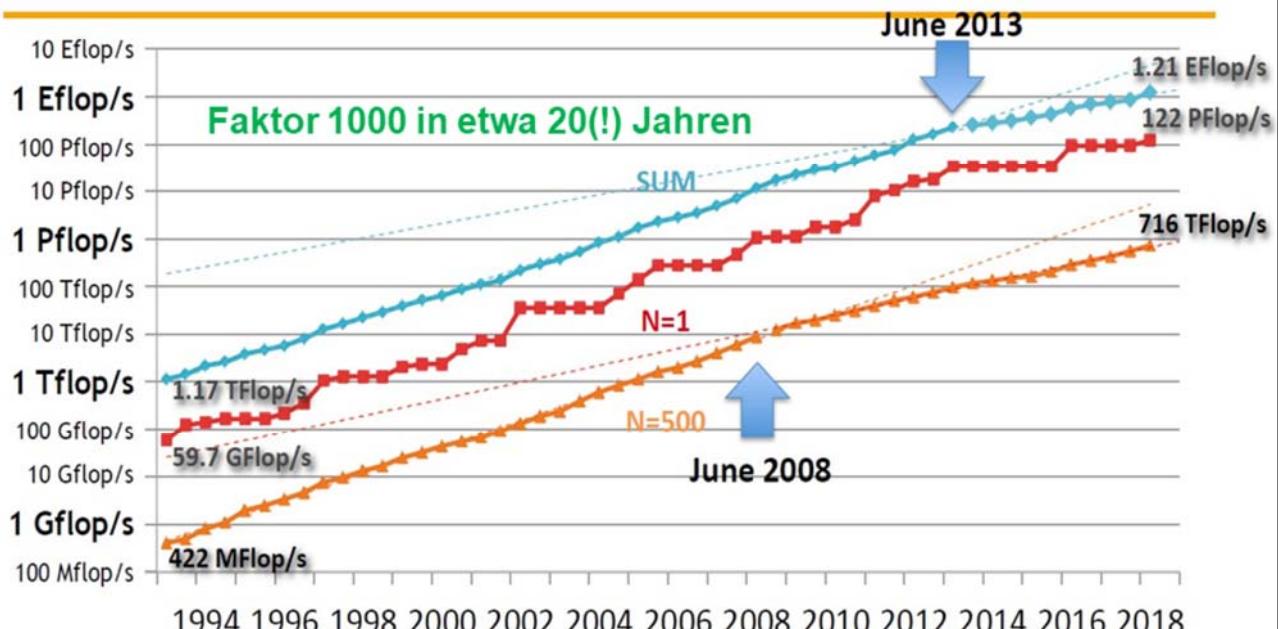
17 Einträge (2019: 16)



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



PERFORMANCE DEVELOPMENT



12.10.2021

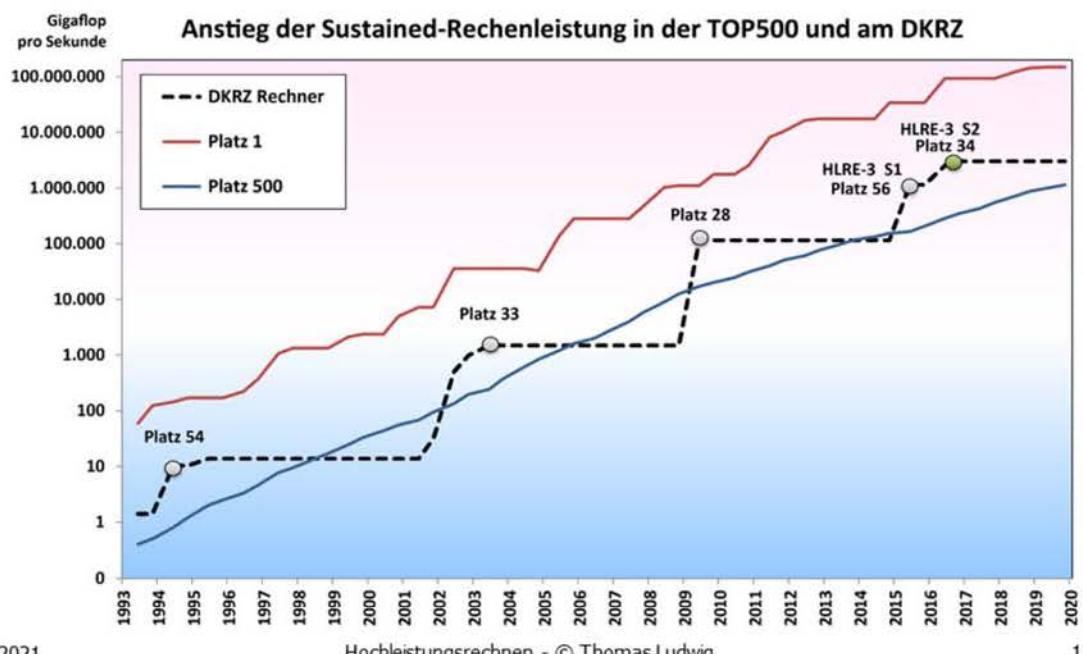
Hochleistungsrechnen - © Thomas Ludwig

122

Siehe:

https://www.top500.org/static/media/uploads/top500_ppt_201806.pdf

Ranking des DKRZ



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

123



https://www.top500.org/static/media/uploads/top500_ppt_201806.pdf

Highlights of the 51st TOP500 List

ISC 2018,
Frankfurt,
June 25, 2018

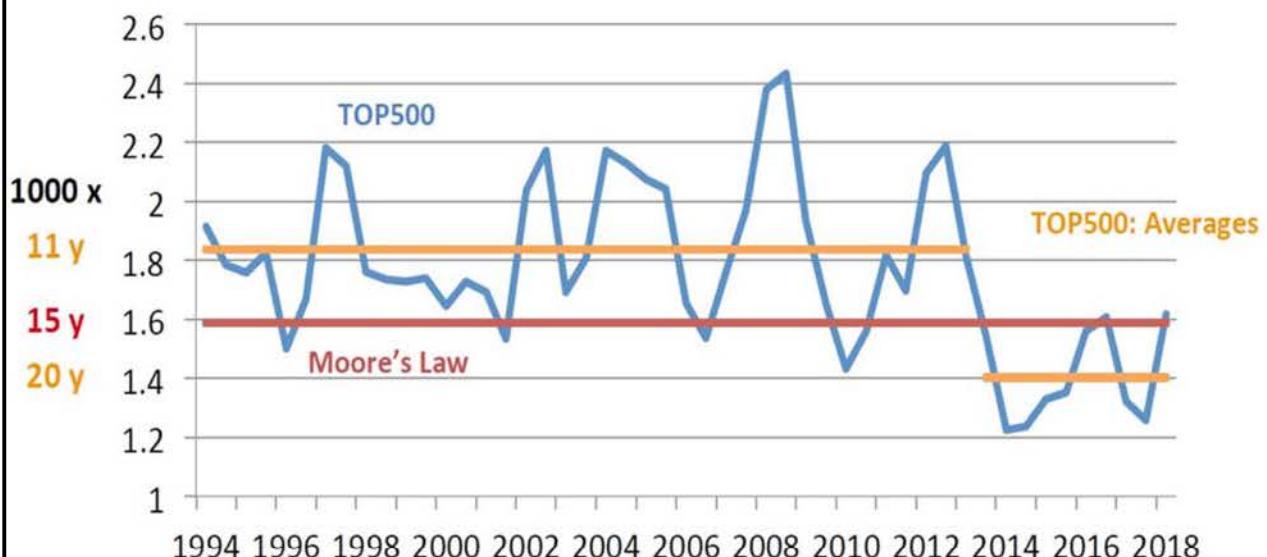
Erich
Strohmaier

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

124

ANNUAL PERFORMANCE INCREASE OF THE TOP500

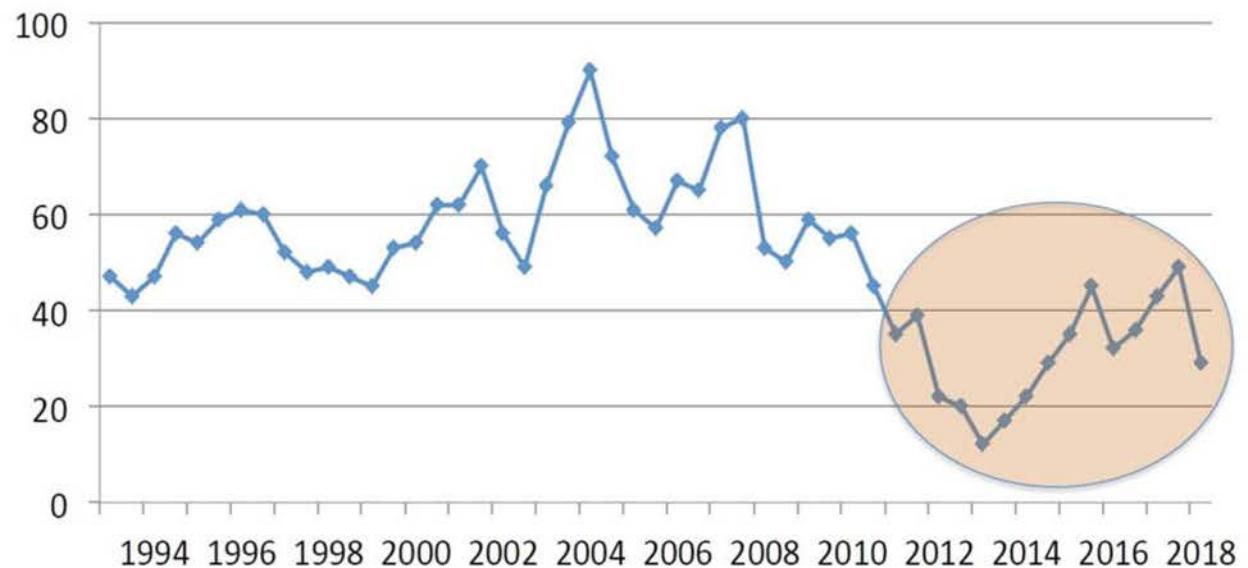


12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

125

RANK AT WHICH HALF OF TOTAL PERFORMANCE IS ACCUMULATED

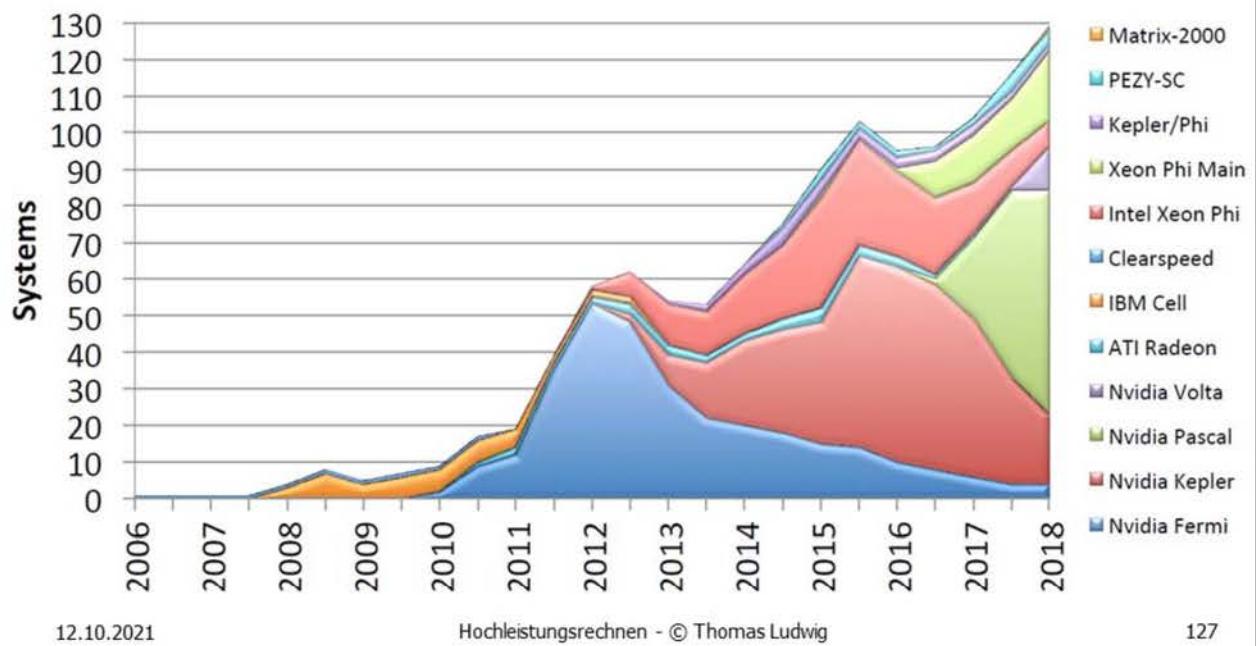


12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

126

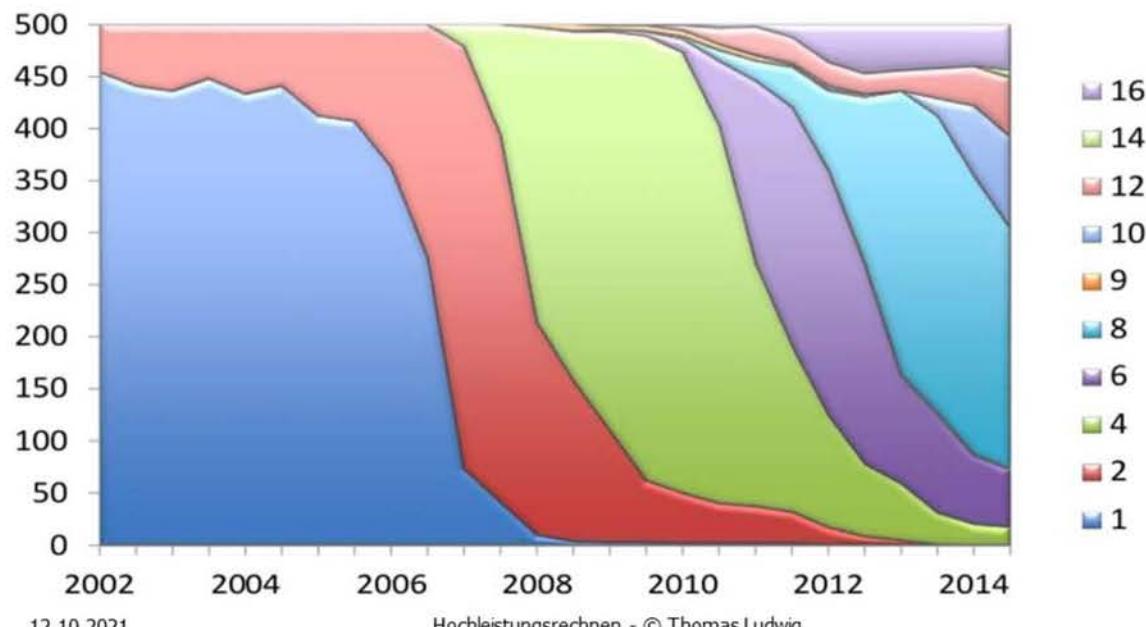
ACCELERATORS

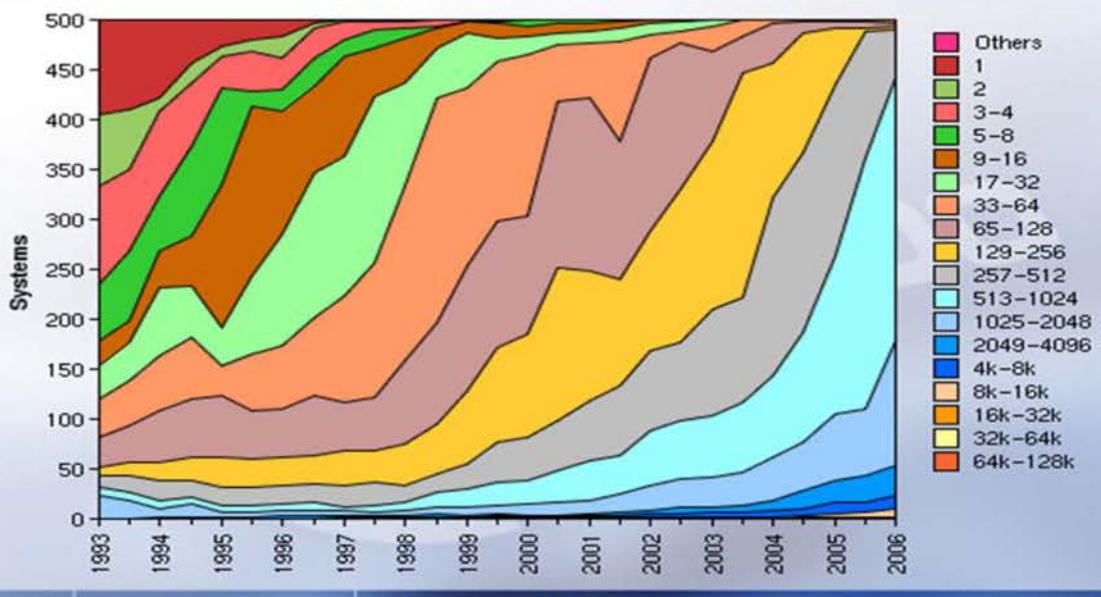


12.10.2021

127

CORES PER SOCKET

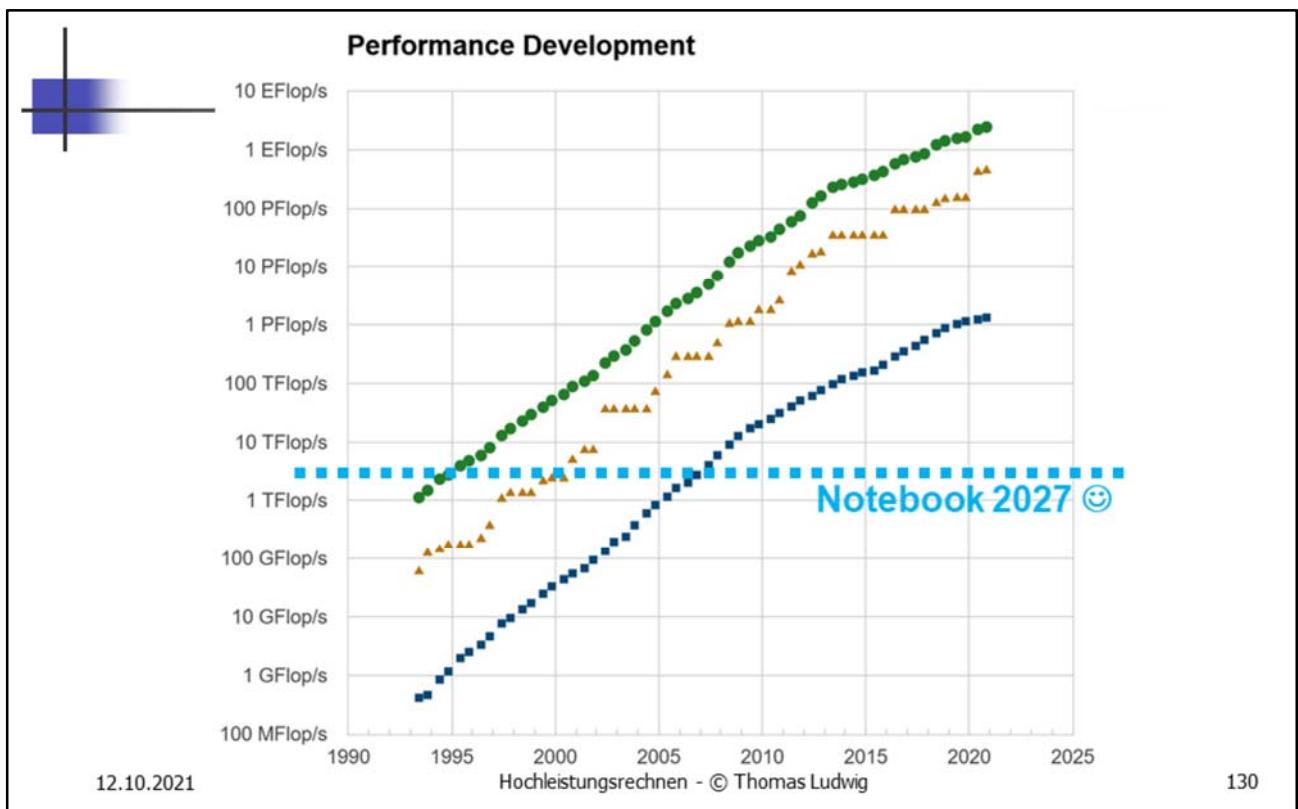


System Processor Counts / Systems

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

129



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)

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

131

AUSBLENDEN

Leistungsentwicklung Nov. 14 bis Nov. 20

Moore's Law

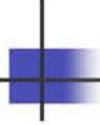
„Verdopplung der Transistorzahl alle 18 Monate“
(entspricht evtl. Leistungsverdopplung)

Nov14–Nov20: 6 Jahre = 4x18 Monate
etwa Faktor $2^4=16$

Leistung Summe: 309 PFlop/s – 2,4 EFlop/s ($\times 7,8$)

Leistung #1: 34 PFlop/s – 442 PFlop/s ($\times 13$)

Leistung #500: 153 TFlop/s – 1,3 PFlop/s ($\times 8,5$)



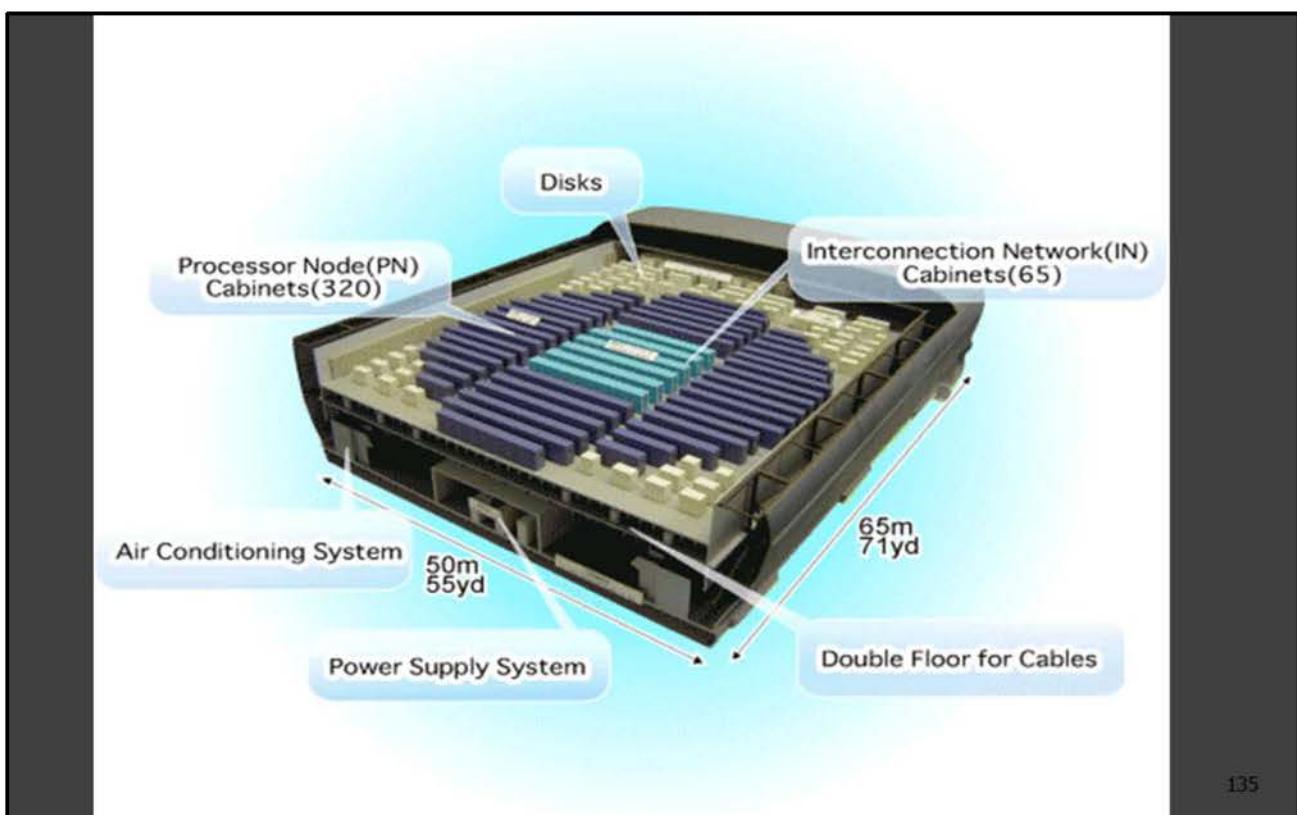
3. Beispielsysteme

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

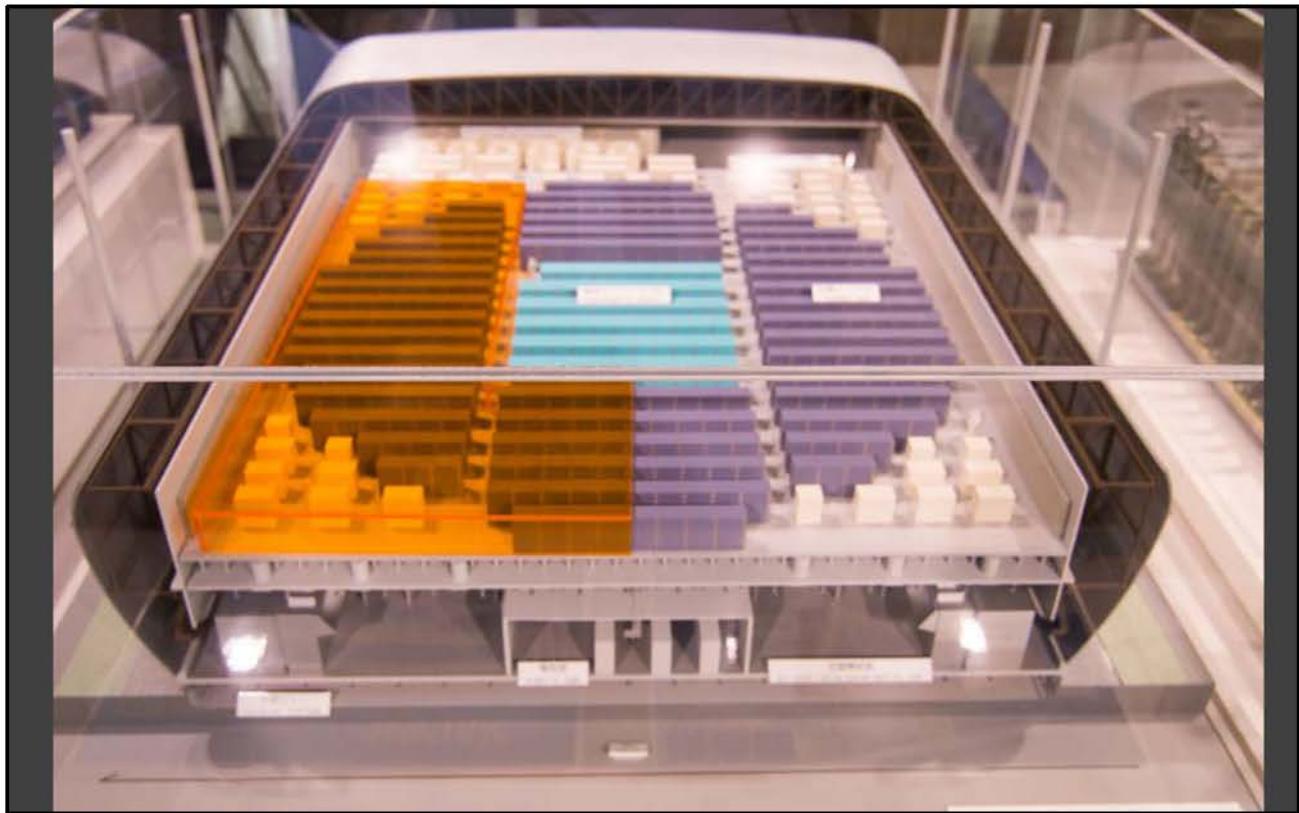
NECs Earth Simulator (6/2002-11/2008)

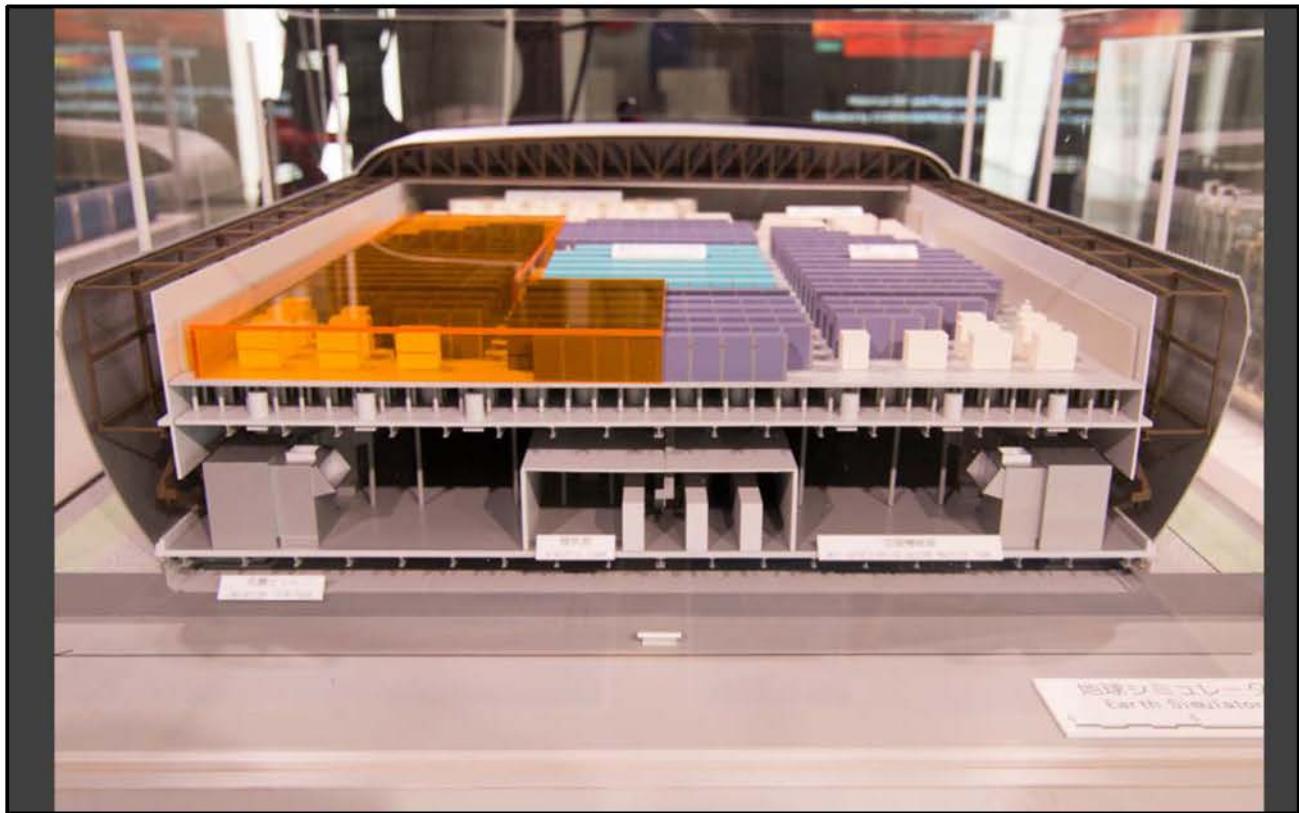
- 640 Knoten
- Zu je 8 Vektorprozess.
- 5120 Prozessoren
- 0,15 mikron Kupfer
- 200 MioUSD Rechner
- 200 MioUSD Gebäude und Kraftwerk
- Zum Zwecke der Klimaforschung etc.
- ▶ 36 TFLOPS
- ▶ 10 TByte Hauptspeicher
- ▶ 700 TByte Festplatten
- ▶ 1,6 PByte Bandspeicher
- ▶ 83.000 Kupferkabel
- ▶ 2.800 km/220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

Computenic-Shock



135





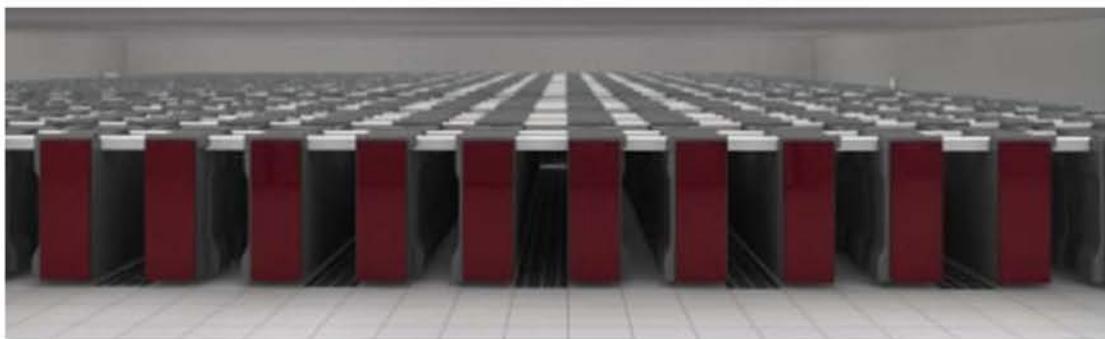






Fujitsu K Computer

- 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

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

141







Gewinner des Schönheitswettbewerbs

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

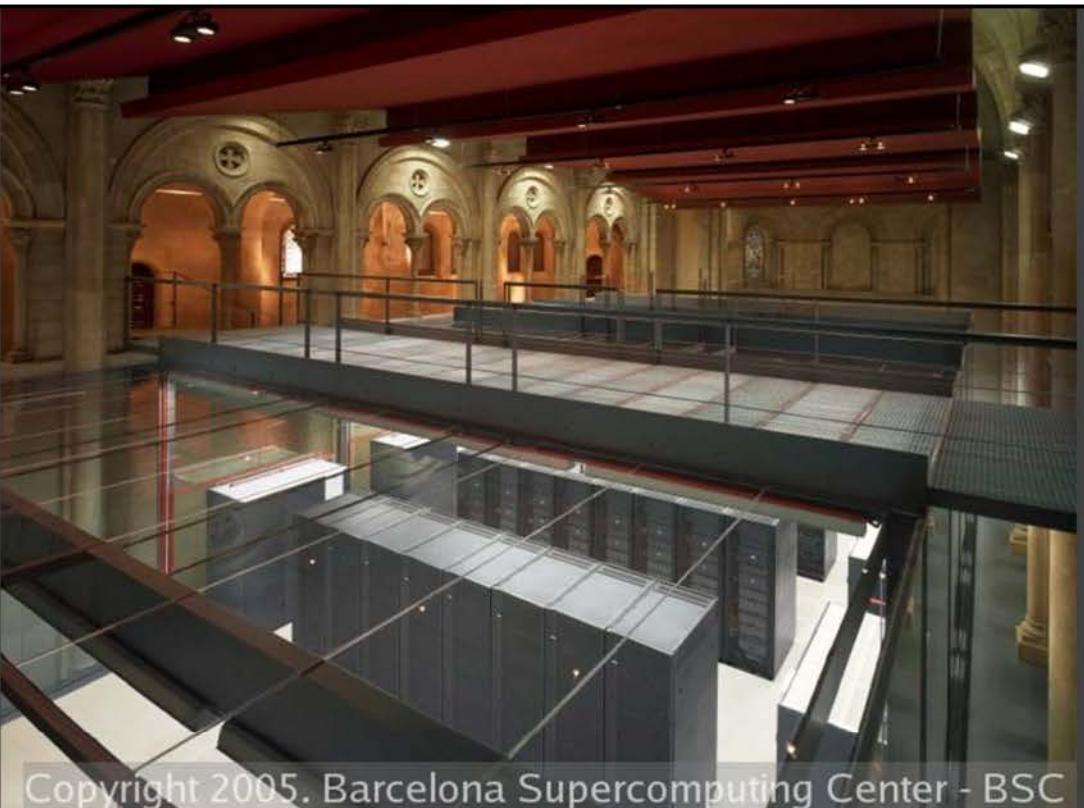
145



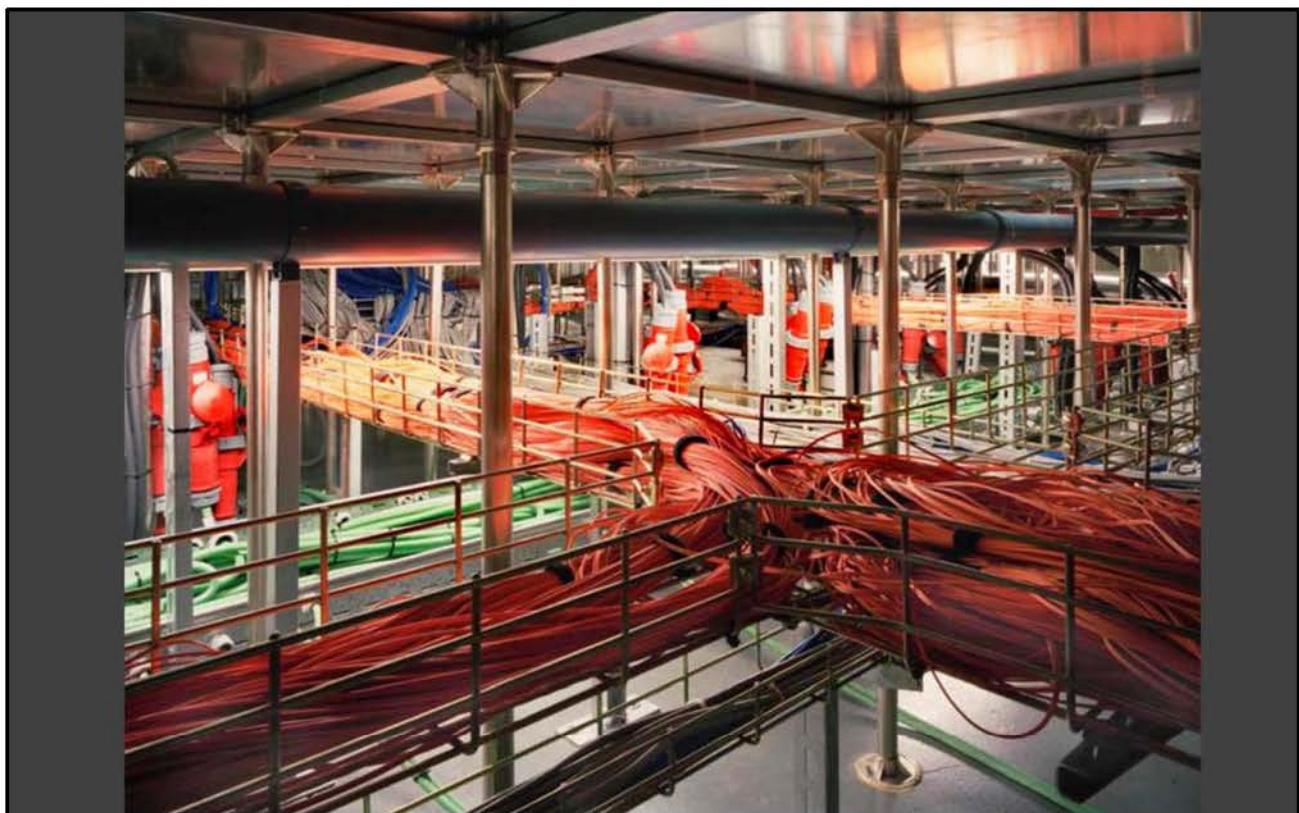
Copyright 2006, Barcelona Supercomputing Center - BSC

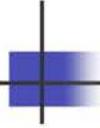


Copyright 2005. Barcelona Supercomputing Center - BSC



Copyright 2005. Barcelona Supercomputing Center - BSC





4. Historische Sicht

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

Jun
1993

12.10.202

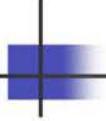
Rank	Manufacturer Computer/Procs	R _{max} R _{peak}	GFLOPS	Installation Site Country/Year	Inst. type Installation Area	Nmax Nhalf	Computer Family Computer Type
1	TMC CM-5/1024/ 1024	59.70 131.00	Los Alamos National Laboratory USA/	Research Energy	52224 24064	TMC CMS CMS	
2	TMC CM-5/1024/ 1024	59.70 131.00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

Rmax-Angabe in GFLOPS

Jun
1993
D

12.10.202

	Rank	Manufacturer Computer/Procs	R _{max} R _{peak}	GFLOPS	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-2z/4-128/ 4	1.41 1.95	DKRZ Germany/1988		Research Weather		Cray2/3 Cray 2
	206	Cray/SGI CRAY-2/4-256/ 4	1.41 1.95	Universitaet Stuttgart Germany/1986		Academic		Cray2/3 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

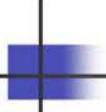
Zusammenfassung

- Die Rechnerleistung wird mit einem numerischen Benchmark-Programm (LINPACK) evaluiert
- Die TOP500-Liste verzeichnet halbjährig die schnellsten Rechner weltweit
- Die schnellsten Rechner haben die 100-Petaflops-Grenze durchbrochen
- Wir erwarten für ca. 2021 den ersten Exaflops-Rechner
- Aktuelle Probleme: Energiebedarf, Skalierbarkeit der Anwendungen

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

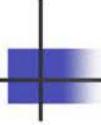
154



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?



Vernetzungskonzepte

1. Vernetzung von Rechnern und Eingabe/Ausgabe
2. Designaspekte effizienter Kommunikation
3. Leistungsentwicklung und Leistungsmaße
4. Netztopologien
5. Einbindung in TCP/IP
6. InfiniBand-Vernetzung

1. Vernetzung von Rechnern und Eingabe/Ausgabe (E/A)

Wozu Vernetzung?

- Die Vernetzung verbindet Rechnerknoten miteinander, E/A-Knoten miteinander und Rechner- mit E/A-Knoten
- Ermöglicht die Interprozesskommunikation
 - Prozesse auf verschiedenen Knoten
- Ermöglicht Prozess-Eingabe/Ausgabe
 - E/A-Hardware meist nicht lokal am Rechenknoten angeschlossen

Rechnerinterne Vernetzung

1. Block Diagram

Figure 1 illustrates the functional block diagram of the motherboard.

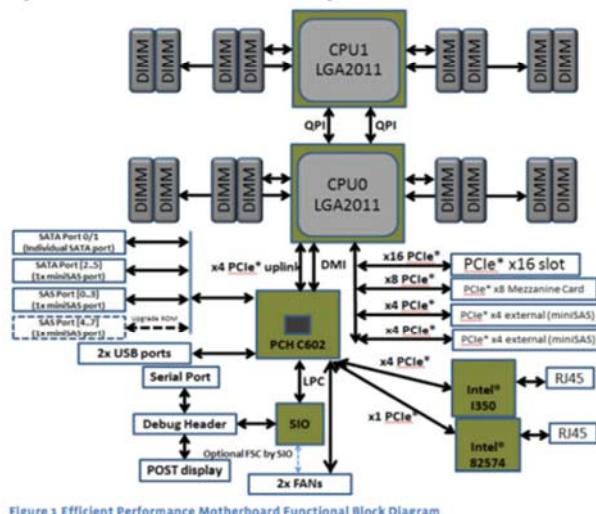


Figure 1 Efficient Performance Motherboard Functional Block Diagram

- Vernetzung auf dem Motherboard mit Intel QuickPath Interconnect
 - Leistet ca. 25GB/s
- Anbindung der externen Komponenten mit PCI
 - PCI Express (PCIe) ca. 63GB/s mit 16 Lanes
- Man erkennt die unterschiedlich langen Wege zu Daten

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

160

Quelle: <https://arstechnica.com/information-technology/2013/02/who-needs-hp-and-dell-facebook-now-designs-all-its-own-servers/>

Siehe auch:

- https://en.wikipedia.org/wiki/File:Motherboard_diagram.svg
- https://en.wikipedia.org/wiki/Intel_QuickPath_Interconnect
- https://en.wikipedia.org/wiki/Conventional_PCI
- https://en.wikipedia.org/wiki/PCI_Express

Im Rechner finden wir verschiedene Bussysteme, die die Komponenten in Verbindung bringen. Softwareseitig verwenden wir zur Interprozesskommunikation Socketprogrammierung basierend auf TCP/IP. Alternativ können gemeinsame Speicherbereiche verwendet werden. Hier gibt es aber keine genormte Programmierschnittstelle.

Prozessor in Mistral: <http://www.cpu-world.com/CPUs/Xeon/Intel-Xeon%20E5-2680%20v3.html>

Vernetzung von Rechnern und E/A

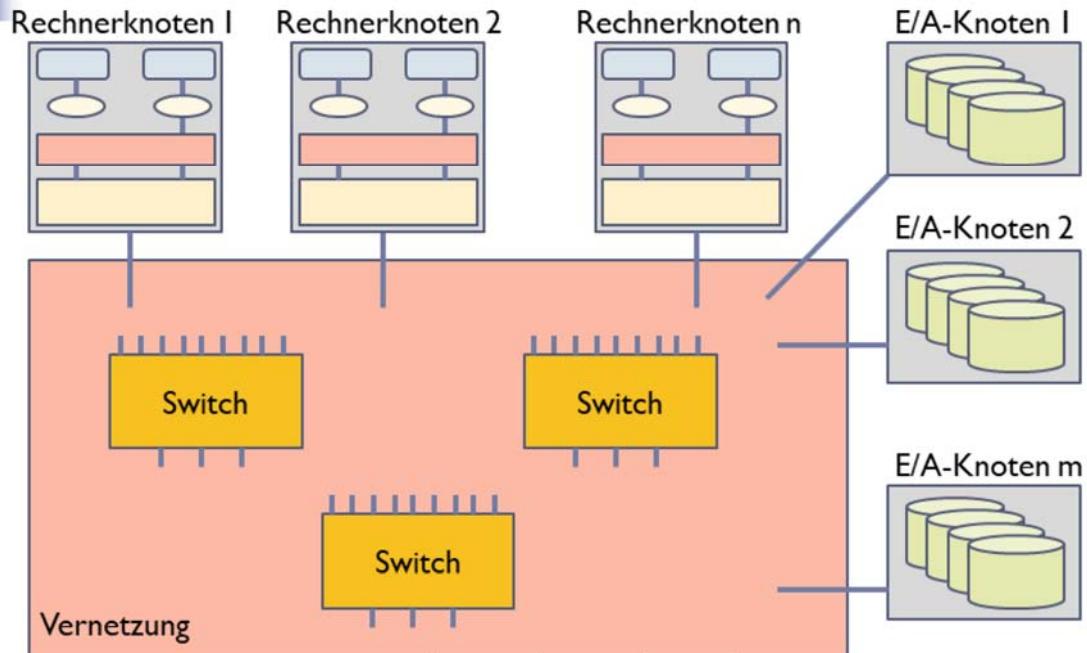
Bestandteile der Vernetzung

- NIC (network interface card) – mindestens eine pro Rechner
- Verbindungsleitungen: Kupfer, Glasfaser
- Switches – zur wechselseitigen Verbindung von Komponenten (Rechnern, E/A-Knoten)

Gegebenenfalls getrennte Vernetzung für

- Prozesskommunikation
- Eingabe/Ausgabe zu Dateisystemen und Bandarchiv
- Wartung und Kontrolle der Rechner

Vernetzung von Rechnern und E/A...



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

162

Die Rechner und die E/A-Komponenten werden über Netze verbunden. Dies können getrennte oder gemeinsam genutzte Netzwerke sein. Die Verknüpfung erfolgt über Switche.

Allgemeine Betrachtungen zur Software

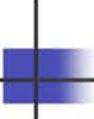
Was interessiert uns an der Vernetzung?

- Programmierschnittstelle hoher Abstraktion
 - Portabilität der Programme wichtig
- Geringe Zusatzlast
- Hardware voll ausnutzbar
- Anpassbar an unterschiedliche Realisierungen der Vernetzungshardware
- Software zum Netzmanagement

Allgemeine Betrachtungen zur Hardware

Was interessiert uns an der Vernetzung?

- Datenraten
- Latenzzeiten
- Bestandteile der Vernetzung
 - Kabel: Kupfer, Glasfaser
 - Netzkomponenten: Karten, Switches
- Ausfallsicherheit
- Adaptive Wegewahl
 - Bei Überlast / bei Fehlern
- Anbindung an TCP/IP



2. Designaspekte effizienter Kommunikation

Überblick über wichtige Designaspekte

- Pufferverwaltung
- Überlappung von Berechnung und Kommunikation
- Realisierung in Hardware
- Datentransport

Pufferverwaltung (1/2)

Warum ist Pufferverwaltung relevant?

- Verwaltung von Speicherplatz ist sehr teuer
- Umkopiovorgänge sind sehr teuer

Aufgabenstellung

- Nachricht steht sendebereit im Adressraum des sendenden Prozesses
- Nachricht durch alle Softwareschichten und das Netz in den Speicher des Empfängers befördern
- Nachricht im Adressraum des Empfängers zur Verfügung stellen

Pufferverwaltung (2/2)

- Zero-Copy-Mechanismen
 - Am besten aus einem Adressraum sofort in den Netzadapter übertragen – schwierig!
- Speicherregistrierung
 - Moderne Vernetzungen gestatten Remote Direct Memory Access (RDMA)
 - Setzt die Registrierung von Speicherbereichen voraus – zeitaufwendig
- Unerwartete Nachrichten
 - Der Empfänger hat keine Kenntnis, dass eine Nachricht eintreffen wird
 - Entsprechend sind keine Puffer allokiert und der Ablauf verlangsamt sich



Überlappung von Berechnung und Kommunikation

Welche Phasen sehen wir bei der Kommunikation?

- Daten aus der Anwendung zum Sendenetzadapter
- Daten vom Sendenetzadapter zum Empfangsnetzadapter übertragen
- Daten vom Empfangsnetzadapter zur Anwendung

Was soll überlappend stattfinden?

- Am besten alle drei Phasen!

Was kann aktuelle Hardware

- Verschieden gute Varianten der optimalen Lösung

Noch problematisch:

- Kann die Software das ausnutzen?

Realisierung in Hardware

Moderne Netztechnologien gestatten die Abarbeitung eines Teils der Software-Schichten in der Adapterhardware (genannt offloading)

Diese Hardware nennt man für TCP/IP:

- TCP/IP offload Engine, kurz ToE

Erhöht gegebenenfalls die Überlappung von Berechnung und Kommunikation

Datentransport

- Zuverlässigkeit: weniger kritisch als in WANs
- Paketgröße und Maximum Transfer Unit (MTU)
 - IP-Paket: max. 64 KB, Ethernet Standard: 1.500 Byte
 - Ethernet verwendet sog. Jumbo-Frames
- DMA-basiert (direct memory access)
 - Entlastet Prozessor
- Unterbrechungen und Polling (Abfrage)
 - Unterbrechungen sind schwergewichtig
 - Polling benötigt Zeit vom Prozessor
 - Wir finden beide Realisierungen

3. Leistungsentwicklung und -maße

In den Jahren von 1977-2020 sehen wir verschiedene Generationen von internen Bussen und externen Netztechnologien

Die Geschwindigkeitssteigerungen sind viel geringer als bei den Rechnern!

- Mindestens eine Größenordnung langsamer

Leistungsentwicklung Bussysteme

Technology	Rate	Year
I ² C	3.4 Mbit/s	425 kB/s 1992 (standardized)
Apple II series (incl. Apple IIGS) 8-bit/1 MHz	8 Mbit/s	1 MB/s ^{[32][33]} 1977
:		
QPI (9.6GT/s, 4.8 GHz)	307.2 Gbit/s	38.4 GB/s 2014
HyperTransport 3.0 (2.6 GHz, 32-pair)	332.8 Gbit/s	41.6 GB/s 2006
HyperTransport 3.1 (3.2 GHz, 32-pair)	409.6 Gbit/s	51.2 GB/s 2008
NVLink 1.0	640 Gbit/s	80 GB/s 2016
NVLink 2.0	1200 Gbit/s	150 GB/s 2017
Infinity Fabric (512 bit * 2s * 2666 MHz) ^[47]	2.730 Tbit/s	341.2 GB/s 2017
Infinity Fabric (Max theoretical)	4.096 Tbit/s	512 GB/s 2017

Englischer Originaltext! „.“ ist unser Komma

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

172

Siehe auch: https://en.wikipedia.org/wiki/List_of_device_bandwidths - für verschiedenste Gerätetypen

Leistungsentwicklung Vernetzungstechniken

Ethernet (1979 -)	10 Mbit/sec
Fast Ethernet (1993 -)	100 Mbit/sec
Gigabit Ethernet (1995 -)	1000 Mbit/sec
ATM (1995 -)	155/622/1024 Mbit/sec
Myrinet (1993 -)	1 Gbit/sec
Fibre Channel (1994 -)	1 Gbit/sec
InfiniBand (2001 -)	2 Gbit/sec (1X SDR)
10-Gigabit Ethernet (2001 -)	10 Gbit/sec
InfiniBand (2003 -)	8 Gbit/sec (4X SDR)
InfiniBand (2005 -)	16 Gbit/sec (4X DDR)
	24 Gbit/sec (12X SDR)
InfiniBand (2007 -)	32 Gbit/sec (4X QDR)
40-Gigabit Ethernet (2010 -)	40 Gbit/sec
InfiniBand (2011 -)	54.6 Gbit/sec (4X FDR)
InfiniBand (2012 -)	2 x 54.6 Gbit/sec (4X Dual-FDR)
25-/50-Gigabit Ethernet (2014 -)	25/50 Gbit/sec
100-Gigabit Ethernet (2015 -)	100 Gbit/sec
Omni-Path (2015 -)	100 Gbit/sec
InfiniBand (2015 -)	100 Gbit/sec (4X EDR)
InfiniBand (2017 -)	200 Gbit/sec (4X HDR)

100 times in the last 16 years

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

173

Quelle: D. Panda, H. Subramoni, S. Chakraborty, Ohio State University,
InfiniBand, Omni-Path, and High-speed Ethernet for Beginners, SC'18,
November 2018.

Leistungsmaße

Wünschenswerte Charakteristika

- Niedrige Latenz (Aufsetzzeit der Nachrichten)
- Hohe Bandbreite
- Geringe Belastung der CPU
- Hohe Bisektionsbandbreite des Netzes

Vorgriff auf Programmierkonzepte

- Möglichst wenig Kommunikation verwendet
- Wenn überhaupt, dann lieber wenige große als viele kleine Pakete

4. Netztopologien

Wünschenswerte Eigenschaften

- Identische Datenraten zwischen beliebigen Knoten
- Vermeidung von Engpässen, Überlastungen, Ausfällen
- Erweiterbarkeit
- Skalierbarkeit
- Gutes Preis/Leistungsverhältnis

Bisektion des Netzes

Bisektionsbreite

Anzahl der Verbindungen, die man trennen muss,
damit das Netz in zwei isolierte Teile mit gleicher
Knotenzahl zerfällt

- Je mehr, desto besser

Bisektionsbandbreite

Aggregierte Bandbreite der durchtrennten Leitungen
(=Fluss an der engsten Stelle)



Bisektion des Netzes

Volle Bisektionsbandbreite

Ein Netz mit n Knoten hat volle Bisektionsbandbreite, wenn die Summe aller Bandbreiten von Verbindungen zwischen zwei beliebigen Hälften des Netzes $n/2$ -mal die Bandbreite einer einzelnen Verbindung ist

Warum interessiert uns das?

- Wir benötigen ausreichend Switches und Wege, um alle Komponenten leistungsfähig zu vernetzen

12.10.2021

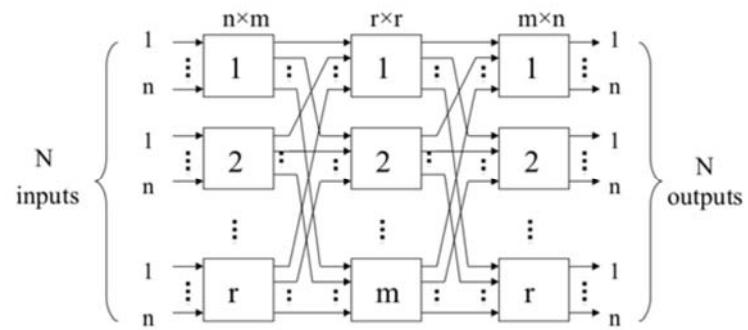
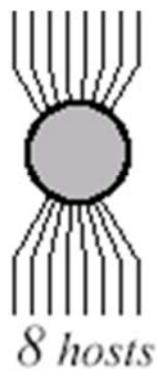
Hochleistungsrechnen - © Thomas Ludwig

177

Wir betrachten einen Switch mit 64 Ports und 1 Gbit/s pro Port. Wenn 32 Sender mit 32 Empfängern kommunizieren wollen, benötigen wir 32 Gbit/s an Bandbreite durch den Switch hindurch. Bidirektional das doppelte. Die interne Topologie des Switches sollte also diese Gesamtleistung ermöglichen. In der Praxis ist sie meist geringer.

Bisektion am Beispiel Myrinet

- Basisbaustein: Kreuzschiene mit 16 Anschlüssen
- Intern: Clos-Netzwerk
 - Benannt nach Charles Clos (1952)



12.10.2021

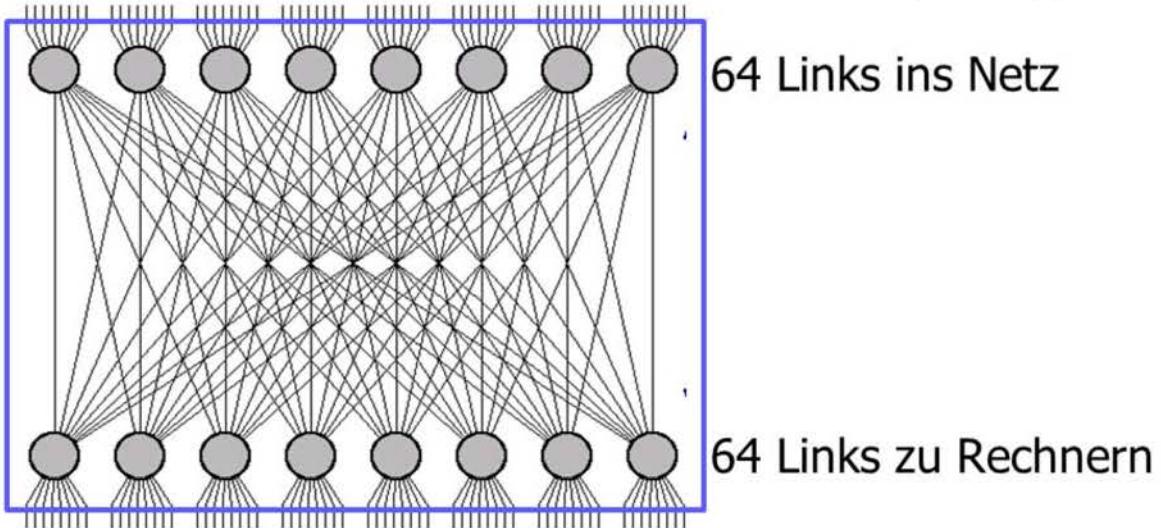
Hochleistungsrechnen - © Thomas Ludwig

178

Siehe: https://en.wikipedia.org/wiki/Clos_network

Einzelner Myrinet-Switch

Neue Basiskomponente aus 16 des bisherigen Typs

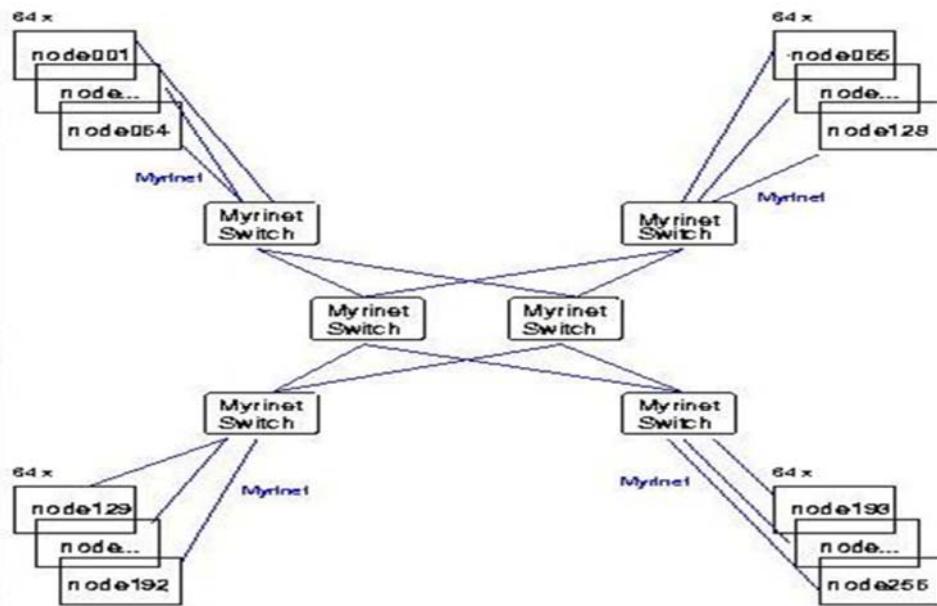


12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

179

Heidelberger Helics-Cluster mit Myrinet



12.10.2021

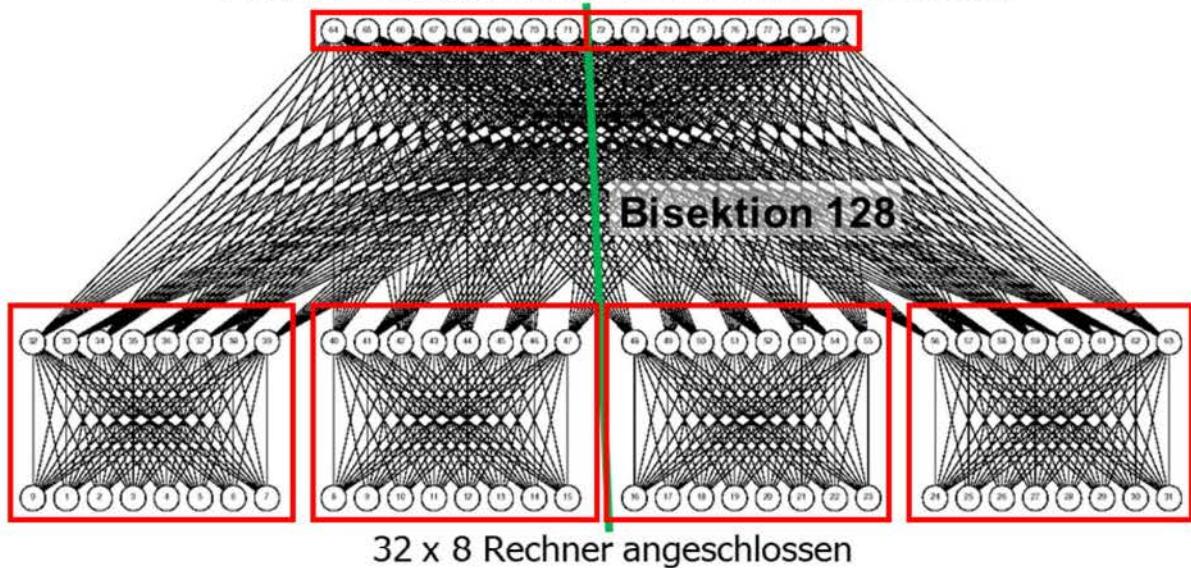
Hochleistungsrechnen - © Thomas Ludwig

180

Es werden 6 Switches vom vorgestellten Typ verwandt. Je einer ist mit 64 Rechnerknoten verbunden. Die 4 Switches für 256 Knoten sind wiederum über 2 Switches miteinander verbunden.

Myrinet im Detail...

4 Clos64-Bausteine und 2 Switches für 256 Rechner



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

181

... und in der Praxis



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

182

Takashi Aoki: Technologies behind the K computer – 5.9.2012

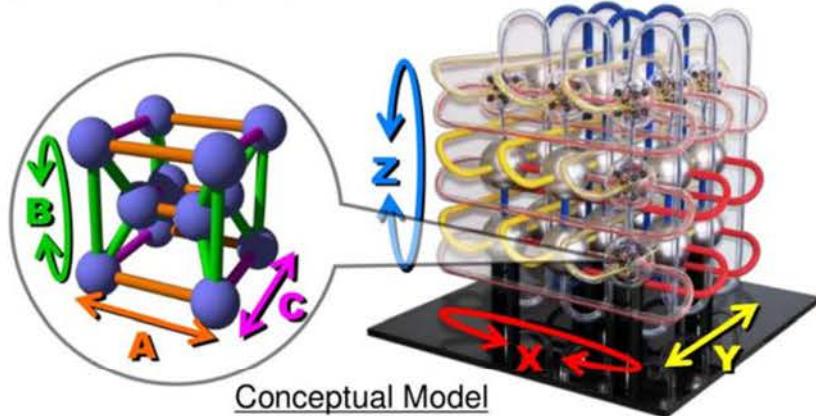
6D-Mesh/Torus Network Topology

FUJITSU

- Higher bisection bandwidth and smaller hops than 3D-Torus

- **Torus fusion**

- ◆ Every XYZ Cartesian grid point has another ABC 3D-Torus
- ◆ X, Z and B are torus (ring) axes
- ◆ A, C and Y are mesh (linear) axes



Conceptual Model

12.10.2021

Sep 5th, 2012 TACC-2012

26/41

Copyright 2012 FUJITSU LIMITED

183

5. Einbindung in TCP/IP

- Aus Sicht des Programms
 - Socket-Programmierung
(Unix-Philosophie: everything is a file)
 - TCP/IP-Schichten im Betriebssystem
- Aus Sicht der Hardware
 - Varianten von Ethernet sehr populär
- Im Rechner-Cluster praktisch immer auch TCP/IP involviert
- Problem: TCP/IP ist schwerfällig

Einbindung in TCP/IP

- Probleme mit TCP/IP
 - Viele Protokollsichten
 - Nicht geringe Prozessorbelastung
 - Integration in das Betriebssystem
 - Kopiervorgänge
 - Anzahl der Unterbrechungen

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

185

In den zahlreichen Protokollsichten werden Aktivitäten ausgeführt, die in einem lokalen Cluster ohne Bedeutung sind, aber Software-Aufwand hervorrufen. Z.B. die Überlaststeuerung zwischen entfernten Knoten.

Die umfangreichen Software-Schichten bedeuten auch eine nicht vernachlässigbare Belastung des Prozessor, der diese ja abarbeiten muss.

Die Integration in das Betriebssystem ist nachteilig, da die Anwendung ja immer erst durch dieses hindurch auf die NIC zugreifen muss, anstatt einen eigenen Weg zu haben.

Beim Durchlaufen der Schichten und beim Übergang zwischen Betriebssystem und Anwendungsprogramm kommt es immer wieder zu zeitaufwendigen Kopiervorgängen.

Aufgrund der maximalen MTU (maximum transmission unit) von 1500 Byte kommt es auch bei GigaBit-Ethernet zu hohen Raten von Unterbrechungen. Größere Frames (Jumbo-Frames) schaffen hier ein wenig Abhilfe.

Einbindung in TCP/IP

- Lösungsansätze
 - TCP-Bypass
 - Definition eines eigenen Paketformats
 - Evtl. Problem: nur cluster-lokal verwendbar
 - TCP Offload Engine
 - Z.B. eine GigE Verbindung kann einen Pentium IV mit 2,4 GHz auslasten
 - Deshalb extra Prozessor für TCP/IP-Protokollstapel
 - Normalerweise auf der NIC untergebracht
 - Kernel Bypass
 - Die NIC kommuniziert direkt mit dem Programm
 - Dies findet man bei allen nicht Ethernet-Vernetzungen!

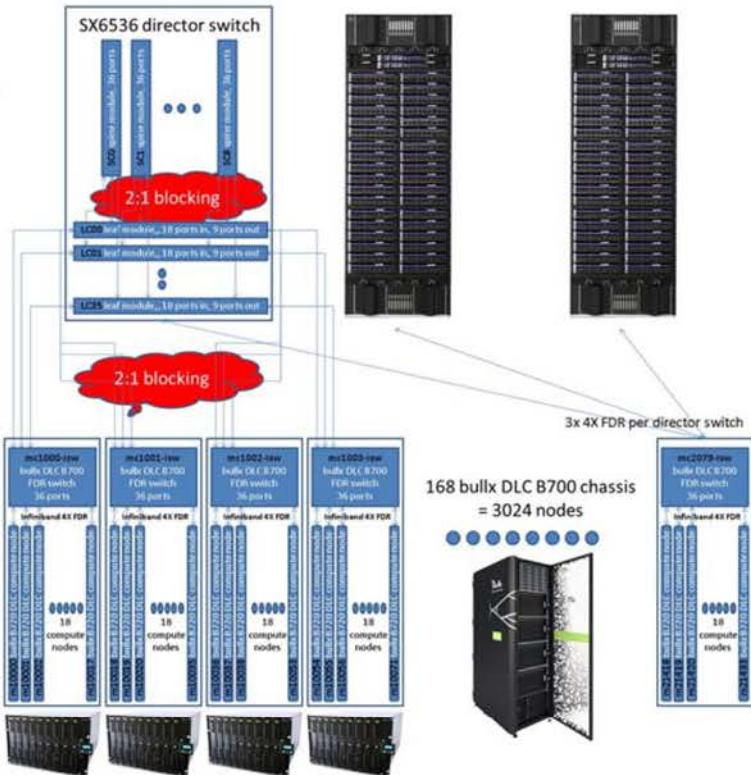
Einbindung in TCP/IP

- Lösungsansätze...
 - Remote Direct Memory Access (RDMA)
 - Die NIC kann direkt in den Speicher eines entfernten Rechners schreiben
 - Zero-Copy Networking
 - Vermeide Kopien zwischen Betriebssystem und Anwendungsprogramm
 - Verwende stattdessen DMA und MMU
 - Interrupt Mitigation
 - Fasse mehrere Unterbrechungen zu einer zusammen

6. InfiniBand-Vernetzung

- InfiniBand ist ein Industriestandard einer Gruppe von Herstellern genannt Open Fabrics Alliance
- Definiert einen Standard für ein Systemnetz
 - Früher auch als Spezifikation eines rechnerinternen Busses bekannt – dieser Ansatz wurde nicht weiter verfolgt
- Unterscheidung zwischen Rechnern und E/A-Geräten
 - Für Rechner verwendet: Host Channel Adapter (HCA)
 - Für E/A-Geräte verwendet: Target Channel Adapter (TCA)
- Kommuniziert am Betriebssystem vorbei: schnell!

Mistral

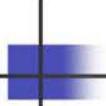


12.10.2021

189

InfiniBand-Software

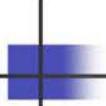
- IP over InfiniBand (IPoIB)
 - Ermöglicht IP-basierten Protokollen eine Kommunikation über InfiniBand
 - Durchsatz von 300 MByte/s ist höher als der von Gigabit-Ethernet (GE)
 - Latenz von 20 µs vergleichbar zu GE
- Sockets direct protocol (SDP)
 - Durchsätze von 900 MByte/s
 - Latenzen von 12 µs



Vernetzungskonzepte

Zusammenfassung

- Die Vernetzung bringt Rechner miteinander und mit den E/A-Geräten in Verbindung
- Vernetzungshardware sind Netzadapter, Kabel, Switches
- Wir möchten hohe Datenraten und geringe Latenzen bei gleichzeitiger guter Skalierbarkeit
- Effiziente Kommunikation umfasst viele Einzelaspekte
- In den letzten 10 Jahren haben sich Rechnergeschwindigkeiten 10x schneller gesteigert als Netzgeschwindigkeiten
- Ein wichtiges Maß der Topologie ist die Bisektionsbandbreite
- TCP/IP ist schwerfällig und wird häufig ersetzt
- InfiniBand ist die aktuelle Hochleistungsvernetzung beim Hochleistungsrechnen



Vernetzungskonzepte

Die wichtigsten Fragen

- Welche Aufgaben hat die Vernetzung?
- Welche Komponenten weist eine Vernetzung auf?
- Welche Fragen finden wir bei der Pufferverwaltung?
- Was bedeutet Überlagerung von Berechnung und Kommunikation?
- Was versteht man unter Hardware-Realisierung von Software?
- Welche Bandbreiten finden wir bei aktuellen Netztechnologien?
- Welche Charakteristiken sollte die Netzhardware aufweisen?
- Was versteht man unter Bisektionsbandbreite?
- Wie umgeht man die Engstelle TCP/IP
- Welche Protokolle finden wir bei InfiniBand?



Hochleistungs-Eingabe/Ausgabe

1. Motivation
2. Abstraktionsebenen
3. Traditionelle und moderne E/A
4. E/A-Klassen bei numerischen Anwendungen
5. Systeme und Schnittstellen
6. Parallel Virtual File System (PVFS)
7. Forschungsthemen

1. Motivation

Hauptspeichergrößen steigen stark an

- Normaler Rechner: einige Gigabyte
 - Z.B. 16 GB Hauptspeicher, 14 TB Festplatte (1:875)
 - 10 GB lesen bei 100 MB/s dauert 100 s
- Hochleistungsrechner: z.B. 2 GB pro Core
 - Mistral: 266 TB Hauptspeicher, 54 PB Plattenspeicher (1:200)
 - DKRZ hat außergewöhnlich gutes Verhältnis!

Motivation...



DATA CENTER EXPLORER

By Andy Patrizio, Network World | DEC 3, 2018 2:50 AM PT

IDC: Expect 175 zettabytes of data worldwide by 2025

By 2025, IDC says worldwide data will grow 61% to 175 zettabytes, with as much of the data residing in the cloud as in data centers.

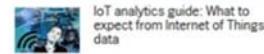


IDC has released a report on the ever-growing datasphere, what it calls the collective world's data, and just like the recent [Cisco study](#), the numbers are staggering. IDC predicts that the collective sum of the world's data will grow from 33 zettabytes this year to a 175ZB by 2025, for a compounded annual growth rate of 61 percent.

12.10.2021

195

RELATED



IoT analytics guide: What to expect from Internet of Things data



What is an SSD? How solid state drives work



10 hot business-continuity startups to watch



VIDEO
Linux tip: Learn to use display release commands

Siehe:

- <https://www.networkworld.com/article/3325397/storage/idc-expect-175-zettabytes-of-data-worldwide-by-2025.html>
- <http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/>
- <https://insidehpc.com/2010/10/14/zettabytes-petabytes-and-all-that-the-intersection-of-supercomputing-and-all-the-data-we-create/>

Speicherung großer Datenmengen

- Datenaufkommen
 - Besonders hoch in den Naturwissenschaften
Klimaforschung, Physik, Biologie, Astronomie, ...
- Zugriff
 - Alle Anwender wollen immer alle Daten aufheben und sie jederzeit zugreifbar haben
- Verfügbarkeit
 - Die Datenspeicherung soll mit Fehlern in der Hardware problemlos umgehen können
- Sicherheit
 - Daten müssen vor Einblick, Veränderung und Löschen geschützt werden

Komplexere E/A-Systeme

- RAID – Redundant Array of Inexpensive Disks
- MAID – Massive Array of Idle Disks
- JBOD – Just a Bunch of Disks

- SAN – Storage Area Network (blockorientiert)
- NAS – Network Attached Storage (dateiorientiert)

- Dateisysteme mit verteiltem Zugriff
 - NFS – Network File System
 - AFS – Andrew File System
- Dedizierte Spezial-Hardware in Hochleistungsrechnern

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

197

Siehe:

- <https://en.wikipedia.org/wiki/RAID>
- <https://en.wikipedia.org/wiki/MAID>
- <https://en.wikipedia.org/wiki/JBOD>
- https://de.wikipedia.org/wiki/Storage_Area_Network
- https://de.wikipedia.org/wiki/Network_Attached_Storage
- https://de.wikipedia.org/wiki/Network_File_System

2. Abstraktionsebenen

Begriff der „parallelen Eingabe/Ausgabe“

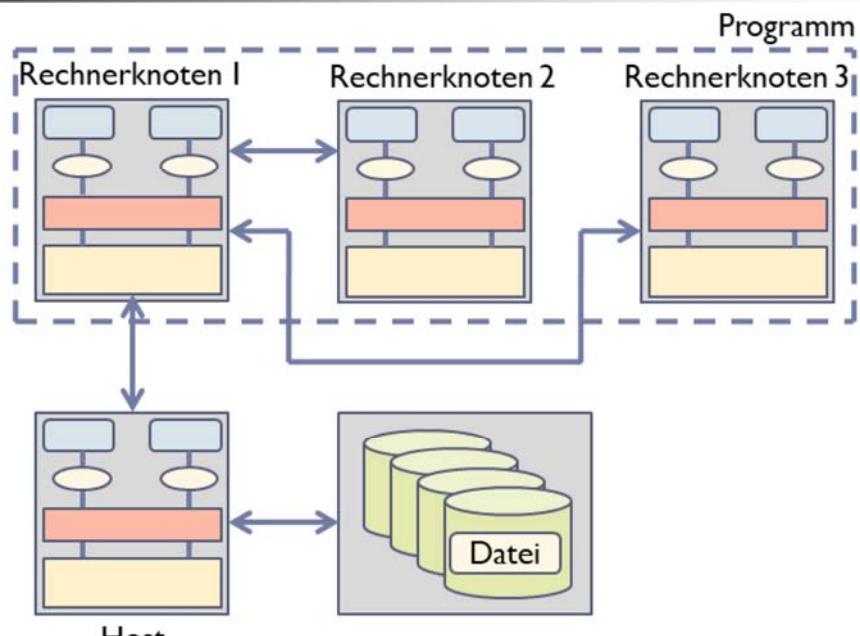
- Programmsicht:
Der Zugriff auf die Bytes *einer* Datei erfolgt aus mehreren parallelen Prozessen heraus
- Systemsicht:
Die Bytes einer Datei liegen über mehrere Platten verteilt

Wie üblich: Ebenen voneinander unabhängig

3. Traditionelle und moderne E/A

- Traditionell
 - E/A nur über Hostrechner
 - E/A nur durch festgelegten Prozess
- Bewertung
 - Aus Effizienzgründen nicht mehr möglich
- Modern
 - E/A über viele Knoten
 - E/A durch alle Prozesse
- Bewertung
 - Verwendung moderner Techniken

Traditionelle E/A



12.10.2021

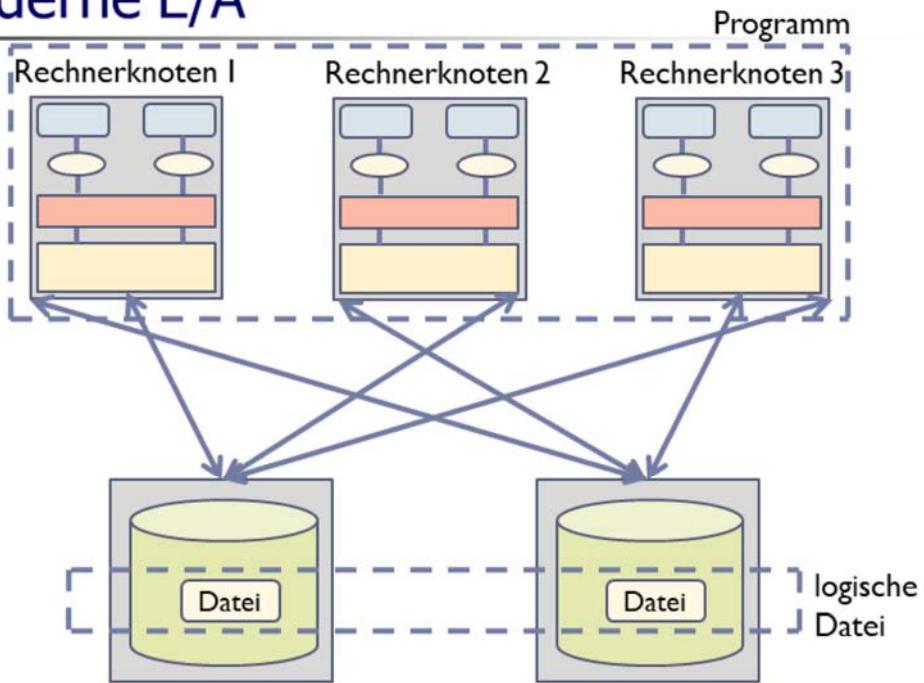
Hochleistungsrechnen - © Thomas Ludwig

200

Ein Hostrechner dient zum Abwickeln der E/A. Seine Anbindung ist der Engpass. Bei einer größeren Anzahl von Rechnerknoten funktioniert das nicht mehr.

Der Inhalt einer Datei liegt in den Prozessen auf den Rechnerknoten, die zusammen das parallele Programm ausmachen.

Moderne E/A



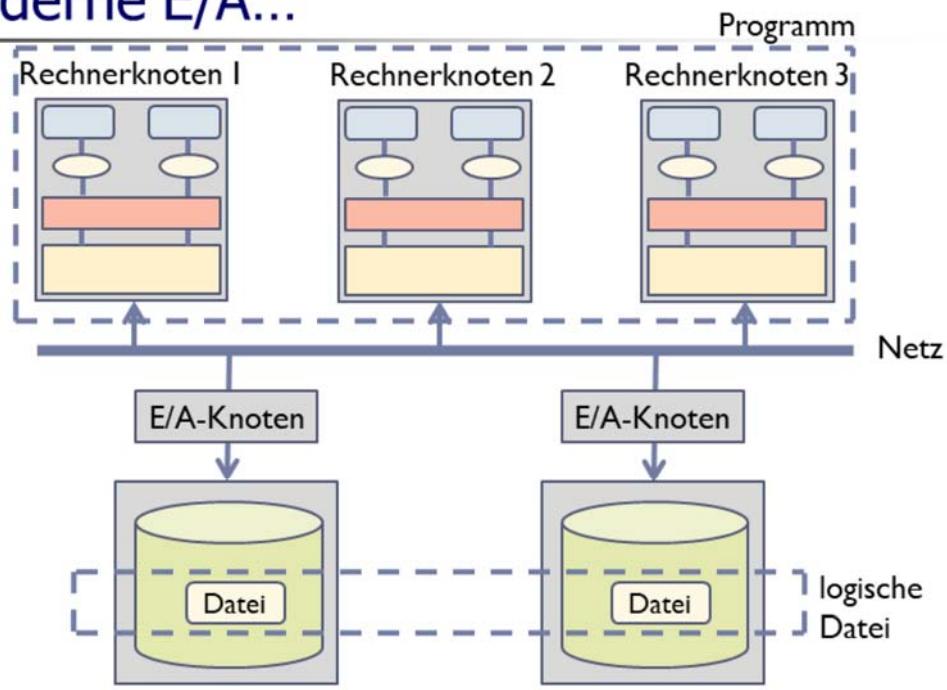
12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

201

Die logische Datei aus Programmsicht ist von ihrem Inhalt her auf physikalische Dateien auf dem parallelen Dateisystem aufgeteilt.

Moderne E/A...



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

202

Wir kommen auf die genaue Abbildung von Byte zwischen den Ebenen später zurück.

Geschichte der parallelen E/A

- Parallele E/A-Systeme und parallele E/A-Bibliotheken entstehen Anfang der 90er Jahre
- Viel Forschung bereits Mitte der 90er
- Nur wenige existierende Systeme im Produktionsbetrieb
- Aber: Sehr viele proprietäre Hochleistungs-E/A-Systeme bei Hochleistungsrechnern; jedoch nicht so oft echt parallele

Kategorien massiver E/A

Anwendungssicht

- Nichtnumerische Anwendungen
 - Unregelmäßig strukturierte Daten
z.B. Datenbank-Anwendungen
 - Datenströme
z.B. Multimedia-Anwendungen
- Beides hier nicht weiter betrachtet
- Numerische Anwendungen
 - Regelmäßig strukturierte Daten
z.B. Vektoren, Arrays mit großen Dimensionen
 - Auch unregelmäßig strukturierte Daten
 - Listen, dünnbesetzte Matrizen

4. E/A-Klassen bei num. Anwendungen

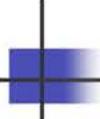
- Lesen der Programmeingabe und Schreiben der Programmausgabe
- Sicherungspunkte
- Temporäre Daten
- „Out-of-core Execution“

Programmeingabe/-ausgabe

- Zeitpunkte
 - Programmstart, Programmende, Zwischenergebnisse
- Datenmengen
 - Maximal Größe des gesamten Hauptspeichers
- Wichtiges Szenarium: Pipelining
 - Daten von einem anderen Gerät
z.B. von physikalischem Experiment
 - Daten zu einem anderen Gerät
z.B. Ergebnisvisualisierung, Datenarchivierung

Sicherungspunkte

- Verwendet zur Programmfortsetzung
 - Nach Absturz oder Unterbrechung
- Sichern aller wichtigen Daten
 - Kompletter Speicherabzug
 - Vom Benutzer ausgewählte Daten
- Anzahl benötigter Sicherungspunkte
 - Mindestens zwei
- Redundanz der Daten notwendig zur Ausfallsicherung



Temporäre Dateien

- Abspeicherung während des Programmlaufs
- Evtl. nicht-parallele E/A auf lokaler Platte ausreichend
- Zur Kommunikation zwischen Prozessen aber parallele E/A erforderlich
 - Verwendung *einer* temporären Datei über alle Platten hinweg

Out-of-Core-Execution

- Out-of-core-Execution:
Bearbeitung einer größeren Datenmenge, als in den
Hauptspeicher passt
 - Eigenprogrammiertes Aus- und Einlagern der
überschüssigen Datenmengen
 - Effizienter als Swapping durch Betriebssystems
- Spezialfall für Spezialanwendungen
 - Bei numerischen Anwendungen wird typischerweise der
Hauptspeicher exakt gefüllt

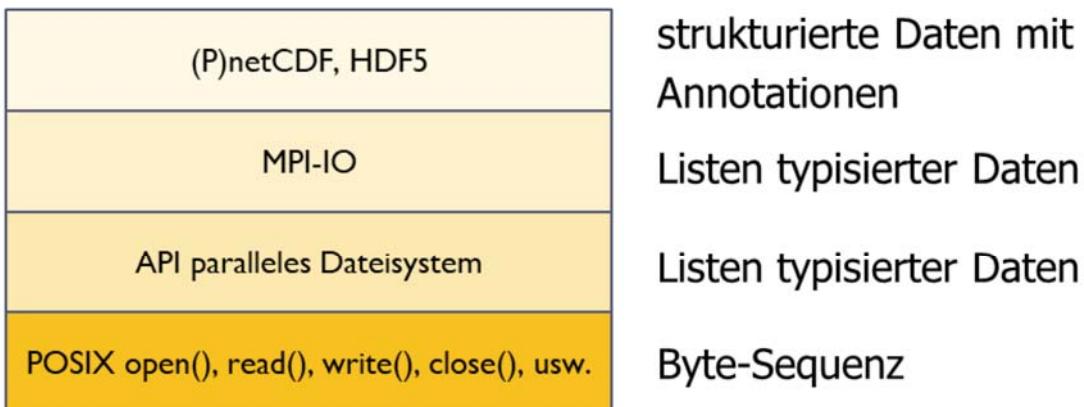
Zugriffsmuster bei E/A

- Wichtig für Fall 1: E/A bei Programmen
- Fragestellung
 - Wann wird E/A durchgeführt?
 - Welche Mengen werden transferiert?
 - Welche Byte einer Datei werden angesprochen?
- Ausführliche Studien aus den 90ern
- Wir müssen hier lernen, was die Klimaforscher tun
- Wichtig um die Abbildung der logischen Datei auf die physikalische zu optimieren

5. Systeme und Schnittstellen

- Hierarchie von Schnittstellen

Im Moment nur paralleles Dateisystem betrachtet



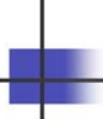
Auf den unterschiedlichen Ebenen werden Objekte verschiedener Abstraktion übertragen. Ganz unten nur einzelne Byte. In den Schichten darüber schreibt/liest man mit einem Aufruf ganze Listen typisierter Daten. Ganz oben gleich komplexe Daten samt ihrer Annotationen.

E/A im Rechnercluster

- Gelegentlich: An jedem Knoten auch eine Platte
 - Entweder nur für temporäre Dateien
 - Oder als richtiges paralleles Dateisystem über alle Platten hinweg
- Alternativen
 - Jeder Rechenknoten ist auch E/A-Knoten
 - Dedizierte E/A-Knoten
 - Balancierung der Rechen- und E/A-Leistung
- Betriebsproblematik
 - Wer darf wann welche Platten benutzen?
(Bisher nur Konzepte für Knotenzuteilung)

E/A im Hochleistungsrechner

- Knoten haben nie Platten
- Ausgewählte Knoten dienen als E/A-Knoten
 - Dicker Netzanschluss
 - Fetter Hauptspeicherausbau
 - Keine Programme auf diesen Knoten
 - Keine eigenen Festplatten
- Wie geht es?
 - E/A-Knoten leitet die gesamte E/A zum E/A-System bestehend aus eigenen Rechnern und sehr vielen Platten



Paralleles Dateisystem

- Merkmale
 - Mehrere Prozesse können gleichzeitig auf dieselben Dateien zugreifen.
 - Die Daten einer Datei liegen physikalisch verteilt
- Schicht
 - Zwischen dem Anwenderprogramm und dem physikalischen E/A-System der E/A-Knoten
 - Somit typische Middleware-Software

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

214

Ein paralleles Dateisystem stellt somit für die Hintergrunddaten so etwas ähnliches dar, wie ein virtuell gemeinsamer Speicher für die Hauptspeicherdaten.

Systeme

- IBM Spectrum Scale
 - Bis Feb. 2015 genannt GPFS (Cluster-Dateisystem)
 - Langjähriges Produkt; ausgereiftes System
- Lustre (Cluster-Dateisystem)
 - Neuerer Ansatz, in dem alles besser gemacht wird
 - Sehr hohe Komplexität!
 - Open-Source und frei erhältlich; viele Varianten
- BeeGFS
 - Entwickelt bei Fraunhofer Institut
- OrangeFS (Paralleles Dateisystem)
 - Bis 2007 entwickelt als PVFS/PVFS2
 - Verbreitetes System bei Selbstbau-Clustern

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

215

Siehe:

- https://en.wikipedia.org/wiki/List_of_file_systems#Distributed_parallel_file_systems
- https://en.wikipedia.org/wiki/IBM_Storage#IBM_Spectrum_Scale.E2.84.A2
- https://en.wikipedia.org/wiki/IBM_General_Parallel_File_System
- [https://en.wikipedia.org/wiki/Lustre_\(file_system\)](https://en.wikipedia.org/wiki/Lustre_(file_system))
- <https://en.wikipedia.org/wiki/BeeGFS>
- <https://en.wikipedia.org/wiki/OrangeFS>
- https://en.wikipedia.org/wiki/Parallel_Virtual_File_System

6. Parallel Virtual File System (PVFS)

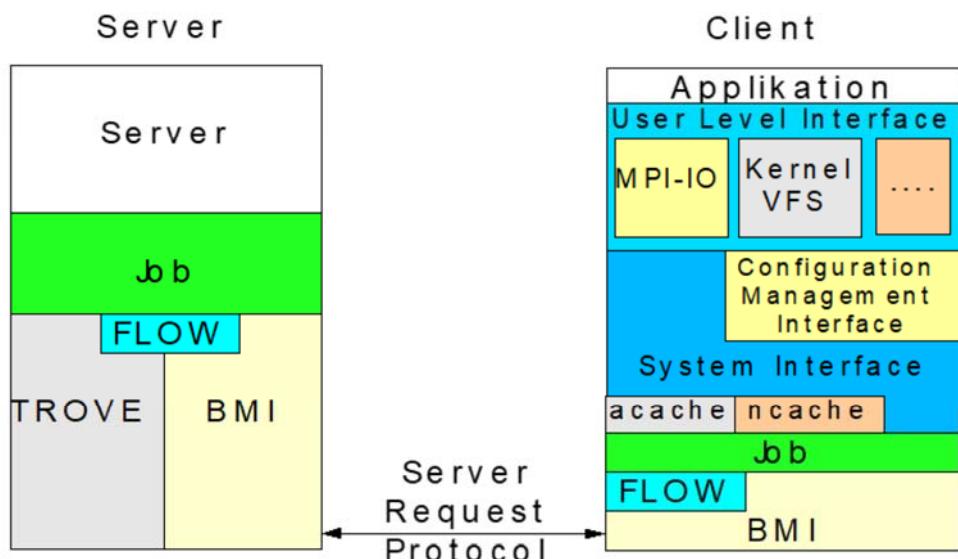
Soll hier beispielhaft dargestellt werden, da „übersichtlich“

- Ziel: E/A massiver Datenmengen in Clustern
- Entwicklergruppen
 - Parallel Architecture Research Laboratory
Clemson University
 - Mathematics and Computer Science Division
Argonne National Laboratories
- Historie
 - Beginn ~1996
 - PVFS2 (Herbst 2003) umbenannt in PVFS 2.0

PVFS2-Eigenschaften

- Komplette Neuentwicklung (ggü. PVFS)
- Setzt auf realem Dateisystem auf (ext[23] o.ä.)
- Modular und hierarchisch aufgebaut
 - Module zum Auswechseln vorgesehen
- Enge MPI-IO-Integration
- Effiziente Durchführung strukturierter nicht-kontinuierlicher Zugriffe
- Zustandsloser Objektzugriff
- Erweiterbare Datenverteilungsfunktion (striping usw.)
- Semantik des Dateisystems variabel
- Explizite Unterstützung paralleler Abläufe
- Redundantes Speichern von Daten und Metadaten

PVFS2-Schichtenmodell



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

218

Die Schicht Job kontrolliert die Abarbeitung einzelner Anfragen in all ihren Phasen. Es sind immer mehrere Schritte in den darunterliegenden Schichten auszuführen.

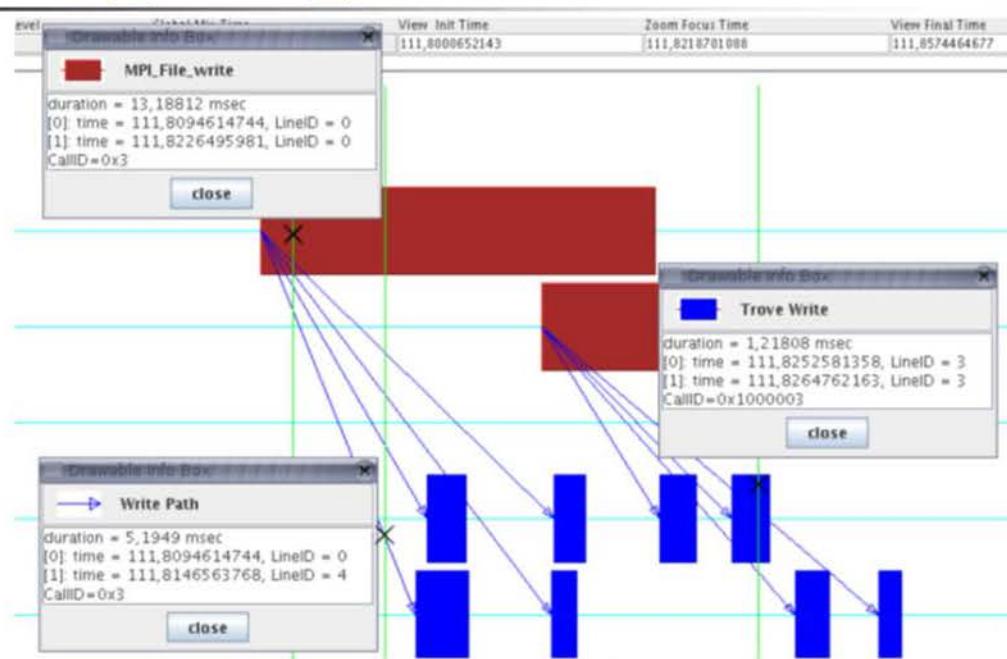
Flow: Steuert den Fluß von Anfragen und Antworten zwischen den tieferen Schichten Trove und BMI.

BMI (Buffered Message Interface): Netzwerk-Abstraktionsschicht. Ermöglicht die Verwendung verschiedener Geräte und Protokolle. Zur Zeit für TCP/IP (Ethernet), Myrinet's GM und Infiniband.

Trove: Abstraktionsschicht für die persistente Datenspeicherung.

Namens-Cache (ncache) und Attribute-Cache (acache).

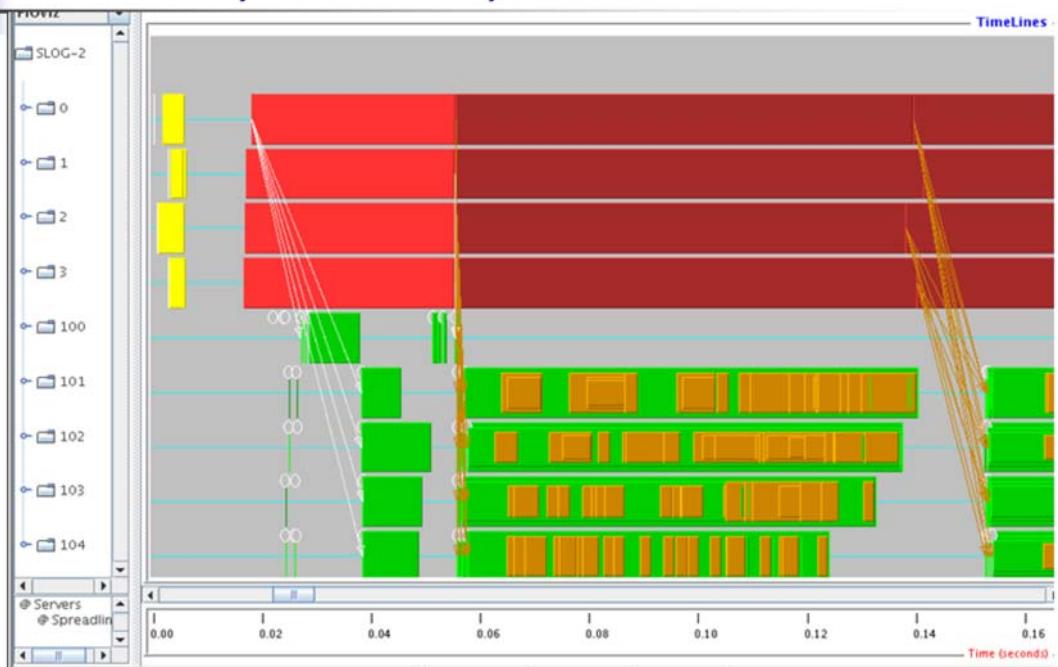
MPI_File_write löst PVFS trove aus



12.10.2021

219

4 Clients, 4 Server, 1 Metadatenserver



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

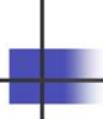
220

Gezeigt ist ein MPI-Programm, das zuerst noch MPI-Kommunikation o.ä. ausführt (gelb). Die vier Prozesse des Programms (Zeilen 0-3) öffnen dann eine gemeinsam genutzte Datei (hellrot). Dabei entsteht zuerst eine Interaktion mit dem Metadatenserver (Zeile 100). Der kontaktiert die einzelnen Datenserver, die lokal eine Datei bei sich öffnen (Zeilen 101-104). Den Erfolg melden sie an den Metadatenserver zurück. Damit ist das Öffnen erfolgreich beendet und es beginnt ein Lesevorgang der Anwendungsprozesse (0-3) in der Zeit 0,06-0,14. Dabei werden Interaktionen mit den Datenservern gestartet und es entstehen Zugriffe auf die lokalen Festplatten (101-104, braun). Nachdem der langsamste der vier Server (101) seine erste Operation beendet hat, ist auch die gesamte MPI-Operation beendet und es wird mit der nächsten fortgesetzt.

7. Forschungsthemen

Der Arbeitsbereich Wissenschaftliches Rechnen
(ehemals Arbeitsgruppe „Parallele und verteilte
Systeme“ an der Universität Heidelberg) verwendet
Lustre als Basis für eigene Forschungs- und
Entwicklungsarbeiten auf dem Gebiet der parallelen
E/A

Themen für Abschlussarbeiten und zur Mitarbeit sind
vorhanden!



Forschungsthemen

- Wie soll parallele E/A genutzt werden?
- Wie steigern wir Leistung und Skalierbarkeit?
- Wie ermittelt man die Leistung des E/A-Systems?

Jetzt neu:

- Kombination mit Bandarchiv und Hierarchical Storage Management (HSM)
- Aspekte der Energie- und Kosteneffizienz

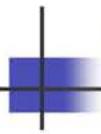
12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

222

Als HSM-System verwendet das DKRZ HPSS (High Performance Storage System)

Siehe: <http://www.hpss-collaboration.org/>



Forschungsthema Nutzung

- Die Frage der Schnittstelle zum Programm ist noch offen
 - Welche Zugriffsvarianten (Semantiken)?
 - Schnittstellen auf welchem Abstraktionsniveau?
- Details im Vortrag zu MPI-IO



Forschungsthema Leistung

- Zugriffsmuster erkennen
- Abbildung logische Daten auf physikalische Daten optimieren oder dynamisch gestalten
- Nichtzusammenhängende Zugriffe optimieren
- Kollektive Operationen unterstützen
- Metadatenzugriff verbessern
- Kompression einbauen

Allgemein: Skalierbarkeit steigern

Kollektive Operationen werden bei MPI-IO besprochen.



Forschungsthema Leistungsbestimmung

- Keine standardisierten Benchmarks
- Was wollen wir messen?
 - E/A-Bandbreiten beim Datenzugriff
 - Verschiedene Zugriffsmuster
 - Verschiedene Datenmengen
 - Unabhängige/identische Orte in Dateien
 - Zugriffsraten beim Metadatenzugriff
- Zur Zeit keine vernünftigen quantitativen Vergleiche zwischen Systemen möglich

Eigene Forschungen

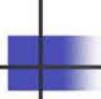
- Leistungsvisualisierung
- Leistungsbewertung
- Modellierung und Simulation
- Metadatenverwaltung
- Anwendung im Produktionsbetrieb
- Zugriffsmuster
- Datenkomprimierung

Mitarbeit ist willkommen!

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

226



Hochleistungs-Eingabe/Ausgabe

Zusammenfassung

- Traditionelle Varianten der E/A jetzt ungenügend
- Parallelisierung der E/A notwendig und möglich
- Parallele E/A etwa ab Mitte der 90er entwickelt
- Uns interessieren hier nur numerische Anwendungen
- E/A gliedert sich in verschiedene Klassen auf
- Benutzerschnittstellen auf verschiedenen Ebenen
- Im Parallelen Dateisystem liegen die Dateien über Platten verteilt und werden von parallelen Prozessen aus gelesen und geschrieben
- Im Einsatz in der TOP500: Lustre, IBM, BeeGFS
- Im Bereich E/A viele offene Forschungsfragen

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

228

Hochleistungs-Eingabe/Ausgabe

Die wichtigsten Fragen

- Warum ist E/A ein sehr wichtiges Thema?
- Wie hantiert man mit Daten in Dateien?
- Welche Abstraktionsschichten unterscheidet man?
- Welche Varianten der E/A finden wir bei numerischen Anwendungen?
- Welche Benutzungsschnittstellen gibt es?
- Wie funktioniert E/A im Cluster?
- Wie funktioniert E/A im Hochleistungsrechner?
- Was ist ein paralleles Dateisystem?
- Wie funktioniert das Parallel Virtual File System?
- Welche offenen Fragestellungen gibt es?

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

229

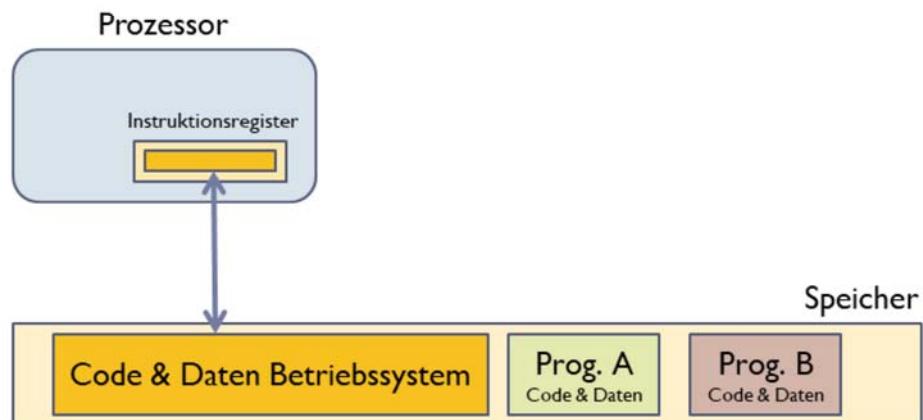


Betriebssystemaspekte

1. Einprozessor-Einkern-Systeme (RIP ☺)
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
(aber vermutlich/hoffentlich noch in allen Betriebssystemlehrbüchern)



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

Einkernprozessoren gibt es noch in Microcontroller-Prozessoren. Siehe z.B.: <https://en.wikipedia.org/wiki/Microcontroller>

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
 - Kommunikation zwischen Prozessen über Nachrichtenaustausch
 - Ressourcen wie z.B. Adressräume, Dateien, Unterbrechungen usw. werden prozessbezogen verwaltet
 - Verwaltet vom BS mittels Prozesskontrollblock
- Thread (in einem Prozess) („Aktionsstrang“)
 - So eine Art Unterprozess im Prozess
 - Threads eines Prozesses teilen sich den Adressraum dieses Prozesses und können darin allen möglichen Unsinn anstellen
 - Threads von verschiedenen Prozessen sind natürlich gegeneinander geschützt – haben aber natürlich auch keine gemeinsamen Variablen
 - Verwaltet vom BS mittels Threadkontrollblock
- Betriebssystem
 - Bodensatz an Code, der alles Obige organisiert und am Laufen hält
 - NICHT als Prozesse oder Threads organisiert ! (zumindest nicht in Linux)

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

232

Siehe:

- https://en.wikipedia.org/wiki/Computer_program
- https://en.wikipedia.org/wiki/Process_%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**)

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

233

Siehe:

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

Siehe:

- 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)

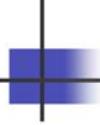
- Prozess A will etwas von Datei 1 lesen
- A ruft read() und geht in Systemmodus
- BS legt Puffer 1 an, sendet Befehl an Platte und legt A schlafen
- Neuer lauffähiger Prozess wird ausgesucht: B
- Prozess B will etwas von Datei 2 lesen
- B ruft read() und geht in Systemmodus
- BS legt Puffer 2 an, sendet 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

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

235

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 ...

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

236

Siehe:

- 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 legt Puffer 1 an, 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 legt Puffer 2 an, 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

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

238

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
- Ununterbrechbarkeit des Kerns ist starker Schutz

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

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

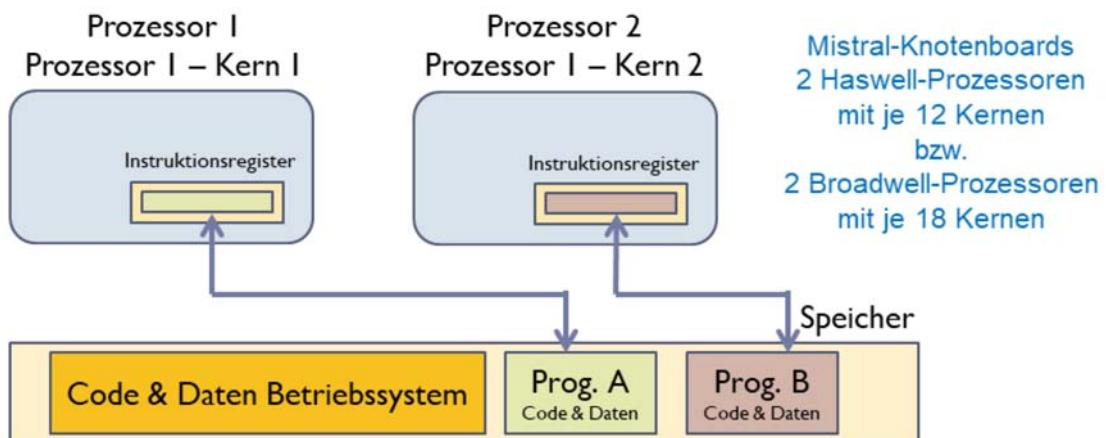
239

Siehe:

- https://en.wikipedia.org/wiki/Realtime_operating_system

2. Mehrprozessor/Mehrkern-Systeme

Mehrkern-Systeme sind seit ca. 2005 Standard
(aber vermutlich nicht in allen Betriebssystemelehrbüchern)



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

240

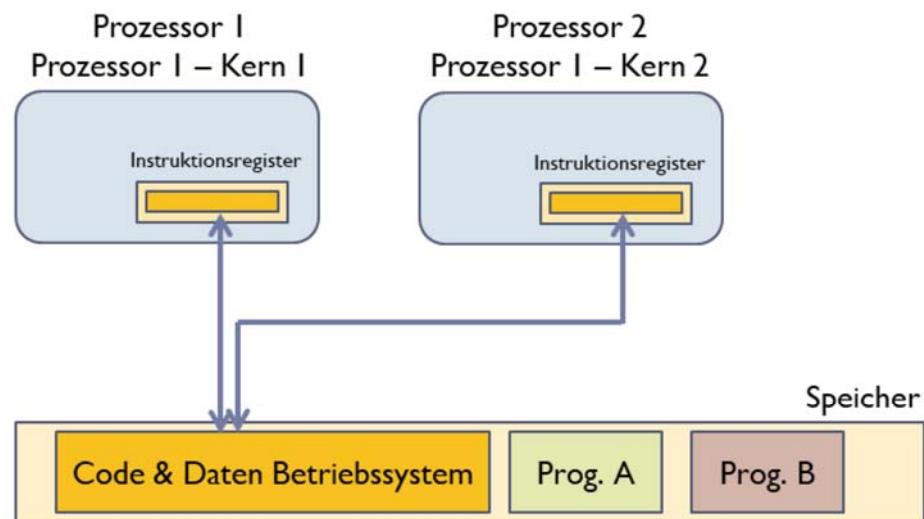
Siehe:

- https://en.wikipedia.org/wiki/Multi-core_processor

Mehrprozessorarchitekturen mit vielen Einkernprozessoren gab es schon in den 90ern. Identisch problematisch zu Mehrkernprozessoren aus Sicht des Betriebssystems.

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

Mehrprozessor/Mehrkern-Systeme...



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

241

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!

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

242

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)

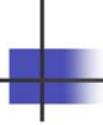
- Prozess A **auf Prozessorkern 1** will etwas von Datei 1 lesen
- A ruft read() und geht in Systemmodus
- Echt gleichzeitig: Prozess B auf Prozessorkern 2 will etwas von Datei 2 lesen
- B ruft read() und geht in Systemmodus
- Ab hier Gefahr, dass beide Prozessorkerne identischen **Code** echt gleichzeitig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

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

243

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

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

244

Siehe:

- 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)

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

245

Siehe:

- https://en.wikipedia.org/wiki/Symmetric_multiprocessing
- https://en.wikipedia.org/wiki/MultiProcessor_Specification
- https://en.wikipedia.org/wiki/Intel_APIC_Architecture
- <https://en.wikipedia.org/wiki/X2API>
- 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

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

247

Siehe:

- https://en.wikipedia.org/wiki/Concurrency_%28computer_science%29
- 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 durch die Sperren
- Ganz viele sehr kurze Sperren:
Daten bzgl. der Konsistenz geschützt
 - Bessere Nebenläufigkeit bei höheren Kosten durch die Sperren

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

248

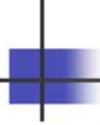
Siehe:

- 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

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

250

Siehe:

- https://en.wikipedia.org/wiki/Linux_kernel#Architecture

5. Synchronisationsmechanismen

- Es müssen jetzt verschiedenste Datenstrukturen geschützt werden, die bei einem einzelnen Prozessor unkritisch waren
 - Z.B. die Zeiger, die auf freie Speicherbereiche zeigen, die vom BS für verschiedene interne Zwecke verwendet werden können
- Einfache Flags reichen nicht aus, da diese gleichzeitig manipuliert werden könnten
- Hardware-Unterstützung ist unabdingbar

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

251

Siehe:

- 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

muss atomar sein!

- **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

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

252

Siehe:

- https://en.wikipedia.org/wiki/Atomic_operation

Atomar und unteilbar bedeutet hier vor allem auch: wenn Sie es in C oder Fortran im Anwenderprogramm programmieren, ist es falsch.

Synchronisation mittels Semaphor

Standardverfahren:

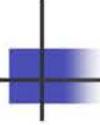
- P() dekrementiert Semaphor und blockiert, wenn Wert kleiner 0 wird
- V() inkrementiert Semaphor und weckt Thread auf, wenn Wert kleiner oder gleich 0 ist

BS-Kern garantiert Atomarität der Aktion

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

Siehe:

- 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

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

255

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;  
}
```

muss atomar sein!

Möglicher Nachteil: Busblockierung

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

256

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

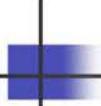
Ein Prozess oder ein Thread sollte auf dem Prozessorkern fortgesetzt werden, auf dem er zuletzt lief

Grund: Gültiger Inhalt im Cache des Prozessorkerns

- Benötigen wir bei der Programmierung mit gemeinsamen Speicher und muss man quasi aktivieren. Ist einfach.

Siehe:

- https://en.wikipedia.org/wiki/Processor_affinity
- 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

12.10.2021

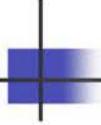
Hochleistungsrechnen - © Thomas Ludwig

261

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?



Parallele Programmierung

1. Was ist Parallelisierung?
2. Paradigmen der parallelen Programmierung
3. Werkzeuge zur Parallelisierung
4. Algorithmische Aspekte
5. Beispiele

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

263

1. Was ist Parallelisierung?

Aufgabe

- Finde impliziten Parallelismus im Algorithmus und mache ihn explizit
- Mittel: Verteile Programme und Daten auf die Ressourcen des Systems
- Wer? Was?
 - Programmierer und/oder Compiler und/oder Laufzeitsystem



Was ist Parallelisierung...

- Verteilung erzeugt neue Last (*overhead*)
 - Minimale Zusatzlast wenn nicht verteilt
- Verteilung nutzt Ressourcen optimal
 - Optimale Leistung wenn vollständig verteilt

Zielvorgabe

Nutze alle Ressourcen und minimiere die Zusatzlast

Anforderungen

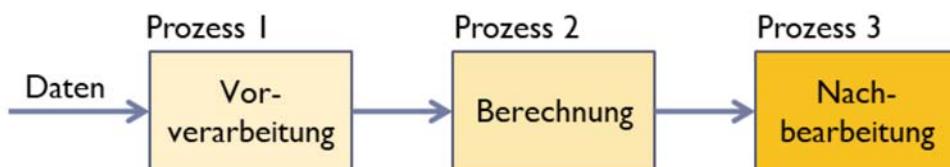
- Zusätzlich zur sequentiellen Software
 - Aufteilung des Programms und/oder der Daten in kleine Teile (*partitioning*)
 - Hinzufügen von Koordination und Kommunikation
 - Abbildung der Teile auf die Komponenten des Computers (*mapping*)
- Probleme
 - Fehlersuche (neue Fehlerarten)
 - Leistungsanalyse (Programmbeeinflussung)
 - Lastausgleich (für optimale Leistung)

2. Paradigmen der Parallelisierung

Code-Aufteilung (auch: Macro-Pipelining)

Verteile den Programmcode über die Knoten

- Unterschiedlicher Code auf jedem Knoten
- Daten variieren gemäß dem Fluss der Berechnung
- Koordinator: erster/letzter Prozess



Siehe:

- https://en.wikipedia.org/wiki/Task_parallelism



Paradigmen der Parallelisierung...

- **Vorteile** der Code-Aufteilung
 - Manchmal sehr einfach zu entwerfen
 - Passende Algorithmen existieren (z.B. FFT)
- **Nachteil** der Code-Aufteilung
 - Mehrere Quellcode-Dateien nötig
 - Schwierig an Zielmaschine anpassbar
 - Komplexe Kommunikationsschemata
 - Schwierige Fehlersuche
 - Schwieriger Lastausgleich

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

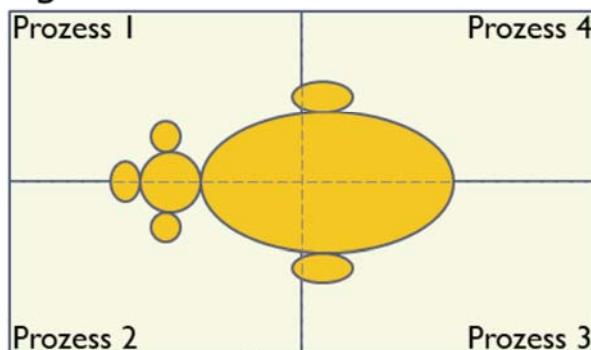
268

Paradigmen der Parallelisierung...

Datenaufteilung

Verteile die Daten auf die Knoten

- Identischer Code auf jedem Knoten
- Daten variieren von Knoten zu Knoten
- Durch ausgewählten Prozess koordiniert



12.10.2021

269

Siehe:

- https://en.wikipedia.org/wiki/Data_parallelism

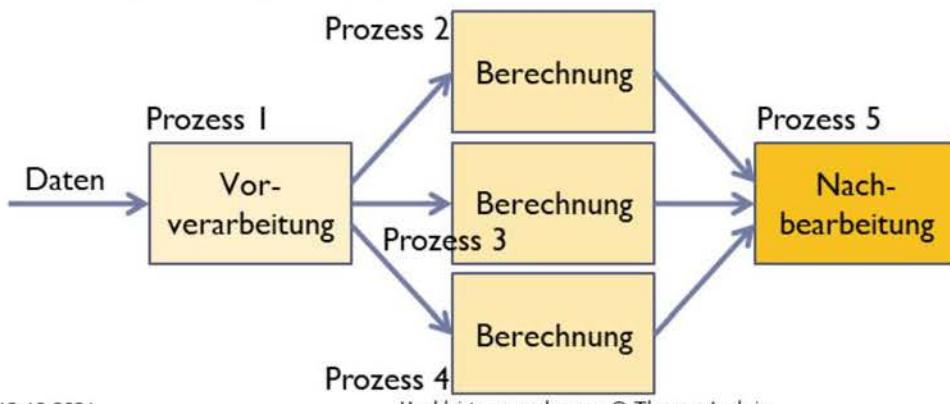
Paradigmen der Parallelisierung...

- **Vorteile** der Datenaufteilung
 - Leicht programmierbar: nur ein Quellcode
 - Einfach an Zielmaschinen anpassbar
 - Sehr oft reguläre Datenaustauschschemata
 - Einfachere Fehlersuche
- **Potentieller Nachteil** der Datenaufteilung
 - Manchmal dem Algorithmus nicht angemessen
- Datenaufteilung ist **Quasistandard**
 - Genannt **SPMD** (single program, multiple data)

Paradigmen der Parallelisierung...

Gemischte Code- und Datenaufteilung

- Dies ist unsere Zielvorstellung
 - Vereint Vorteile beider Ansätze
 - Benötigt Mehrprozessbetrieb auf den Knoten



12.10.2021

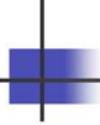
Hochleistungsrechnen - © Thomas Ludwig

271

3. Werkzeuge zur Parallelisierung

Überblick

- Automatisch parallelisierende Compiler
- Manuell parallelisierende Compiler
- Parallelisierung durch Laufzeitumgebung
- Parallele Sprachen und parallele Erweiterungen zu existierenden sequentiellen Sprachen
- Parallele Programmierbibliotheken für existierende sequentielle Sprachen



Automatisch parallelisierende Compiler

Parallelisierung auf Schleifenebene (Fortran)

- Compiler analysiert Datenabhängigkeiten
- Erkennt parallel ausführbare Schleifenindizes und verteilt sie
- Compiler-Pragmas notwendig (Steueranweisungen)
- Normalerweise schlechte Leistungsausbeute

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

273

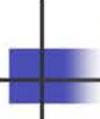
Siehe:

- https://en.wikipedia.org/wiki/Automatic_parallelization



Manuell parallelisierende Compiler

- Wichtiger neuer Ansatz: OpenMP
- Parallelisierung für Systeme mit gemeinsamem Speicher
- Übersetzung durch spezielle Kommentare kontrolliert
- Zusätzliche Verwendung von Bibliotheken



Parallelisierung durch Laufzeitumgebung

- Anwender übergibt dem System eine (höhere) Anzahl von Einzelaufträgen
- Laufzeitsystem schiebt mittels dynamischem Lastausgleich die Aufträge auf unbelastete Rechnerknoten
- Nur geeignet für grobgranulare Parallelisierung auf der Ebene der Aufträge



Parallele Sprachen & Spracherweiterungen

Ansatz: High Performance Fortran (HPF)

- Entworfen 1990+ durch ein Konsortium
- Schwindende Bedeutung für das Hochleistungsrechnen

Problem: Parallelisierungs-Qualität der Compiler

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

276

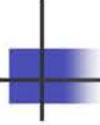
Siehe:

- https://en.wikipedia.org/wiki/Parallel_programming_model

Parallele Programmierbibliotheken

Sind der Standard im Hochleistungsrechnen

- Konzept
 - Starten autonomer Prozesse (*spawning*)
 - Kooperation mittels Nachrichtenaustausch
- Beispiele
 - Parallel Virtual Machine (PVM) [veraltet]
 - Message Passing Interface (MPI)



Software/Hardware-Wechselspiel

Prinzipiell sind alle Programmierkonzepte auf allen Architekturen einsetzbar

In der Realität nutzen wir aus Effizienzgründen

- Bibliotheken zum Nachrichtenaustausch für Architekturen mit verteiltem Speicher
- Threads und automatische/manuelle Parallelisierung für Architekturen mit gemeinsamem Speicher

4. Algorithmische Aspekte

Zweigeteilte Welt des Programmierers

- Numerische Algorithmen
 - Grand Challenge-Algorithmen: Wettervorhersage, Klimabestimmung, Proteindesign usw.
- Nichtnumerische Algorithmen
 - Suchverfahren: Theorembeweiser, Spieleprogramme usw.
 - Datenbank-Anwendungen

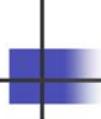
12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

279

Siehe:

- https://en.wikipedia.org/wiki/Grand_Challenge
- https://en.wikipedia.org/wiki/Theorem_prover



Numerische Algorithmen

Strömungsmechanik (*Computational fluid dynamics, CFD*), numerische Berechnungen, Optimierungen, Simulationen usw.

- Iterative Algorithmen
- Beenden aufgrund einer globalen Bedingung
- Reguläre Datenstrukturen (Vektoren, Felder)
- Reguläre Kommunikationsstrukturen
- Statische Prozessstruktur

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

280

Siehe:

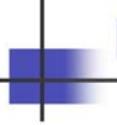
- https://en.wikipedia.org/wiki/Computational_fluid_dynamics



Nichtnumerische Algorithmen

Datenbankanwendungen, künstliche Intelligenz

- Suchbaumverfahren
- Irreguläre Kommunikationsstrukturen
- Irreguläre Datenstrukturen (dynamisch, *garbage collection*)
- Dynamische Prozess/Thread-Struktur



Eine erste Zusammenfassung

- Paradigmen der Parallelisierung
 - Datenaufteilung, Code-Aufteilung
- Werkzeuge zur parallelen Programmierung
 - Compiler und Bibliotheken
 - Das wichtigste: natürliche Intelligenz
- Zweigeteilte Welt
 - Numerische / nichtnumerische Algorithmen

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

282

Siehe:

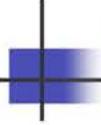
- <https://en.wikipedia.org/wiki/Paradigm>

5. Beispiele von Parallelisierungen

Drei Beispiele

- Numerische Anwendung
 - Diskussion bzgl. der Aufteilung und der Speicherarchitektur
- Strömungsmechanik (CFD)
 - Diskussion bzgl. der zu verteilenden Objekte
- Suchbaumverfahren
 - Diskussion bzgl. der Speicherarchitektur

Alle Beispiele manuell parallelisiert



Beispiel 1: Numerik (1/9)

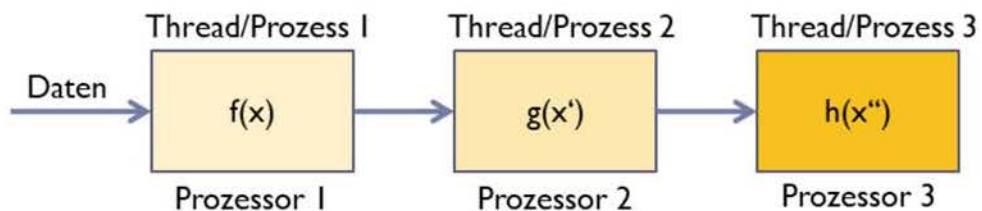
- Drei Funktionen $f()$, $g()$ und $h()$
- Wende Funktionen auf eine Menge von Werten an und berechne Ergebnis $h(g(f(x)))$
- Wir betrachten vier Fälle:
 - Code-Aufteilung / Datenaufteilung
 - Gemeinsamer Speicher / verteilter Speicher

Vereinfachende Annahme für dieses illustrierende Beispiel: bei verteiltem Speicher hat jeder Rechnerknoten nur einen Prozessor mit einem Kern (prä-2005-Architektur)

Beispiel 1: Numerik (2/9)

Code-Aufteilung

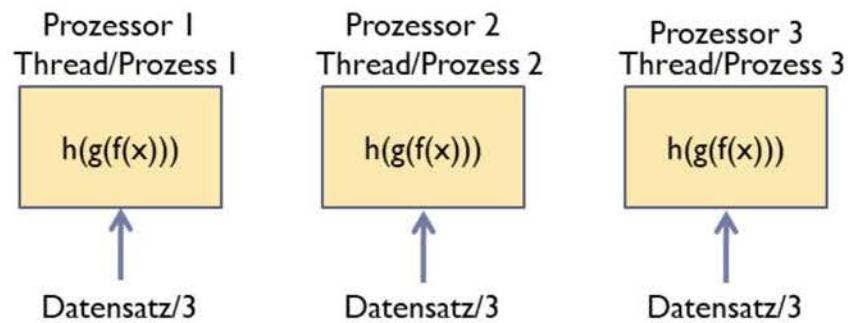
- Verteile drei Funktionen auf drei Knoten
- Arbeitet im Makro-Pipelining-Modus
- Wertemenge ist Eingabedatensatz
- Nur drei Prozessoren sinnvoll nutzbar



Beispiel 1: Numerik (3/9)

Daten-Aufteilung

- Repliziere die Funktionen auf den Knoten
- Verteile die Wertemenge auf die Knoten



Beispiel 1: Numerik (4/9)

Code-Aufteilung mit **gemeinsamem** Speicher

- Basis-Implementierung
 - Werte in Vektor gespeichert
 - Drei Threads, jeder auf eigenem Prozessor
 - Jeder Thread berechnet eine Funktion f, g, h
 - Vektoreinträge werden durch Berechnungsergebnisse ersetzt
 - Eine Variable kennzeichnet den aktiven Thread
- Nachteil
 - Dies ist **kein** paralleles Programm (offensichtlich)!

Beispiel 1: Numerik (5/9)

Korrektur der Basis-Implementierung

- Zähler für Thread [i], der seine Position anzeigt
- Thread [i] berechnet k Elemente und verschiebt danach seinen Zähler um k
- Thread [i] darf bis zum Zählerstand von Thread [i-1] -1 vorrücken

- Vorteil: Gute parallele Implementierung
- Nachteil: Am Anfang und Ende nicht mehr voll parallel
- Achtung: k geschickt wählen!

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

288

Offensichtlich gilt folgendes: Setzt man $k=1$, so wird ständig auf die Koordinationsvariable zugegriffen. Zuviel Koordinationsarbeit, zu wenig mathematische Arbeit. Setzt man $k=\text{Vektorlänge}$, so erhält man das vorher genannte sequentielle Programm. Dazwischen liegt irgendwo ein Optimum, das man für jedes Programm und jede Maschine ermitteln muss. Nachteil für $k>1$: man erhält einen Pipeline-Effekt und die Pipeline muss sich anfangs füllen, erst dann laufen alle Prozessoren parallel. Am Ende der Berechnung läuft sie leer und nicht mehr alle laufen parallel.

Beispiel 1: Numerik (6/9)

Code-Aufteilung mit **verteiltem** Speicher

- Basis-Implementierung
 - Werte in Vektor abgespeichert
 - Drei Prozesse, jeder läuft auf eigenem Prozessor
 - Jeder Prozess berechnet eine der Funktionen f, g, h
 - Prozess [i] berechnet alle Zwischenergebnisse und sendet Vektor an Prozess [i+1]
- Nachteil
 - Dies ist wieder **kein** paralleles Programm!

Beispiel 1: Numerik (7/9)

Korrektur der Basis-Implementierung

- Prozess [i] sendet k berechnete Werte sofort zu Prozess [i+1]

- Vorteil: Gute parallele Implementierung
- Nachteil: Am Anfang und Ende nicht mehr voll parallel
- Achtung: k geschickt wählen!

Wiederum gilt: setzt man $k=1$, so wird zu häufig gesendet und man verliert Zeit. Setzt man $k=\text{Vektorlänge}$ so entsteht ein sequentielles Programm. Das Optimum liegt dazwischen und muss ermittelt werden. Auch hier tritt ein Pipelineeffekt auf.

Beispiel 1: Numerik (8/9)

wichtiger
Lerninhalt

Datenaufteilung mit **gemeinsamem** Speicher

- Basis-Implementierung
 - Werte auf drei Blöcke aufgeteilt
 - Drei Threads berechnen je $h(g(f(x)))$ pro Block
 - Eine Variable pro Block signalisiert das Ende der Berechnung
 - Ausgewählter Thread organisiert Ausgabe der Ergebnisse
- Vorteile
 - Gute parallele Implementierung
 - Keine Zugriffskonflikte bei den einzelnen Werten
- Nachteil (gering)
 - Potentieller Zugriffskonflikt auf Binärkode der Threads

Beispiel 1: Numerik (9/9)

wichtiger
Lerninhalt

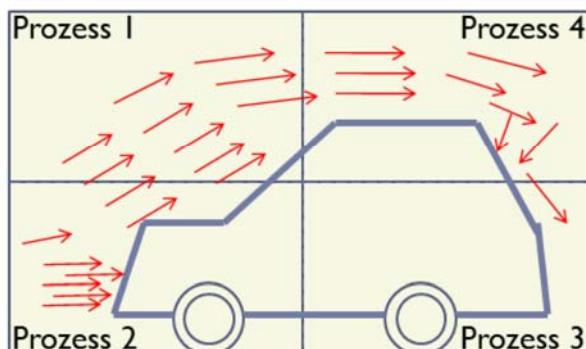
Datenaufteilung mit **verteiltem** Speicher

- Basis-Implementierung
 - Werte sind auf die Knoten verteilt, jeder hat ein Drittel
 - Drei Prozesse auf drei Knoten berechnen $h(g(f(x)))$
 - Ergebnisse werden einzeln an Prozess 0 gesendet
- Vorteile
 - Gute parallele Implementierung
 - Gutes Kommunikations-/Berechnungs-Verhältnis
- Nachteil
 - Programmierung der Datenverteilung

Beispiel 2: Strömungsmechanik (1/3)

Simulation eines Windkanals

- Iterative Berechnung mit Zeitschritt t
 - Mikroskopischer Ansatz: Berechne Teilchen
Molekulardynamik vs. Monte Carlo
 - Makroskopischer Ansatz: Berechne Verteilung von Druck, Temperatur usw.



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

293

Beim molekulardynamischen Verfahren werden die Flugbahnen und Kollisionen der Teilchen exakt berechnet. Dies ist extrem aufwendig.

Bei den Monte Carlo-Verfahren werden Flugbahnen und Kollisionen nicht exakt berechnet sondern quasi ausgewürfelt (mittels Zufallszahlen). Für bestimmte Konfigurationen des Experiments kommen hier nahezu exakte Ergebnisse heraus, wobei aber die Rechenzeiten dramatisch kürzer sind.

Für den makroskopischen Ansatz kommen sogenannte Gitterfahren zum Einsatz. Hier werden feinmaschige Gitter über das Gebiet gelegt und die Werte an den Kreuzungspunkten der Gitterlinien oder für die Gitterzellen berechnet.

Beispiel 2: Strömungsmechanik (2/3)

Mikroskopischer Ansatz:

Erste Variante: Verteile Teilchen

- Jeder Prozess (Prozessor) verwaltet und berechnet einen Teil der Teilchen
- Nachteil
 - Physikalisch benachbarte Teilchen nur schwer bestimmbar
- Vorteil
 - Gleichverteilung der Anzahl der Teilchen ergibt meist guten Lastausgleich

Beispiel 2: Strömungsmechanik (3/3)

Mikroskopischer Ansatz:

Zweite Variante: Verteile Volumenanteile

- Jeder Prozessor berechnet einen Teil des Volumens
- Nachteil
 - Wechselnde Anzahl von Teilchen führt meist zu schlechter Lastbalance
- Vorteil
 - Benachbarte Teilchen leicht bestimmbar

Beispiel 2a: Klimasimulation

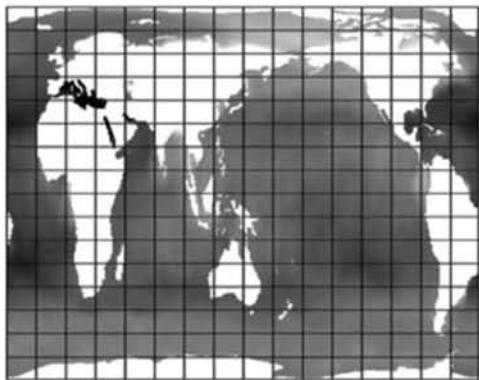
Parallelisierung und paralleler Ablauf durch folgende Schritte

1. Zerlegung des Gebiets in eine sehr große Anzahl von Zellen oder Gitterpunkten (Partitioning)
 - Komplexes Problem der Klimasimulation, der numerischen Modellierung und der Informatik,
2. Bestimmung der zu verwendenden Prozessoranzahl (zum Zeitpunkt des Jobversendens)
3. Zusammenfassung von Zellen oder Gitterpunkten zu Gruppen, damit es genau zur Prozessoranzahl passt (Mapping)
 - Komplexes Problem der Informatik

Beispiel 2a: Klimasimulation...

Beispiel:

- Gittererstellung für ein zu modellierendes Gebiet eines Ozeanmodells



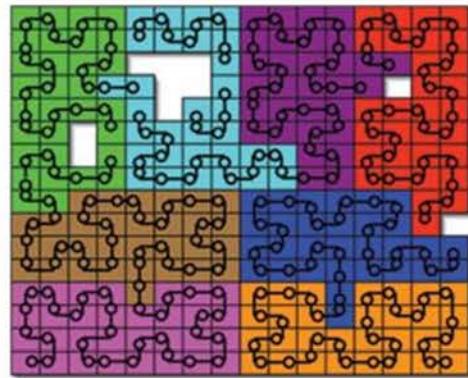
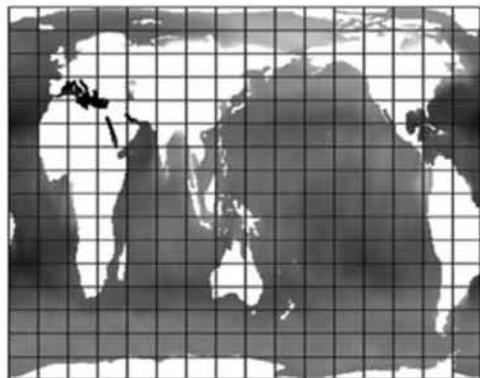
12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

297

Beispiel 2a: Klimasimulation...

- Verwendung von 8 Prozessoren
- Aufsammeln der Zellen im Gitter durch raumfüllende Kurven



Quelle: <http://www.cisl.ucar.edu/research/2006/numerical.jsp>

Hochleistungsrechnen - © Thomas Ludwig

12.10.2021

298

Beispiel 2a: Icosaeder-Gitter (20-Flächer)

ICON (Icosahedral non-hydrostatic) general circulation model

- Max-Planck-Institut für Meteorologie Hamburg
- Deutscher Wetterdienst Offenbach

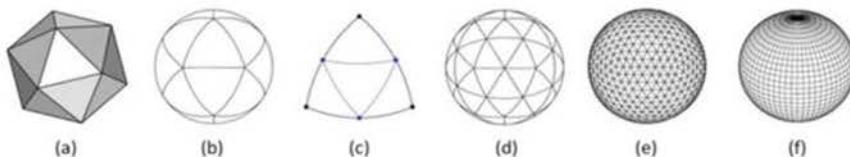


Figure 1 describes the construction of the grids. The icosahedron (a) is projected onto a sphere (b). The edges of each triangle are bisected into equal halves or more generally into n equal sections. The new edge points are connected by great circle arcs to yield 4 (or more generally n^2) spherical triangles within the original triangle (c). After refining the first spherical icosahedrons (d), this method can be extended in the same way. After two more steps the grid is reached (e). Such grids avoid polar singularities of latitude-longitude grids and allow a high uniformity in resolution over the whole sphere (f).

Beispiel 2a: Icosaeder-Gitter (20-Flächer)



Icosaeder-Gitter mit regionalen Verfeinerungen

- blau: global
- grün: nördl. Hemisphäre
- rot: Europa

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

300

Siehe:

- https://en.wikipedia.org/wiki/Principles_of_grid_generation
- https://en.wikipedia.org/wiki/Grid_classification
- <http://www.mpimet.mpg.de/en/communication/news/research-news-overview/icon-entwicklung.html>
- https://de.wikipedia.org/wiki/Platonischer_K%C3%B6rper

Beispiel: Konzepte der Parallelisierung

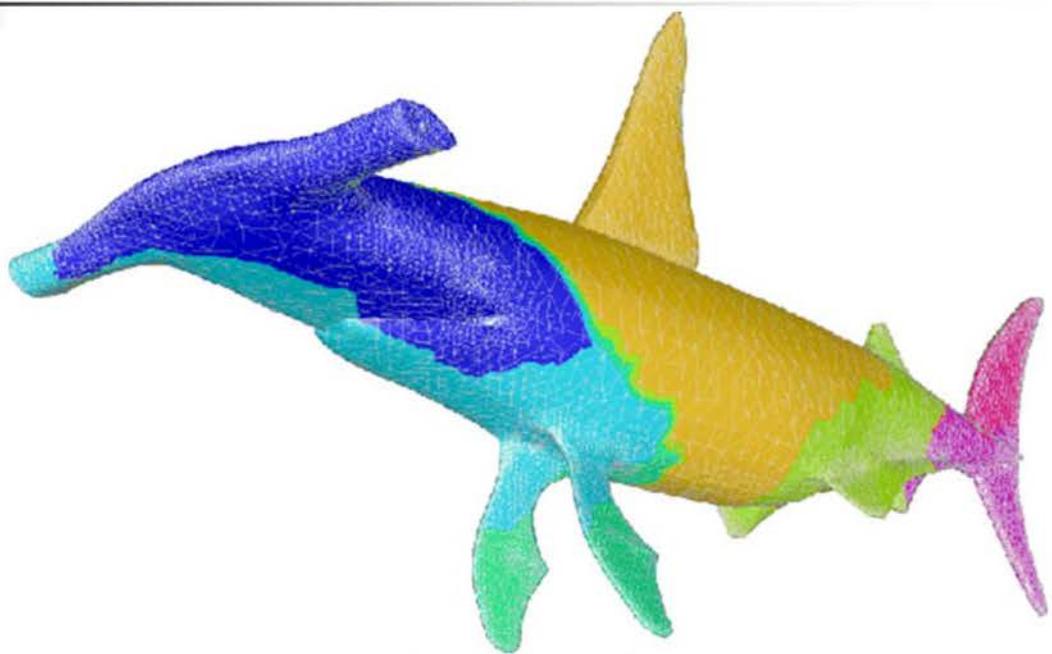
Bei Klima häufig: **Multiple** Program, Multiple Data (MPMD)

Z.B. MPI-ESM (MPI-M Earth System Model) (Stand 2016)

- Atmosphärenmodell: ECHAM (192 Prozesse)
 - Alleine laufend als MPI/OpenMP-Programm
- Ozeanmodell: MPI-OM (63 Prozesse)
- Modellkoppler: OASIS3 (1 Prozess)

Wird abgebildet auf 256 Prozessorkerne

 Zurück zur Strömungsmechanik ☺



12.10.2021

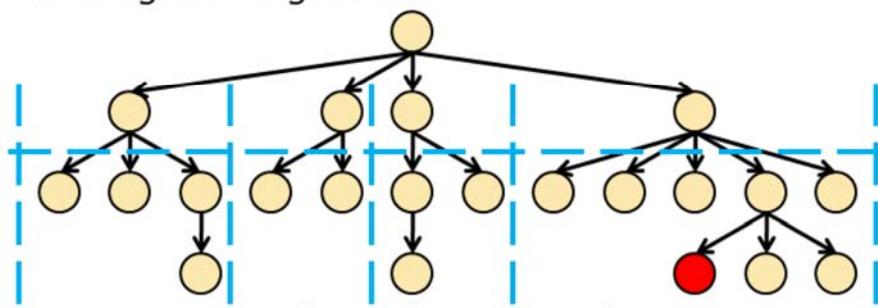
Hochleistungsrechnen - © Thomas Ludwig

302

Beispiel 3: Suchbaumverfahren (1/3)

(Das Beispiel illustriert eine wichtige Problemklasse)

- An jeder Position mehrere Fortsetzungen möglich
 - Probleme
 - Ebene der Lösung(en) unbekannt
 - Lastausgleich zwischen den Prozessoren
 - Feststellung des Programmendes



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

303

Siehei:

- http://en.wikipedia.org/wiki/Search_tree

Beispiel 3: Suchbaumverfahren (2/3)

Baumsuche mit **gemeinsamem** Speicher

- Thread berechnet Baum bis zur Ebene j und gibt Beschreibung der Fortsetzungsmöglichkeiten in eine Warteschlange
- Verfügbare Threads entnehmen Beschreibung aus der Warteschlange und berechnen den verbleibenden Baum
- Gute parallele Implementierung
 - Lastausgleich ist kein Problem
 - Feststellung des Berechnungsendes: setze Ende-Bit; andere Prozesse prüfen regelmäßig

Beispiel 3: Suchbaumverfahren (3/3)

Baumsuche mit **verteiltem** Speicher

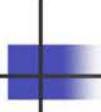
- Prozess i berechnet bis zum Level j und gibt Beschreibung der Fortsetzungsmöglichkeiten in eine lokale Warteschlange
- Untätige Prozesse kontaktieren Prozess i , bekommen Elemente der Warteschlange als Nachricht und berechnen den restlichen Baum

Gute parallele Implementierung

- Lastausgleich kein Problem, aber mehr Aufwand
- Feststellen des Berechnungsendes: sende Ende-Nachricht an alle anderen Prozesse; diese prüfen regelmäßig

Zusammenfassung zu den Beispielen

- Es gibt verschiedenste Varianten, Programme zu parallelisieren
- **Die konkrete Variante beeinflusst die maximal erzielbare Leistung des Verfahrens**
- ***Normalerweise kann die Effizienz der parallelen Programmvariante nicht am sequentiellen Programm bestimmt werden***
- **Datenaufteilung ist in den meisten Fällen einfacher zu programmieren**
- Suchbaumverfahren sind oft trivial zu parallelisieren



Parallele Programmierung

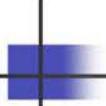
Zusammenfassung

- Parallelisierung von Programmen ist eine komplexe Tätigkeit
- Man nutzt Code-Aufteilung und Datenaufteilung
- Datenaufteilung meist einfach und effizient
- Werkzeuge: Programmierbibliotheken, Erfahrung
- Deutliche Unterschiede zwischen numerischen und nichtnumerischen Algorithmen
- Effizienz der Parallelisierung selten vorhersagbar

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

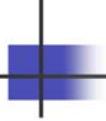
307



Parallele Programmierung

Die wichtigsten Fragen

- Was ist Parallelisierung eigentlich?
- Wie ist die allgemeine Vorgehensweise?
- Welche Paradigmen der Parallelisierung gibt es?
- Für welche Algorithmen sind diese Paradigmen jeweils gut geeignet?
- Mit welchen Werkzeugen wird parallelisiert?
- Welche Probleme gibt es bei der Parallelisierung durch den Compiler?
- Welche Klassen von Algorithmen können unterschieden werden?
- Wie parallelisiert man einfache numerische Verfahren?
- Wie parallelisiert man Anwendungen im Gebiet der Strömungsmechanik?
- Wie parallelisiert man Suchbaumverfahren?



Programmiermodell

Nachrichtenaustausch

1. Problemstellungen
2. Das Message Passing Interface (MPI)
3. Punkt-zu-Punkt-Kommunikation
4. Abgeleitete Datentypen
5. Kollektive Kommunikationen
6. Gruppen, Kontexte, Kommunikatoren
7. Profiling-Interface
8. Bewertung, Ausblick, Vergleiche

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

309

Literatur:

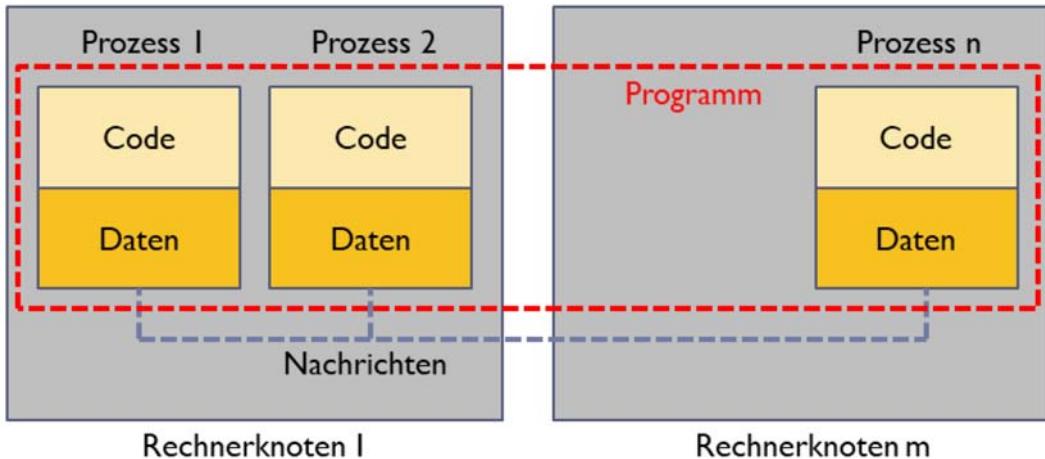
- Mark Snir et al.: MPI – The Complete Reference. Volume 1, The MPI Core. Second Edition. MIT-Press, 1998.
- William Gropp et al.: Using MPI – Portable Parallel Programming with the Message-Passing Interface. Second Edition. MIT-Press, 1999.

Leider gibt es keine richtig guten Bücher zum Thema. Online finden sich viele Unterlagen.

Siehe: https://en.wikipedia.org/wiki/Message_Passing_Interface

1. Problemstellungen

Die Codeteile des Programms in den Prozessen können identisch oder verschieden sein



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

310

Nochmal zum Begriff des Rechnerknotens oder Rechenknotens:

- Früher war ein Rechnerknoten ein einzelner Rechner, meist so eine Art PC, von denen man mehrere in einem Cluster zusammengebaut hat. D.h.: ein Knoten = ein Prozessor (=ein Prozessorkern). Darauf bringt man einen Prozess der Anwendung zum Ablauf.
- Heute ist ein Knoten z.B. am System „Mistral“ des DKRZ ein Einschub mit 2 Prozessoren zu je 12 (bzw. 18) Cores, also ein 24(bzw. 36)-Prozessor-System.

Die Zuweisung von Jobs zum Rechner erfolgt durch die Angabe der Anzahl von Knoten, die man haben möchte. Es werden den Benutzern immer ganzzahlige Vielfache dieser Knoten zugewiesen.

Auf einem Knoten, der dann gemeinsamen Speicher hat, kann ein Prozess nochmal in Threads unterteilt werden, die dann wiederum Code-Aufteilung und/oder Datenaufteilung folgen. Mischt man Nachrichtenaustausch mit Nutzung gemeinsamer Variable durch Threads, so wie z.B. bei OpenMP, so spricht man von hybrider Programmierung.

Problemstellungen

- Laden des Codes auf unterschiedliche Knoten
- Start der Prozesse auf den Knoten
- Wechselseitiges Bekanntmachen der Prozesse
- Informationsaustausch zwischen Prozessen
- Optimierung der Kommunikationseffizienz
- Kommunikationsrelationen der Prozesse zueinander
- Überwachungsmöglichkeit der Abläufe

Laden und Starten des Codes

- Prinzipieller Aufruf (allgemeiner Fall)
`spawn(<binary_name>,<node_list>, ...);`
 - Ist ähnlich wie bei einer Thread-Erzeugung
- Wenn nur ein Programmcode existiert, dann z.B.

```
if (myid()==0)
  then /* I'm the first */
    spawn(...); /* if others do not exist */
    send(init_data);
  else /* I was spawned */
    receive(init_data);
  fi
```

Nicht notwendigerweise nur ein Prozess pro Prozessor

Bei z.B. MPI werden die beteiligten n Prozesse von 0 bis n-1 durchnummerniert und das definiert quasi ihren Namen. Prozess 0 ist dann ein natürlicher Kandidat für Organisationsaufgaben. Bei anderen Bibliotheken können die Benennungen aber unterschiedlich funktionieren!

Informationsaustausch

- Senden von Nachrichten
 - `send(<to_proc_id>,<data>);`
 - `broadcast(<data>);`
- Empfangen von Nachrichten

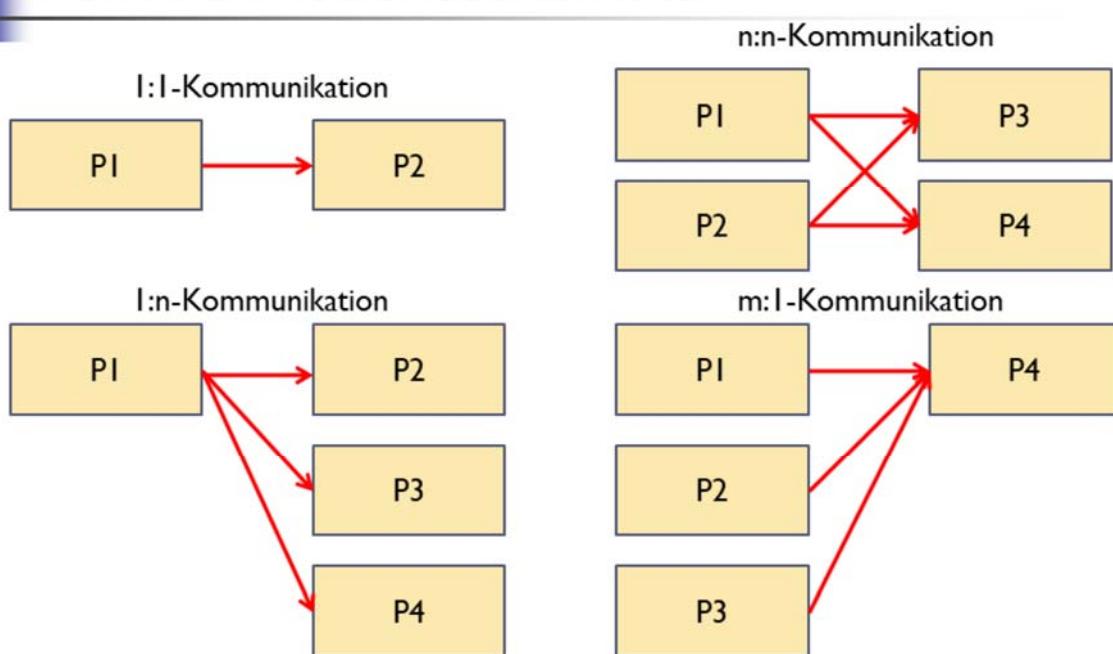
```
receive(<from_proc_id>,<data>);  
testreceive(<from_proc_id>);
```

Charakteristisch: eigenhändiges Einfügen der Kommunikationsanweisungen in den Code

Hoher Aufwand aber auch hohe Leistungsausbeute

Die Programmierung des Nachrichtenaustausch wird auch als die Maschinenprogrammierung des parallelen Rechnens bezeichnet, weil sie auf einer sehr niedrigen Abstraktionsebene ansetzt.

Kommunikationsschemata



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

314

1:1-Kommunikation: Ein Sender-Prozess sendet an genau einen Empfänger-Prozess. Der Sender kennt die Adresse vom Empfänger, der Empfänger die vom Sender.

1:n-Kommunikation: Ein Sender-Prozess sendet an viele Empfänger-Prozesse mit einem einzigen Aufruf. Geht die Nachricht an alle anderen, so spricht man von Broadcast, geht sie an eine echte Teilmenge, so spricht man von Multicast.

m:1-Kommunikation: Insgesamt m Sender-Prozesse senden an ein und denselben Empfängerprozess. Man unterscheidet syntaktisch Fälle, bei denen der Empfänger die Sender kennt und solche, wo das nicht der Fall ist. Ersterer wird verwandt, wenn z.B. Ergebnisse korrekt zusammengefügt werden müssen, letzterer, wenn man z.B. Teilergebnisse nur aufaddiert. Der zweite Fall kann meist mit besserer Effizienz implementiert werden.

n:n-Kommunikation wird verwendet, wenn eine Menge von Prozessen untereinander Daten austauschen, so dass jeder an Sende- und Empfangsoperationen beteiligt ist.

Alle komplexen Kommunikationsschemata können immer durch eine

Menge 1:1-Kommunikationen dargestellt werden. Manchmal wird das auch so implementiert, das ist dann schnell und ineffizient. Vielfach wird aber gerade hier sehr viel Aufwand in interne Optimierungen der Bibliotheken gelegt.

Optimierung der Kommunikationseffizienz

PI

Rechnen

Senden

Empfangen

Rechnen

Möglichst Senden und Empfangen nebenläufig abwickeln

Kombination aus Hardware (die das kann) und Software (Threads im Hintergrund) erforderlich

PI

Rechnen

Rechnen

Thread 1

Empfangen

Senden

Thread 2

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

315

Wenn eine Hintergrundtätigkeit ausgeführt wird, dann erledigt die Laufzeitbibliothek das normalerweise so, dass sie einen weiteren Thread startet, der die Aufgabe im Hintergrund abwickelt. Die Hauptanwendung läuft im Vordergrund weiter.

Existierende Ansätze Anfang der 90er

- **P4, Parmacs, Chameleon, NX, ...**

Historie der Bibliotheken zum Nachrichtenaustausch

- **Parallel Virtual Machine (PVM)**

Implementierung einer Bibliothek für nahezu alle Architekturen

Lange Zeit de facto-Standard bei Clustern

- **Message Passing Interface (MPI)**

Spezifikation einer Schnittstelle zum Nachrichtenaustausch

De facto-Standard auf allen Hochleistungsrechnern und auf Cluster-Architekturen

Siehe: https://en.wikipedia.org/wiki/Parallel_Virtual_Machine

Problemstellungen und Lösungen

- Laden des Codes auf unterschiedliche Knoten
 - Spezialprogramm der MPI-Implementierung kümmert sich darum
- Start der Prozesse auf den Knoten
 - Laufzeitbibliothek der MPI-Implementierung kümmert sich darum
- Wechselseitiges Bekanntmachen der Prozesse
 - Laufzeitbibliothek der MPI-Implementierung kümmert sich darum
- Informationsaustausch zwischen Prozessen
 - Aufgabe des Programmierers
- Optimierung der Kommunikationseffizienz
 - Aufgabe des Programmierers
- Kommunikationsrelationen der Prozesse zueinander
 - Möglichkeit des Programmierers, unterstützt von MPI
- Überwachungsmöglichkeit der Abläufe
 - Einsatz von geeigneten Werkzeugen durch den Programmierer

2. Das Message Passing Interface (MPI)

- Vorangetrieben vom MPI-Forum
(Firmen, Universitäten, ...)
- Beginn 1992
- MPI-Standard 1995 (nur Kommunikation)
- MPI-2-Standard 1997 (der nötige Rest)
- MPI-3.1-Standard 2015 (alles zusammen)
 - Dokument hat 868 Seiten
- Vorteile eines Standards: Portabilität, Einfachheit
Vorher etwa ein Dutzend konkurrierende Ansätze
 - Alle funktional fast identisch aber syntaktisch unterschiedlich
- Probleme: Standardkonformität der Implementierungen

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

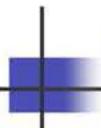
318

Siehe:

- <https://www mpi-forum.org/>
- <https://www mpi-forum.org/docs/mpi-3.1/mpi31-report/mpi31-report.htm>
- https://en.wikipedia.org/wiki/Message_Passing_Interface

Ziele von MPI

- Entwurf einer Programmierschnittstelle (API)
- Unterstützung effizienter Kommunikationsmethoden
- Unterstützung heterogener Umgebungen
- Sprachanbindungen für Fortran und C/C++
(jetzt auch für Java und Skript-Sprachen)
- Konstrukte nahe an bereits Existierendem
- Semantik der Schnittstelle soll sprachunabhängig sein
- Soll eine thread-sichere Implementierung gestatten
 - Wiedereintrittsfähige Routinen der Bibliothekimplementierung!



Was MPI ursprünglich enthält

- Punkt-zu-Punkt-Kommunikation
- Kollektive Operationen
- Prozessgruppen
- Kommunikationskontakte
- Prozesstopologien
- Abfragefunktionen zur Programmumgebung
- Profiling-Schnittstelle

Was MPI (zunächst) nicht enthält

- Explizite Operationen für gemeinsamen Speicher
- Zusätzliche Unterstützung durch das Betriebssystem für z.B. unterbrechungsgesteuerte Kommunikation
- Explizite Unterstützung zur Prozessverwaltung
- Parallele Ein-/Ausgabe

MPI zunächst nur Nachrichtenaustausch

MPI-2 geht die obigen Punkte an

MPI-3 fasst alles in einem Standard zusammen

MPI-Spezifikationsmethode

- Aufrufe sprachunabhängig definiert
- Argumente mit IN, OUT oder INOUT annotiert

Z.B. `MPI_WAIT(request, status)`
 INOUT `request`
 OUT `status`

C: `int MPI_Wait(MPI_Request *request,`
 `MPI_Status *status)`

F77: `MPI_WAIT(REQUEST, STATUS, IERROR)`
 `INTEGER REQUEST,`
 `STATUS(MPI_STATUS_SIZE),`
 `IERROR`

MPI-Definitionen

MPI sehr sorgsam mit Problemen der Sprache

Wichtige Begriffe werden eindeutig definiert

- *Nonblocking*: Der Aufruf kehrt zurück, bevor die Operation abgeschlossen ist und bevor die Ressourcen wiederverwendet werden dürfen
- *Locally blocking*: Bei Rückkehr dürfen die lokalen Ressourcen wiederverwendet werden
 - Hängt nur vom lokalen Prozess ab
- *Globally blocking*: Bei Rückkehr ist die Kommunikationsoperation abgeschlossen
 - Hängt von anderen Prozessen ab
- *Collective*: Alle Prozesse einer Gruppe müssen den Aufruf ausführen

3. Punkt-zu-Punkt-Kommunikation

Senden

```
MPI_SEND(buf, count, datatype, dest, tag, comm)
  IN buf      Adresse des Sendepuffers
  IN count    Anzahl der Elemente im Puffer
  IN datatype Datentyp des Elements
  IN dest     Rangangabe des Ziels
  IN tag      Nachrichtenkennung
  IN comm     Kommunikator (Gruppe, Kontext)
```

Datentypen: int, long int, float, char, ...

Nachrichten bestehen aus Inhalt und Umschlag

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

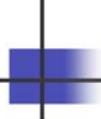
324

Der Datentyp kann ein elementarer sein, z.B. Byte, Integer, Float, oder auch ein zusammengesetzter komplexer wie z.B. ein Feld komplexer Zahlen mit zehn Zeichenketten hintendran.

Punkt-zu-Punkt-Kommunikation...

Empfangen

```
MPI_RECV(buf, count, datatype, source, tag,  
         comm, status)  
  
OUT buf      Adresse des Empfangspuffers  
IN count     Anz. der Elemente im Puffer  
IN datatype  Datentyp des Elements  
IN source    Rangangabe der Quelle  
IN tag       Nachrichtenkennung  
IN comm      Kommunikator (Gruppe, Kontext)  
OUT status   Ergebnis des Empfangens
```



Punkt-zu-Punkt-Kommunikation...

Empfangen...

- Gesteuert durch den Umschlag
`MPI_ANY_SOURCE`, `MPI_ANY_TAG` (Wildcard)
- Abfrage mittels
`MPI_GET_SOURCE()`, `MPI_GET_TAG()`

Aufgabe: In MPI gibt es zum Senden kein `MPI_ANY_DEST`. Überlegen Sie, wie die Semantik hiervon sein sollte, wenn es das gäbe. Wofür könnte man das gebrauchen? Wie könnte man es trotzdem implementieren?

Punkt-zu-Punkt-Kommunikation...

- Semantik der Kommunikation
 - Nachrichtenreihenfolge bleibt erhalten
- Blockierend / nichtblockierend
 - Legt fest, wann der Aufruf zurückkehrt
 - Nichtblockierende Kommunikation
 - Verbesserte Effizienz durch Überlappung von Berechnung und Kommunikation
 - Erhöht Wahrscheinlichkeit schwer auffindbarer Fehler
- Prinzip: der Aufruf wird mit einer Referenz versehen
Durch Abfragen bzgl. der Referenz kann der Status der Ausführung ermittelt werden

Punkt-zu-Punkt-Kommunikation...

Nichtblockierende Kommunikation

MPI_ISEND(. . . , request)	immediate send
MPI_IRecv(. . . , request)	immediate receive
MPI_TEST(request, flag, status)	nichtblock.
MPI_WAIT(request)	blockierend
MPI_CANCEL(request)	

MPI „Hello World“

```
#include "mpi.h"
#include <stdio.h>

int main (int argc, char *argv[])
{
    int rank, size;
    MPI_Init( &argc, &argv ); /* kein Code davor! */
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    printf("Hello World from process %d of %d\n",
           rank, size );
    MPI_Finalize(); /* kein Code danach! */
    return 0;
}
```

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

329

`MPI_Init()` und `MPI_Finalize()` rahmen den parallelen Teil eines Programmes eines. Für davor und dahinter gilt der Text aus den Man-Pages: „The MPI Standard does not say what a program can do before an MPI_Init or after an MPI_Finalize. In the MPICH [and Open MPI] implementation, it should do as little as possible. In particular, avoid anything that changes the external state of the program, such as opening files, reading standard input, or writing to standard output.“ – Der Hintergrund ist, dass üblicherweise schon alle Prozesse gestartet wurden. Es ist NICHT `MPI_Init`, das die Prozesse startet und auch nicht `MPI_Finalize`, das sie beendet. Üblicherweise werden die Prozesse von `mpexec` gestartet.

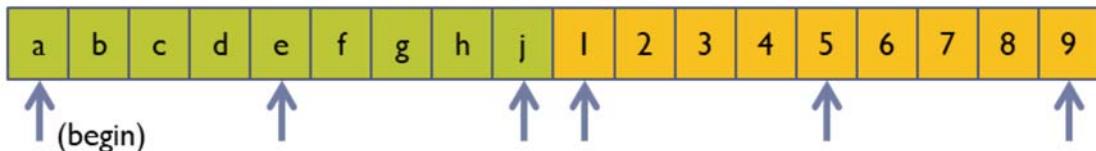
4. Abgeleitete Datentypen

- Verwendungszweck
 - Nachrichten mit gemischten Datentypen
 - Nachrichten mit nichtzusammenhängenden Bereichen
- Ein-/Auspicken der Nachrichten erfordert Rechenaufwand
- Effizienz hängt von der Hardware ab (z.B. Direct Memory Access, DMA)

Abgeleitete Datentypen...

Beispiel: Zwei Matrizen mit komplexen Zahlen

Aufgabe: Versende die beiden Diagonalen



```
MPI_TYPE_VECTOR(3/*blocks */, 1/*element/block*/,
                 4/*blockstride*/, MPI_COMPLEX, diag)
MPI_TYPE_CREATE_HVECTOR(2/*blocks*/, 1/*elm/blck*/,
                        9*sizeof(MPI_COMPLEX), diag, doublediag)
MPI_TYPE_COMMIT(doublediag)
MPI_SEND(begin, 1, doublediag, me, other, comm)
```

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

331

MPI_TYPE_VECTOR hängt n Blöcke zusammen (hier: 3), die die Blocklänge l (hier: 1) haben und im Abstand von s Elementen des Ausgangsdatentyps (stride, Versatz) (hier: 4) liegen. Ausgangsdatentyp ist hier MPI_COMPLEX, der neue Datentyp ist hier diag.

MPI_TYPE_CREATE_HVECTOR mißt den Versatz in Byte.

Das Konzept stützt sich sehr auf die Kenntnis der speicherinternen Organisation der einzelnen Datentypen.

5. Kollektive Kommunikationen

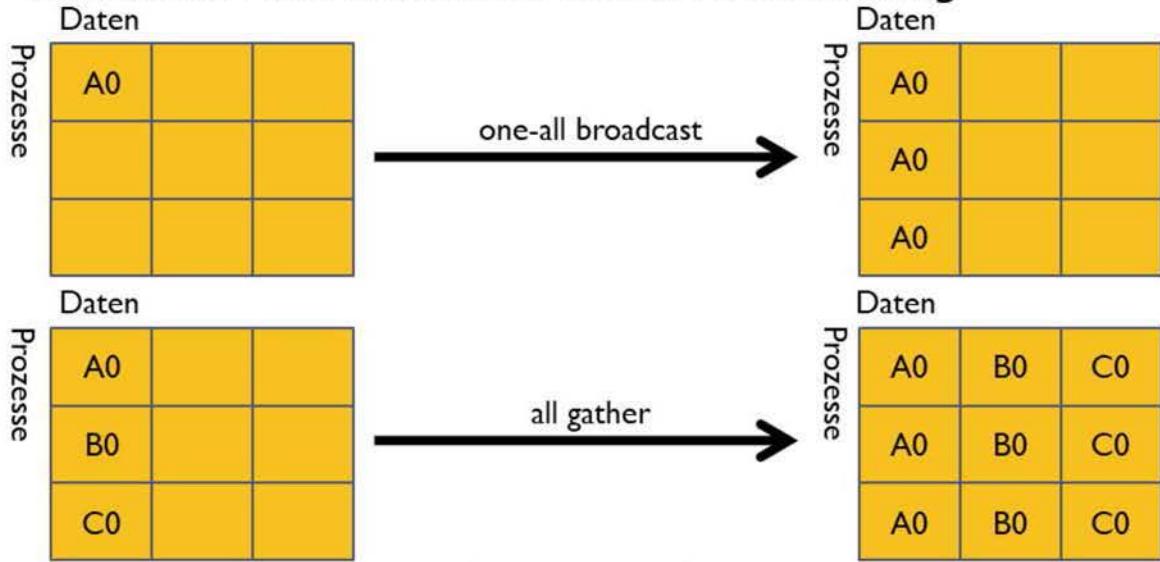
Kollektive Kommunikationen werden immer von allen Mitgliedern einer Gruppe durchgeführt

- Broadcast von einem an alle
- Barrierensynchronisation
- Daten einsammeln / verteilen
- Globale Berechnung von Funktionen

Möglicherweise durch spezielle Hardware unterstützt
Spielraum für Implementierungsoptimierungen

Kollektive Kommunikationen...

Kollektive Funktionen zur Datenverschiebung



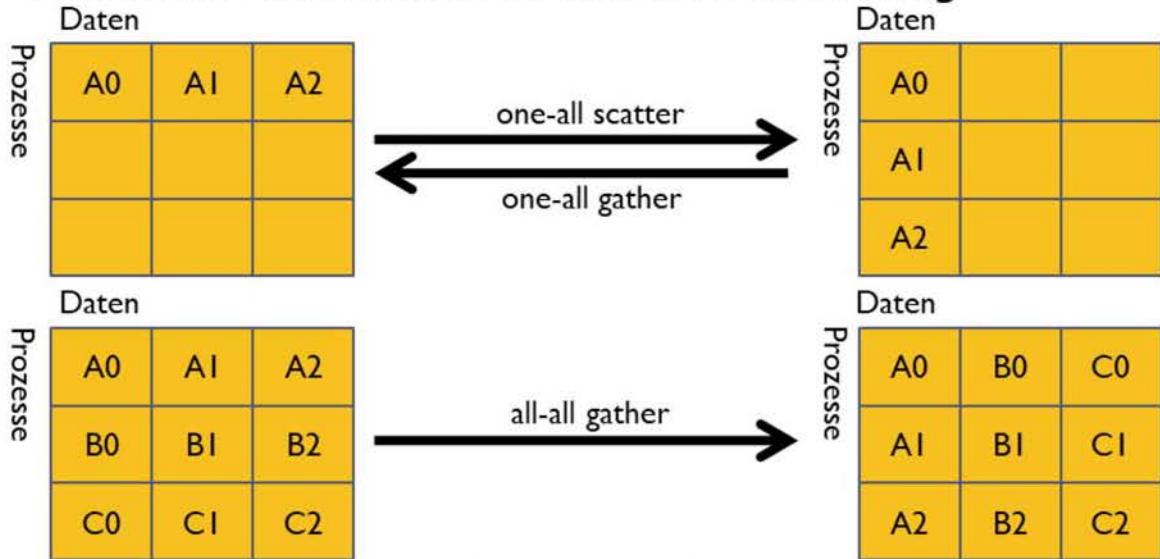
12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

333

Kollektive Kommunikationen...

Kollektive Funktionen zur Datenverschiebung



Kollektive Berechnungen

- Häufig müssen alle Prozesse dieselbe Funktion auf Daten anwenden, z.B. die Summenoperation
- Funktion **MPI_REDUCE**(..., op, ...)
 - Jeder Prozess trägt seinen Datenanteil bei
 - Am Ende hat jeder Prozess das Endergebnis
 - max, min, sum, product, AND, OR, XOR**
- Auswertereihenfolge beliebig
 - Evtl. nichtdeterministisches Ergebnis
- In Parallelrechnern teilweise durch Hardware unterstützt
- Eigene Funktionen möglich (kritisch)

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

335

Die Operation op wird immer als assoziativ angenommen, d.h. $a \text{ op } (b \text{ op } c) = (a \text{ op } b) \text{ op } c$. Außerdem sind alle vordefinierten Funktionen auch kommutativ, d.h. $a \text{ op } b = b \text{ op } a$. Z.B. ist aber eine Gleitkommaaddition im Rechner nicht absolut kommutativ und assoziativ wegen der begrenzten Rechengenauigkeit.

6. Gruppen, Kontexte, Kommunikatoren

- Neues Konzept, das es vorher nirgends gab
- Problem:
 - Drittanbieter entwickeln Bibliotheken mit Nachrichtenaustausch
 - Kennungen dieser Nachrichten und Rangangaben dürfen nicht mit dem Anwenderprogramm in Konflikt geraten
- Lösung
 - Gruppen fassen zusammengehörige Prozesse zusammen
 - Kontexte unterscheiden logische Teile des Programms
 - Kommunikator: fasst Gruppe und Kontext zusammen
 - Default-Kommunikator: **MPI_COMM_WORLD**

7. Profiling-Interface

- Möglichkeit zum Anschluss von Werkzeugen in MPI integriert
- Konzept
 - Unterstütze die Aktivierung von Überwachungen beim Aufruf von MPI-Funktionen
- Realisierung
 - Jede Funktion MPI_xyz muß auch über den Namen PMPI_xyz aufrufbar sein (*profiling*)

Profiling-Interface...

Beispiel: überwache Broadcast-Funktion

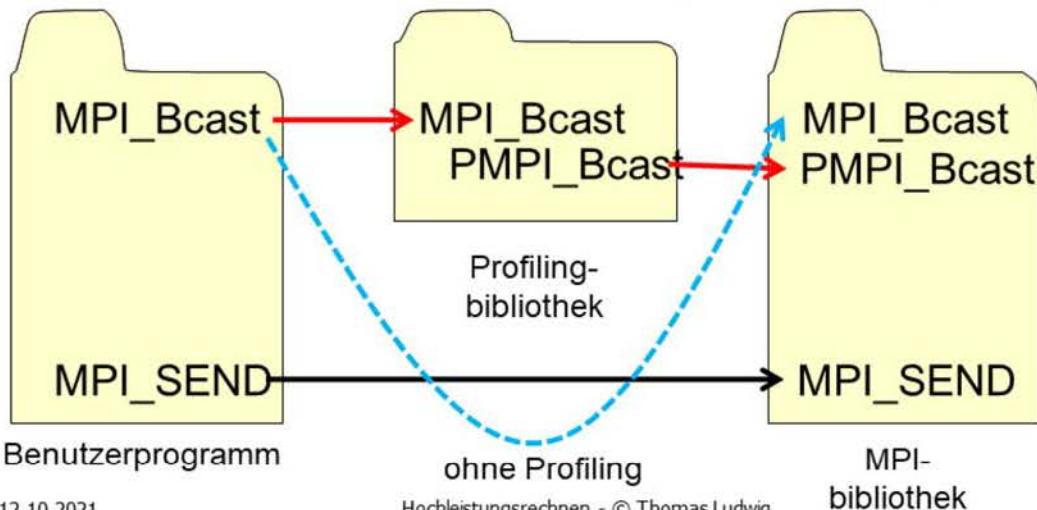
```
int MPI_Bcast(original_parameter)
{ int result;
  write_log_entry(...);
  start_timer();
  result=PMPI_Bcast(original_parameter);
  stop_timer(); write_log_entry(...);
  return result;
}
```

Dies definiert eine Profiling-Version der Funktion

Profiling-Interface...

Der Trick: Reihenfolge beim Linken

zuerst: Profiling-Bibliothek, dann: libmpi



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

339

8. Bewertung, Ausblick, Vergleich

Bewertung MPI(-1)

- Spezifikation ausschließlich für Nachrichtenaustausch
- Sehr viele Funktionen
 - Meist zur Bequemlichkeit und möglichen Optimierung
- Prozessverwaltung fehlt
- Kein dynamisches Prozesskonzept
 - Will man bei normalen Anwendungen sowieso nicht
 - Zu komplex in der Programmierung und Nutzung

Ausblick auf MPI-2

- MPI-2 war eine Erweiterung zu MPI, nicht eine neue Version
- Umfasst Klarstellungen zu MPI und Erweiterungen
- Wichtige Erweiterung: Prozessverwaltung
(Vorher machte jeder Hersteller was er wollte)
- Wichtige Erweiterung: Ein-/Ausgabe
(Idee: äquivalent zu Senden und Empfangen von Nachrichten)
- Relativer Nachteil: sehr viele neue Funktionen

Vergleich der Ansätze

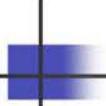
	Pthreads	OpenMP	MPI
Skalierbarkeit	Begrenzt	Begrenzt	Ja
Fortran / C und C++	Ja? / Ja	Ja / Ja	Ja / Ja
Hohe Abstraktion	Nein	Ja	Nein
Leistungsorientierung	Nein	Ja	Ja
Portierbarkeit	Ja	Ja	Ja
Herstellerunterstützung	Unix/SMP	Verbreitet	Verbreitet
Inkrement. Parallelisierung	Nein	Ja	Nein

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

342

Inkrementelle Parallelisierung bedeutet: man kann das Programm stückweise von einem sequentiellen in ein paralleles Programm umbauen. Z.B. je nach Zeit, die einem zur Verfügung steht. Dies geht hier in beiden Fällen nicht. Man kann nur entweder ein richtiges voll parallelisiertes Programm erstellen, oder man arbeitet mit dem sequentiellen weiter.



Programmiermodell Nachrichtenaustausch

Zusammenfassung

- Relevante Probleme beim Nachrichtenaustausch:
Kommunikationsschemata, Effizienz, Prozessverwaltung
- MPI ist eine Spezifikation eines API zum
Nachrichtenaustausch
- Punkt-zu-Punkt-Kommunikation mit vielen Varianten
möglich:
 - Für uns hauptsächlich: blockierend/nichtblockierend
- Abgeleitete Datentypen vereinfachen die Kommunikation
- Gruppen und Kontexte dienen zur wechselseitigen
Abgrenzung von Programmteilen
- MPI-2 erweitert MPI um wesentliche Aspekte

Programmiermodell Nachrichtenaustausch

Die wichtigsten Fragen

- Welche Kommunikationsschemata gibt es?
- Was sind die Ziele der MPI-Definition?
- Was enthält die MPI-Definition?
- Welche Variationen von Blockierungen gibt es bei den Funktionsaufrufen?
- Wie ist die Punkt-zu-Punkt-Kommunikation definiert?
- Wie funktioniert nichtblockierende Kommunikation?
- Was sind abgeleitete Datentypen?
- Was sind kollektive Kommunikationen?
- Wie funktioniert das Profiling-Interface?



Parallele Eingabe/Ausgabe

1. Einleitung
2. Konzepte und Definitionen
3. Einfache Beispiel-E/A
4. Methoden und Schnittstellen
5. Leistungsaspekte
6. Die Implementierung

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

345

Siehe:

- <https://www.mpi-forum.org/docs/>

Buch:

- Prabhat, Quincey Koziol: High Performance Parallel I/O, CRC-Press, 2014.
<https://www.crcpress.com/High-Performance-Parallel-IO/rabhat-Koziol/p/book/9781466582347>

1. Einleitung

Was ist MPI-2 I/O?

- Erweiterung des MPI-Standards um parallele Eingabe/Ausgabe
- Wird zuerst definiert im MPI-2-Standard
 - Jetzt Bestandteil von MPI-3.1
- Ein-/Ausgabe mit einer Semantik analog zum Nachrichtenaustausch von Prozessen
 - Z.B. collective, nonblocking werden auf E/A übertragen
 - E/A äquivalent zum Senden und Empfangen von Nachrichten



Wozu parallele E/A in MPI?

- Leistungsgewinn
 - Z.B. durch kollektive Aufrufe
 - Z.B. durch asynchrone E/A
 - Z.B. durch spezielle Dateisichten
- Einfacherer Zugriff durch Problemanpassung
 - Z.B. abgeleitete Datentypen bei irregulären Daten
 - Dadurch auch Portabilität in heterogenen Umgebungen

2. Konzepte und Definitionen

Wichtige Konzepte

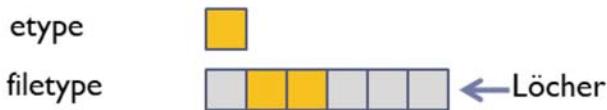
- File View (Dateisicht)
 - Prozessbezogene Sicht auf die Daten einer Datei
- File Pointer (Dateizeiger)
 - Individueller/gemeinsamer Dateizeiger
- Noncontiguous Access (Nichtzusammenhängender Zugriff)
- Collective Call (Kollektiver Aufruf)
- Hints (Hinweise)
 - Informationen für die Implementierungsschicht

Einige Definitionen

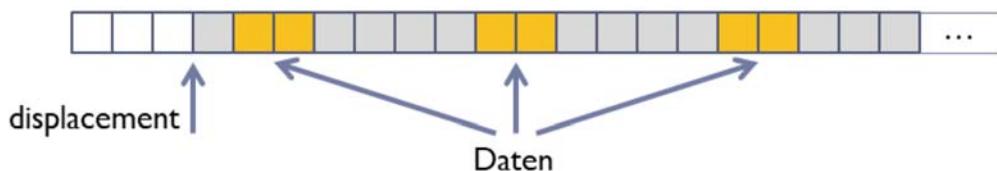
- **file (Datei)**
 - Eine geordnete Sammlung typisierter Daten
 - Zugriff erfolgt wahlfrei oder sequentiell
 - Kollektives Öffnen durch eine Gruppe von Prozessen
- **displacement (Versatz)**
 - Eine absolute Byte-Position relativ zum Dateianfang, an der eine Dateisicht beginnt
- **etype (elementary datatype)**
 - Die Einheit, mit der auf die Datei zugegriffen und in ihr positioniert wird

Einige Definitionen...

- filetype (Dateityp)
 - Schablone, nach der eine Datei aufgebaut wird
 - Besteht aus etype's und gleichgroßen Löchern



Aufbau einer Datei



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

350

Der Trick ist natürlich, dass jeder Prozess nur seine Daten sieht, dabei dann aber das, was alle in der Summe sehen, die gesamte Datei ergibt!

Einige Definitionen...

- view (Prozeßdateisicht)
 - Definiert durch displacement, etype und filetype

etype



process 0 filetype



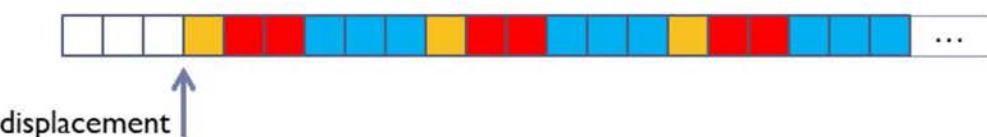
process 1 filetype



process 2 filetype



Aufbau einer Datei



Einige Definitionen...

- offset (Versatz)
 - Position in der Datei relativ im aktuellen view ausgedrückt durch die Anzahl der etype's
- file size (Dateigröße)
 - Anzahl Bytes ab dem Anfang der Datei
- file pointer (Dateizeiger)
 - Intern von MPI verwalteter Versatz
 - individual file pointer: jeder Prozeß hat einen
 - shared file pointer: alle Prozesse teilen sich einen
- file handle (Datei-Handle ☺)
 - Wie üblich

3. Einfache Beispiel-E/A

Mehrere Prozesse schreiben/lesen **eine** Datei

- Prozesse öffnen (kollektiv!) eine Datei, ...
MPI_FILE_OPEN
- ... jeder Prozess positioniert mit seinem eigenen Dateizeiger ...
MPI_FILE_SEEK
- ... und liest aus der Datei/schreibt in die Datei ...
MPI_FILE_READ
MPI_FILE_WRITE
- ... und schließt die Datei
MPI_FILE_CLOSE

Prototypen in C

```
int MPI_File_open (MPI_Comm comm,
                   char *filename, int amode, MPI_Info info,
                   MPI_File *fh)

int MPI_File_seek (MPI_File fh, MPI_Offset,
                   int whence)

int MPI_File_read (MPI_File fh, void *buf,
                   int count, MPI_Datatype datatype,
                   MPI_Status *status)

int MPI_File_write (MPI_File fh, void *buf,
                    int count, MPI_Datatype datatype,
                    MPI_Status *status)

int MPI_File_close (MPI_File *fh)
```

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

354

Bei MPI_File_seek bestimmt whence, wie die Position aus dem Offset berechnet wird: relativ zum Dateianfang, zur aktuellen Position oder zum Dateiende.

4. Methoden und Schnittstellen

- Datenzugriff und Positionierung
- Nichtzusammenhängende Zugriffe
- Kollektive Aufrufe
- Nichtblockierende E/A
- Gemeinsamer Dateizeiger
- Hinweise (hints)
- Dateiformate

Datenzugriff und Positionierung

- Drei Varianten der Positionierung
 - Explicit offsets
 - Individual file pointers
 - Shared file pointers
- Können innerhalb eines Programms gemischt verwendet werden
- Syntax
 - Explicit offsets: **MPI..._AT**
 - Shared: **MPI..._SHARED**, **MPI..._ORDERED**

Nichtzusammenhängende Zugriffe

- Bisher vorgestellte E/A ist auch durch üblich Unix-E/A zu bewerkstelligen: eine Datei, zusammenhängende Daten
- Aber: parallele Programme greifen oft mit mehreren Prozessen unabhängig und auf nichtzusammenhängende Positionen einer Datei zu
- MPI-2 I/O bietet Funktionen, die mit **einem Aufruf** nichtzusammenhängende Daten lesen können und es mehreren Prozessen gestatten, gleichzeitig auf die Datei zuzugreifen

Nichtzusammenhängende Zugriffe: Dateisicht

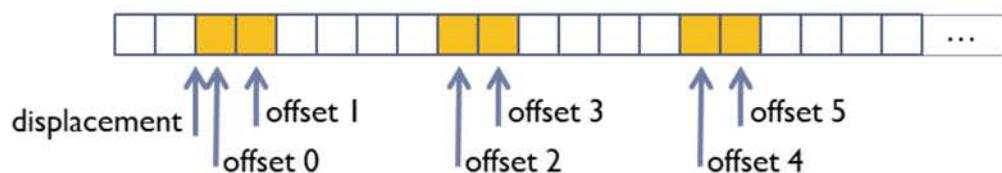
- Durch Dateisichten sieht jeder Prozess nur „seine“ Daten
- Dateisicht definiert durch
 - displacement, etype, filetype
etype und filetype sind Standard-Datentypen oder aus ihnen abgeleitete Datentypen!
- Dateisicht definiert durch
MPI_FILE_SET_VIEW
- Löcher müssen auch definiert werden
MPI_TYPE_CREATE_RESIZED

Nichtzusammenhängende Zugriffe: Beispiel

etype = MPI_INT

filetype = 2*MPI_INT resized zur Größe 6

Aufbau einer Datei



Nichtzusammenhängende Zugriffe: Beispiel...

```
/* 2 MPI_INT zusammenhängend als derived data type */
MPI_Type_contiguous(2,MPI_INT,&contig);

/* 4 Löcher anhängen; ergibt Größe 6 */
lower_boundary=0;
extent=6*sizeof(int);
MPI_Type_create_resized(contig,lower_boundary,extent,
&filetype);

/* und machen den neuen Typ bekannt */
MPI_Type_commit(&filetype);

/* und jetzt die Dateisicht */
MPI_File_set_view(filehandle,displacement,etype,filetype,
"native",MPI_INFO_NULL);
```

Kollektive Aufrufe

- Zur weiteren Optimierung können alle Prozesse gleichzeitig in der Datei zugreifen
- Definition einer Sicht wie zuvor, zusätzlich aber spezielle Funktionen
 - Erlaubt es der MPI-Implementierung, Zugriffe mehrerer Prozesse zu optimieren
- Selbst wenn jeder Prozess nur kleine, unzusammenhängende Stücke liest, kann die MPI-Implementierung (womöglich) einen großen, zusammenhängenden Zugriff daraus erstellen
- **MPI_FILE_READ_ALL, MPI_FILE_WRITE_ALL**

Nichtblockierende E/A

- Wird verwendet, um E/A mit Kommunikation und/oder Berechnung zu überlappen
- Alle nicht-kollektiven(!) Lese- und Schreibfunktionen haben nichtblockierende Entsprechungen
 - Zur Überprüfung der Beendigung kommt die Standard-MPI-Test-Funktion zum Einsatz
- Namenskonvention: **MPI_FILE_I...**
Also z.B. **MPI_FILE_IREAD**

Gemeinsamer Dateizeiger

- Bisher nur individuelle Zeiger und Versatz
- Ebenso möglich: gemeinsamer Zeiger
 - Von allen Prozessen gemeinsam genutzt
 - Jeder Zugriff irgendeines Prozesses verändert die Position
 - Nächster zugreifender Prozess sieht neue Position
- Funktionen
 - `MPI_FILE_SEEK_SHARED`**
 - `MPI_FILE_READ_SHARED`**
 - `MPI_FILE_WRITE_SHARED`**

Gemeinsamer Dateizeiger...

- Bei kollektiven Aufrufen kann gemäß dem Rang der Prozesse serialisiert werden
MPI_FILE_READ_ORDERED
- Typischer Anwendungsfall (**ordered** oder nicht)
 - Gemeinsame Protokolldateien
- Probleme
 - Erfordern Sperren – dadurch verringelter Gesamtdurchsatz
 - Zeiger in Metadatenserver zentral gespeichert – ineffizient

Hinweise (hints)

- Hinweise geben dem Nutzer die Möglichkeit, Informationen an die MPI-Implementierung durchzurichten
- Beispiele für Hinweise sind hier
 - Anzahl der Festplatten, über die eine Datei verteilt werden soll (striping)
 - Breite der Streifen (stripsize)
- Hinweise sind immer optional, der Benutzer muss sie nicht angeben
 - Gleichzeitig darf eine Implementierung Hinweise beliebig ignorieren

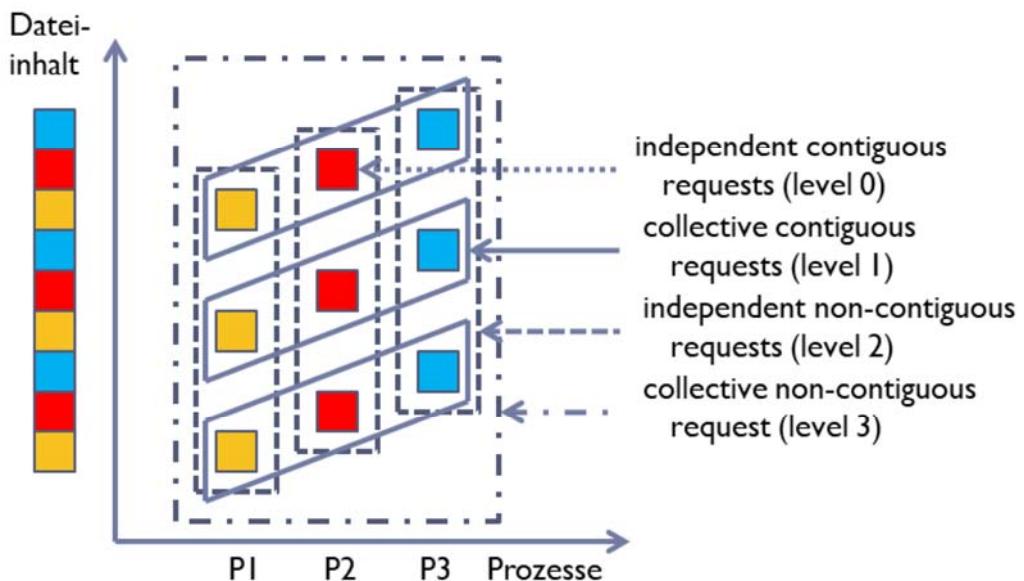
Dateiformate

- Dateien werden als Folge von Bytes gesehen
Die konkrete Abspeicherung ist Sache des Implementierungen
- MPI definiert drei Daten-Repräsentationen, die unterschiedliche Portabilität erlauben
 - „native“: keine Wandlung (=Speicherabbild)
schnell und nichtportabel
 - „internal“: portabel zwischen den Plattformen, die diese MPI-Implementierung unterstützt
 - „external32“: 32-bit big endian; portabel zu jeder MPI-Implementierung auf jeder Architektur; langsam

5. Leistungsaspekte

- Die Wahl der geeigneten E/A-Methode bestimmt die erzielbare E/A-Bandbreite
 - Zusammenhängend / nichtzusammenhängend
 - Kollektiv / nichtkollektiv
- Beispiel
 - Datei einer 3x3-Matrix komplexer Datentypen

Leistungsaspekte...



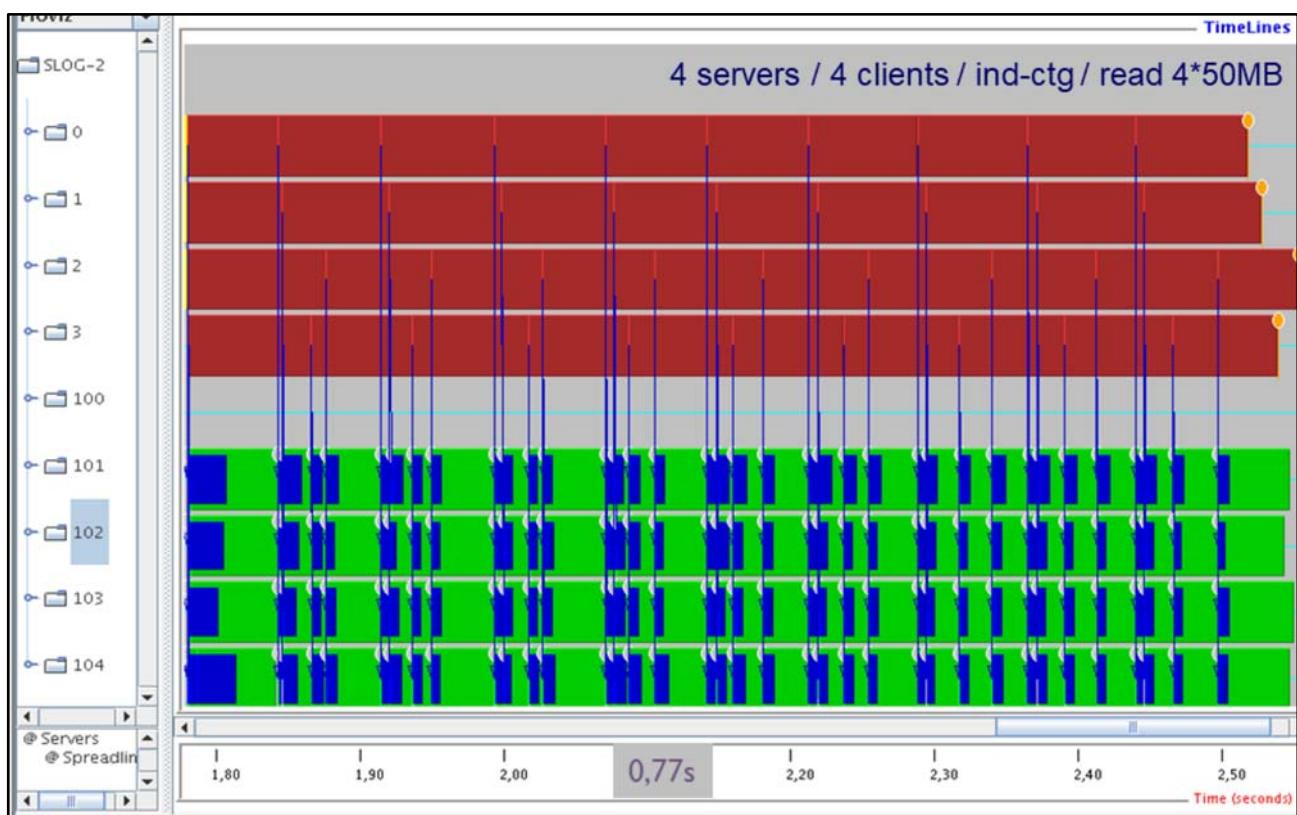
12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

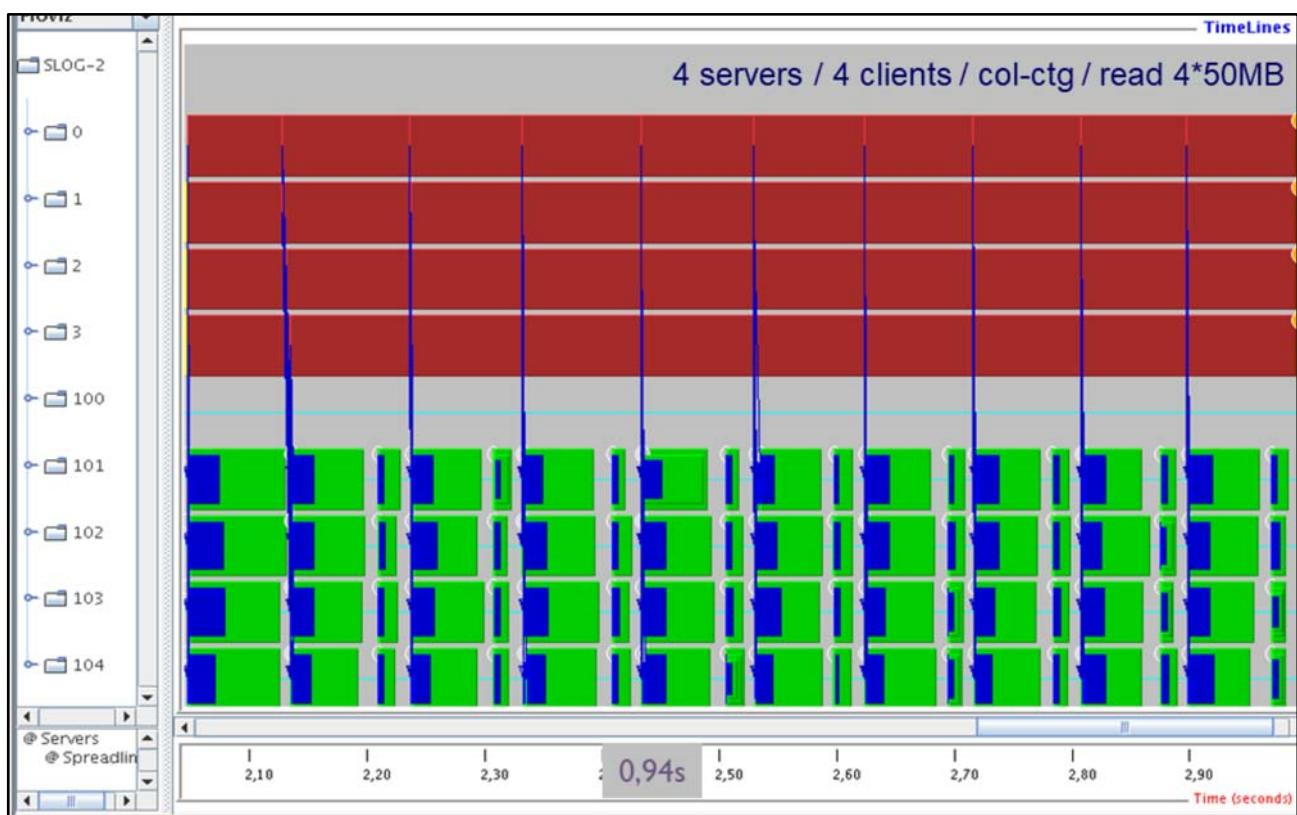
368

Je nach Level nimmt die Bibliothek verschiedene Optimierungen vor. Z.B. wird bei level 2 und lesenden Zugriffen auf nichtzusammenhängende Datenmenge ggf. eine größere Datenmenge in einem Zugriff gelesen und die nicht benötigten Teilabschnitte werden weggeworfen. Die anderen Optimierungen sind zu komplex, als dass wir sie hier diskutieren könnten.

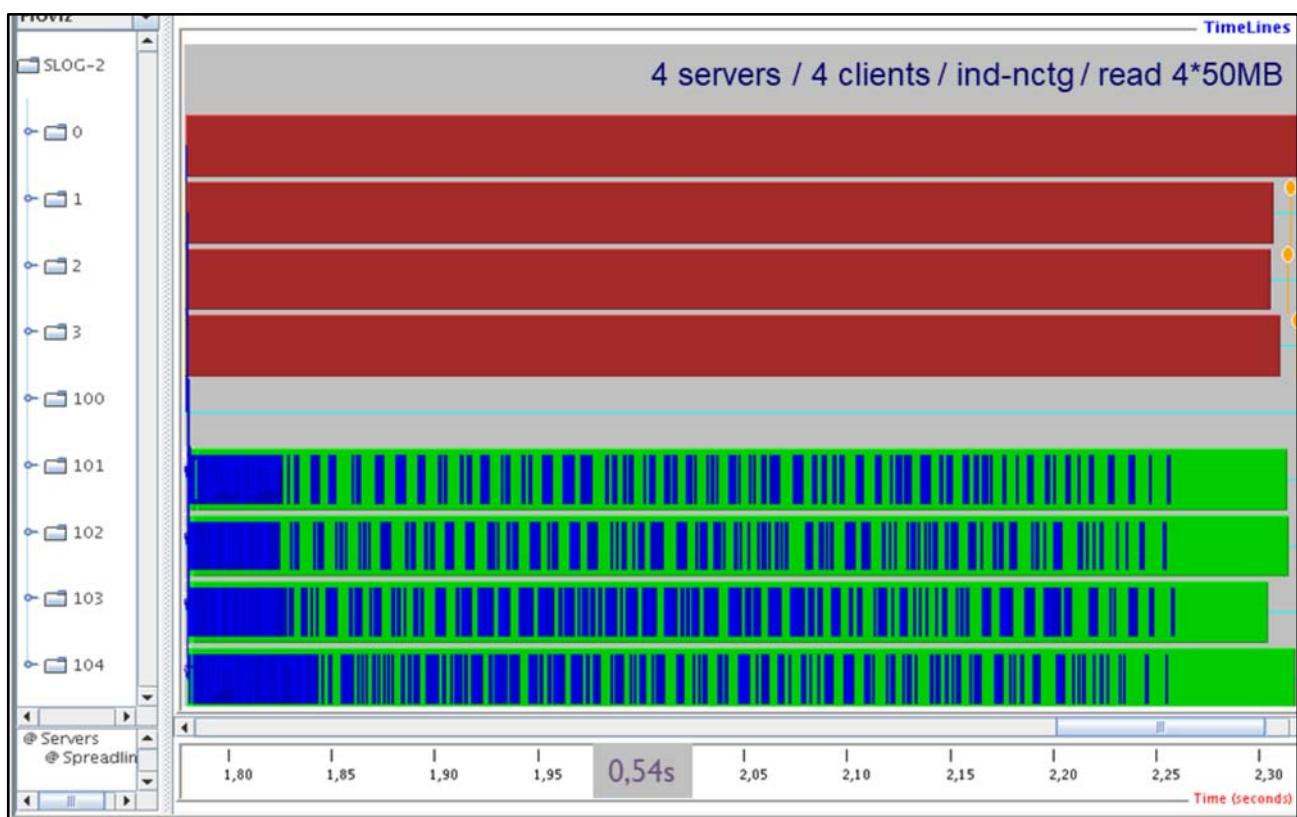
Wichtig ist: je mehr Information wir der Bibliothek geben können (in Form von Kenntnis über den gesamten Ablauf aller Zugriffe), desto mehr kann sie evtl. optimieren. Sie muss es allerdings nicht.



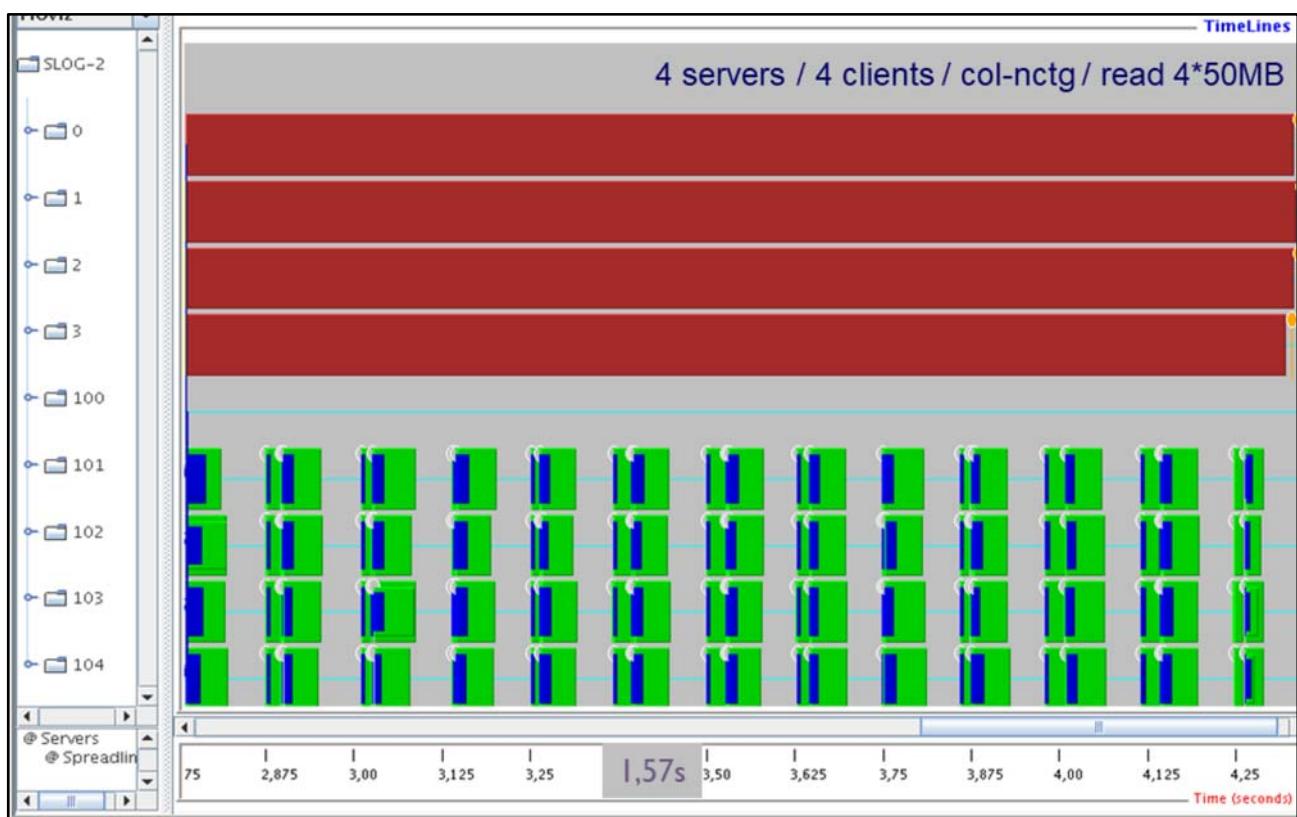
4s4c l0 r50M



4s4c l1 r50M

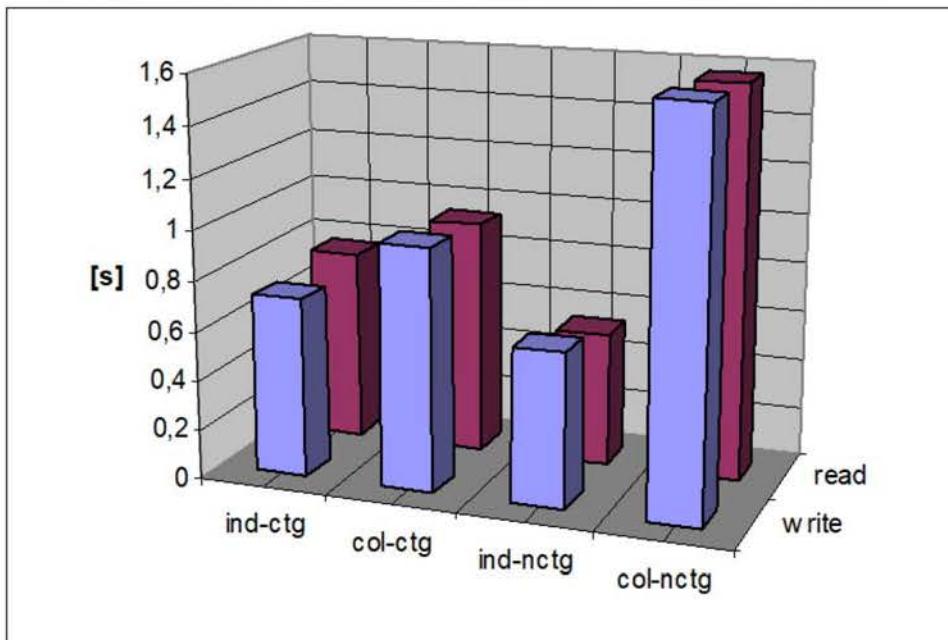


4s4c l2 r50M4s4c l2 r50M



4s4c l3 r50M

Zusammenfassung für 4*50 MB



12.10.2021

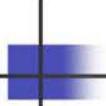
Hochleistungsrechnen - © Thomas Ludwig

373

6. Die Implementierung

- ROMIO ist eine/(die) Implementierung von MPI-2 I/O
 - Gehört zu MPICH, kann aber separat verwendet werden, insbesondere in anderen MPI-Implementierungen
- ROMIO unterstützt eine Reihe von Hardware-Architekturen und Dateisystemen
- ROMIO unterstützt alle(?) Merkmale von MPI I/O

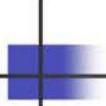
Siehe: <https://www.mcs.anl.gov/projects/romio/>



Parallele Eingabe/Ausgabe

Zusammenfassung

- Parallelle E/A analog zur Kommunikation definiert
- Verwendet auch abgeleitete Datentypen
- Dateien sind eine Sequenz elementarer Datentypen
- Jeder Prozess hat seine eigene Dateisicht
- Wir positionieren explizit, mit individuellen Dateizeigern oder einem gemeinsamen
- Nichtzusammenhängende Zugriffe erhöhen die Effizienz
- Kollektive Aufrufe erhöhen die Effizienz
- ROMIO ist die Standard-Implementierung



Parallele Eingabe/Ausgabe

Die wichtigsten Fragen

- Was ist MPI-2 I/O und wozu braucht man es?
- Welche Konzepte gibt es dabei?
- Wie ist eine Datei strukturiert?
- Wie geht die einfachste E/A?
- Wie funktionieren nichtzusammenhängende Zugriffe?
- Wie funktionieren kollektive Aufrufe?
- Wie funktioniert nichtblockierende E/A?
- Wozu verwendet man gemeinsame Dateizeiger?
- Wie optimiert man die Leistung?
- Welche Implementierung gibt es?

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

376



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

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

377

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

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

378

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

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!

In der Praxis sind die Threads dann Pthreads. Die sind der aktuelle Standard.

OpenMPs Hello World

rot: OpenMP-Konstrukte

```
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**
- Bibliotheksauftrag: **omp_get_thread_num()**
- Compiler-Direktive: **!\$omp parallel**

- Thread-Nummern: 0...**OMP_NUM_THREADS-1**

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

380

Hier sieht man auch sofort alle drei Konstrukte, die OpenMP beinhaltet:
Als wichtigstes die Compiler-Direktive, dann ein OpenMP-Bibliotheksauftrag
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 erst 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 u.a.
 - Aktuell Version 5.1 (Nov. 2020)
- 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

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

384

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

Variable können von folgenden Typen sein

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

Überblick (5)

Synchronisation

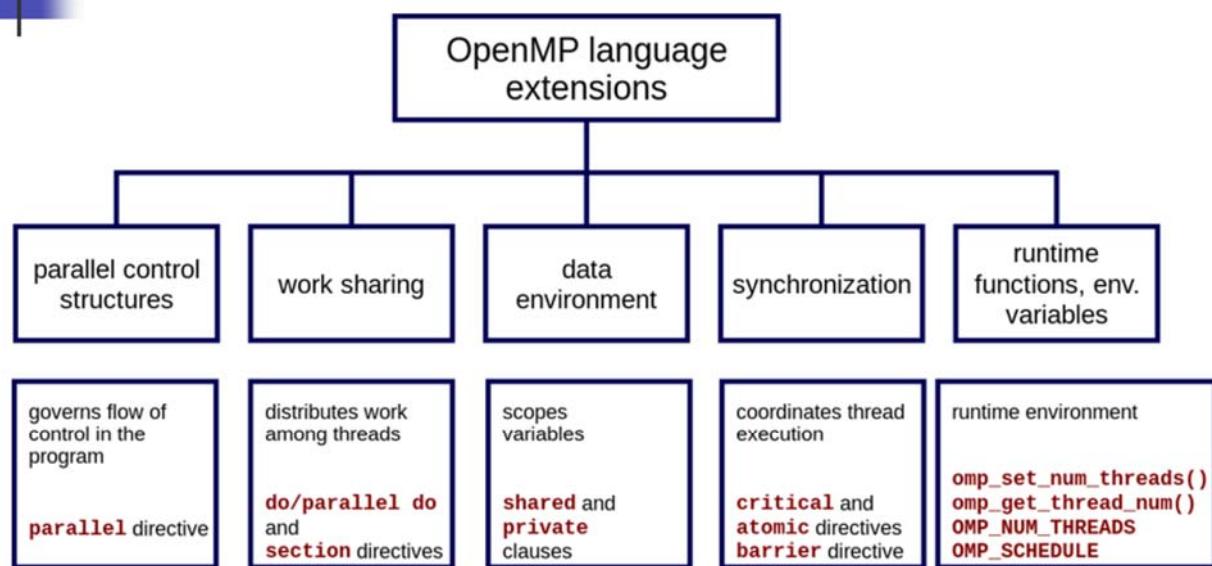
- Regelt den Ablauf der Threads

Zwei Hauptformen

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

Weitere Konstrukte zur Bequemlichkeit oder Leistungsoptimierung

Überblick (6)



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

388

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
```

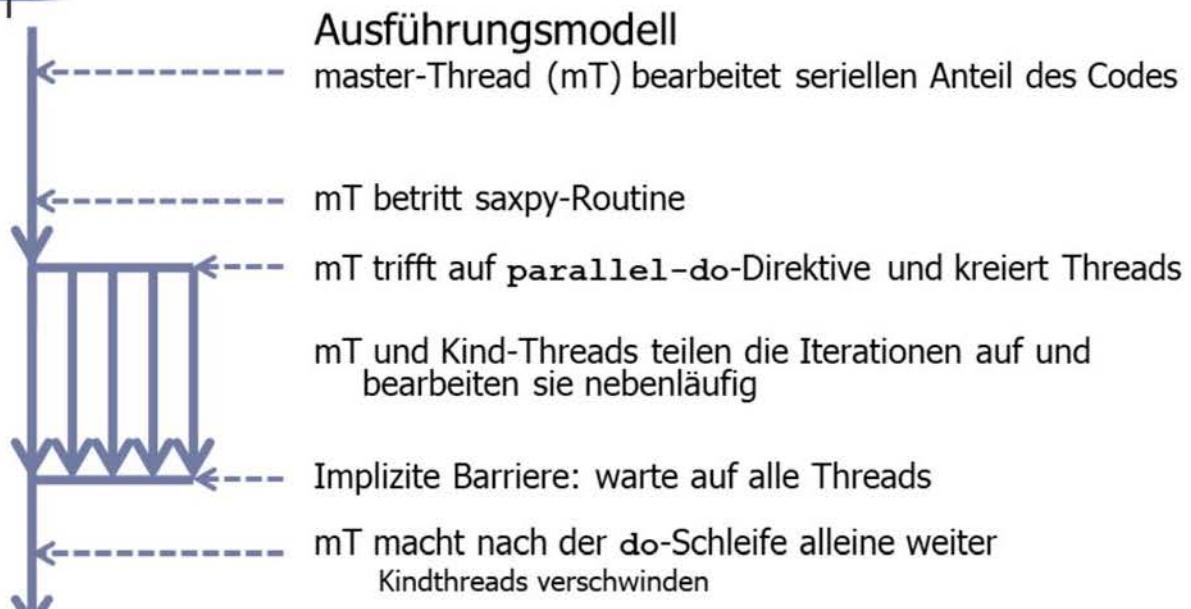
Keine Datenabhängigkeiten in der Schleife

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 Schleifenindexeben

Parallelisierung einer Schleife (3)



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 i 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)
- Variable `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
```

12.

396

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**

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
```

Vorsicht mit
bitweiser
Reproduzierbarkeit

12.1

399

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.

Bitweise Reproduzierbarkeit: in allen Berechnungsreihenfolgen kommt aufs Bit genau dasselbe heraus. Ist im Rechner nicht gegeben, da bei begrenzter Länge der Zahldarstellung die Operationen nicht assoziativ sind. D.h. es gilt NICHT: $a+(b+c) = (a+b)+c$. Im Falle DIESES Beispiels ist es kein Problem, da mit ganzen Zahlen gearbeitet wird. Aber eine Reduktionsvariable kann auch eine Gleitkommazahl sein und je nach Reihenfolge bei der Aufaddierung können die Ergebnisse leicht variieren.

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. Parallele 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
```

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
```

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

403

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]
```

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

407

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 der Threads (d.h. wie in einem sequentiellen Programm)

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

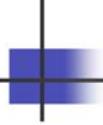
9. Bibliotheken und Compiler

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

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

413

Siehe:

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

10. Bewertung

Vorteile

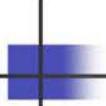
- Portabler Multithread-Code
- Einfach zu programmieren
- Inkrementelle Parallelisierung: beginne mit kleinen Änderungen
- Einheitlicher Code für sequentielle und parallele Programmversion
- Sequentieller Code muss nicht modifiziert werden, dadurch keine versehentlichen neuen Fehler
- Parallelismus auf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

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

416

Programmierung mit OpenMP

Die wichtigsten Fragen

- Was charakterisiert den Ansatz von OpenMP?
- Welche Konzeptklassen beinhaltet OpenMP?
- Welche Konstrukte zur Parallelarbeit gibt es?
- Wie werden Variable 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?



Programmierung mit Threads

1. Prozesse und Threads
2. Thread-Programmierung
3. Die Pthreads-Schnittstelle
4. Thread-sichere Programmierung
5. Threads und Linux
6. Bewertung

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

418

1. Prozesse und Threads

Traditionelle Prozesse

- Ein Prozess ist der Ablauf eines Programms
- Aus Betriebssystemsicht
 - Prozess ist Einheit der Ressourcenbelegungen
(Speicher, Dateien, E/A-Ports)
 - Prozess ist Einheit der Prozessorzuteilung
- Grundidee von Threads (eine Art Unterprozess)
(auch bezeichnet als „leichtgewichtiger Prozess“)
 - Aufspaltung dieser Eigenschaften:
Prozess ist Einheit der Ressourcenbelegung
Thread ist Einheit der Prozessorzuteilung

Prozesse und Threads (2)

Eigenschaften von Threads

- Threads eines Prozesses haben gemeinsame Ressourcen (Speicher, Dateien, ...)
 - Einfache, effiziente Kooperation möglich
 - Aber: kein gegenseitiger Schutz
- Parameter zur Erzeugung eines Threads
 - Zeiger auf Programmcode (typisch auf Funktion)
 - Weitere Parameter: Kellergröße, Scheduling usw.
- Einheit der Prozessor(kern)zuteilung
- Geringer Verwaltungsaufwand
- Schneller Wechsel (innerhalb desselben Prozesses)

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

420

Zur Systemstabilität: Stürzt ein Prozess ab, so sollte der Rest des Systems unberührt weiterlaufen. Stürzt ein Thread ab, z.B. mit einer Division durch Null, dann reißt er alle Threads des Prozesses und den Prozess mit sich. Der Rest des Systems sollte unberührt weiterlaufen. Verklemmt sich ein Thread, so laufen die anderen Threads im Prozess weiter. Meist ist dann das Ergebnis inkorrekt.

Prozesse und Threads (3)

Bedeutung von Threads

- Verringerung der Antwortzeit von Servern
 - Unterbrechbarkeit langer Anfragen
- Durchsatzsteigerung
 - Überlappung blockierender Systemaufrufe
- Behandlung asynchroner Ereignisse
- Realzeitanwendungen
 - Hochpriorisierte Threads für zeitkritische Aufgaben
- Basis für Parallelverarbeitung auf Mehrkernprozessoren
- Strukturierung von Programmen



Threads und Hochleistungsrechnen?

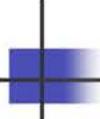
Fakten

- Alle modernen Betriebssysteme können mit Threads in Prozessen umgehen
- Sie bilden diese auf die Prozessorkerne des Systems ab
- Moderne Bibliotheken arbeiten mit Threads oder müssen zumindest thread-sicheren Code implementieren
 - Details später
- Der Compiler für OpenMP-Programme erzeugt Threads
 - Leider nicht auf eine einfach nachvollziehbaren Weise
 - Leider auch nicht auf eine standardisierte Weise

2. Thread-Programmierung

Nochmal im Überblick

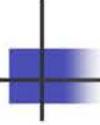
- Prozesse
 - Jeder Prozess hat eigenen Adressraum
 - Kommunikation zwischen Prozessen nur mit Betriebssystem-Unterstützung
 - signals, pipes, sockets, streams
 - shared memory segments
- Threads
 - Nutzen gemeinsam den Adressraum ihres Prozesses
 - Nutzen auch alle anderen Ressourcen ihres Prozesses gemeinsam: offene Dateien, sockets zur Kommunikation mit anderen Prozessen usw.
 - Müssen sorgfältig koordiniert werden, um falsche Programmergebnisse zu verhindern



Thread-Programmierung (2)

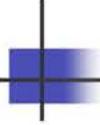
Drei Varianten der Programmierung von Mehrkernsystemen

- Unabhängige Prozesse
 - Z.B. bei WWW- und Datei-Servern (`fork()`)
 - Keine spezielle Programmierung nötig
 - Wechselseitiger Schutz der Server-Prozesse
 - Hohe Antwortzeiten
- Kommunizierende Prozesse
 - Wenn Kooperation **und** Schutz erforderlich sind
Z.B. X Window Client und Server
 - z.b. die Implementierung von Copy&Paste-Mechanismen
 - Kommunikation aufwendig und unkomfortabel



Thread-Programmierung (3)

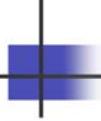
- Threads
 - Bei allen Arten von Servern
 - Für parallele Programme oder Programmabschnitte
 - Hohe Effizienz
 - Einfache Kooperation
 - Kein wechselseitiger Schutz durch Betriebssystem
 - Korrekte Synchronisation schwieriger



Thread-Programmierung (4)

Im Folgenden

- Einführung in POSIX-Threads (Pthreads)
 - Standard IEEE P1003.1c
 - In vielen Systemen realisiert
 - Konzepte in anderen Thread-Realisierungen meist ähnlich
 - POSIX-konforme Realisierungen unter Linux



3. Die Pthreads-Schnittstelle

Eingeteilt in drei (informelle) Klassen

- Thread-Verwaltung
- Mutex-Verwaltung
- Bedingungsvariablen-Verwaltung

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

427

Siehe:

- <https://en.wikipedia.org/wiki/Pthreads>
- <https://computing.llnl.gov/tutorials/pthreads/> – sehr gutes Tutorial zum Thema

Tippe unter Linux: `man pthreads`

Die Pthreads-Schnittstelle (2)

Thread-Erzeugung

- Programmiermodell
 - Bei Start eines Prozesses existiert genau ein Thread
 - Dieser erzeugt ggf. weitere Threads und wartet auf deren Ende
 - Terminierung des Prozesses bei Terminierung des Master-Thread
- Funktion zur Thread-Erzeugung

```
int pthread_create(pthread_t *new_thrd_ID,  
                  const pthread_attr_t *attr,  
                  void *(*start_func)(void *) ,  
                  void *arg)
```

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

428

Das Kreieren eines Thread liefert eine ID zurück. Man übergibt als Parameter die Attribute für diesen Thread, einen Zeiger auf eine Funktion, die den Thread-Code darstellt und den einen (erlaubten) Parameter für diesen Thread.

Die ID ist „opak“ (opaque identifier), d.h. die Details bleiben vor uns verborgen und wir benötigen sie auch nicht. Das ist ähnlich wie bei einem Dateideskriptor für eine geöffnete Datei.

Die Pthreads-Schnittstelle (3)

```
void* print_hello (void* threadid)
{ printf("Hello world from thread # %d!\n",
        (int)threadid);
  pthread_exit(NULL);
}

int main (int argc, char *argv[])
{ pthread_t threads[NUM_THREADS];
  for (int t = 0; t < NUM_THREADS; t++) {
    printf("In main: creating thread %ld\n", t);
    pthread_create(&threads[t], NULL,
                  print_hello, (void*)t);
  }
  for (int t = 0; t < NUM_THREADS; t++) {
    pthread_join(threads[t], NULL);
  }
}
```

Die Pthreads-Schnittstelle (4)

Mögliche Ausgabe des Programms bei NUM_THREADS=5

```
In main: creating thread 0
In main: creating thread 1
Hello world from thread #0!
In main: creating thread 2
Hello world from thread #2!
Hello world from thread #1!
In main: creating thread 3
Hello world from thread #3!
In main: creating thread 4
Hello world from thread #4!
```

Die Rückmeldungen der gestarteten Threads erfolgen verzögert in Abhängigkeit des Scheduling. Insofern kann es bei den Rückmeldungen zu Änderungen in der Reihenfolge kommen.

Hier wird ein bisschen geschummelt: die vom Thread ausgegebene Nummer ist nicht seine Thread-ID sondern die im Programm vergebene Integer-Zahl bei seiner Generierung. Wir arbeiten aber meist mit letzterer.



Die Pthreads-Schnittstelle (5)

Thread-Attribute (1)

- Keller und Kellergröße
- Scheduling-Schema und Priorität
- Affinität
- **detachstate**: Verhalten bei Beendigung des Threads

Bei `detached` thread erfolgt die Freigabe der Ressourcen sofort bei Beendigung, ansonsten erst nach Abschlussynchronisation

Die Pthreads-Schnittstelle (6)

Thread-Attribute (2)

- **pthread_attr_init**
Attributstruktur initialisieren
- **pthread_attr_destroy**
Attributstruktur löschen
- **pthread_attr_getXYZ / pthread_attr_setXYZ**
Lesen und setzen von Attributwerten
 $XYZ = \text{stacksize}, \text{schedpolicy}, \dots$



Die Pthreads-Schnittstelle (7)

Thread-Verwaltung (1)

- **pthread_self**
Liefert eigene Thread-ID
- **pthread_exit**
Eigene Terminierung
- **pthread_cancel**
Terminiert anderen Thread
 - Kann maskiert werden
 - Terminierung asynchron oder an bestimmten Punkten

Die Pthreads-Schnittstelle (8)

Thread-Verwaltung (2)

- **pthread_join**

Wartet auf Terminierung des spezifizierten Threads
(Abschlußsynchronisation)

- **pthread_sigmask**

Setzt Signalmaske (jeder Thread hat eigene Maske)

- **pthread_kill**

Sendet Signal an anderen Thread innerhalb des Prozesses

- Von außen nur Signal an *irgendeinen* Thread möglich

Die Pthreads-Schnittstelle (9)

Thread-Synchronisation durch eine Barriere

- **pthread_barrier_init(..., count)**
Erzeuge und initialiere eine Barriere und spezifizierte die Anzahl der Threads, die über sie synchronisiert werden sollen
- **pthread_barrier_wait**
Jeder zu synchronisierende Thread ruft die Funktion auf und wird schlafend gelegt bis die vorher spezifizierte Anzahl von Threads in die Barriere eingelaufen sind. Danach werden alle fortgesetzt. Einer(!) der fortgesetzten Threads erhält den reservierten Rückgabewert PTHREAD_BARRIER_SERIAL_THREAD. Man nutzt diesen, um Aktionen zu steuern, die nur einer durchführen soll, wie z.B. das Melden des Verlassens der Barriere. Alle anderen erhalten den Rückgabewert 0.
- **pthread_barrier_destroy**
Lösche die Barriere mit ihren Ressourcen

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

435

Barrieren werden verwendet, um alle Threads wieder zu synchronisieren. Bei OpenMP haben wir z.B. implizite Barrieren nach der parallelen Abarbeitung von Schleifen, um sicherzustellen, dass alle Werte aktualisiert worden sind und sich die Daten damit in einem konsistenten Zustand befinden.

Die Pthreads-Schnittstelle (10)

Mutex-Operation

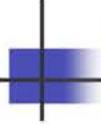
- wechselseitiger Ausschluss, mutual exclusion
- Z.B. Schutz von (globalen) Variablen bei mehreren Schreibern
- **pthread_mutex_init**
Initialisiert Sperrvariable (mutex)
- **pthread_mutex_destroy**
- **pthread_mutex_lock**
Blockiert Thread, bis Sperre frei ist (a) und belegt dann die Sperre (b)
- **pthread_mutex_trylock**
Belegt Sperre, falls möglich / kein Blockieren
- **pthread_mutex_unlock**

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

436

Mutex wird verwendet, wenn z.B. mehrere Schreiber auf eine Variable zugreifen. Hierdurch wird die Variable geschützt, so dass sie immer nur ein Schreiber zu einem Zeitpunkt manipulieren kann.



Die Pthreads-Schnittstelle (11)

Anmerkungen zu Mutex-Operationen

- Operationenpaar (a), (b) muss unteilbar sein
- Bei Terminierung eines Threads werden Sperren nicht automatisch freigegeben
- Zur Blockierung wird kein Busy-Waiting verwendet, d.h. der blockierte Thread benötigt keine CPU-Zeit

Die Pthreads-Schnittstelle (12)

Bedingungsvariable

- Zur Signalisierung von Bedingungen zwischen Threads
- Erlauben Realisierung von Monitoren
 - (strukturierte Form des wechselseitigen Ausschlusses nach Hoare)
 - Extern sichtbare Funktionen eines Moduls stehen unter wechselseitigem Ausschluss
 - Aufrufer braucht sich damit nicht um Synchronisation zu kümmern

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

438

Siehe: [https://en.wikipedia.org/wiki/Monitor_\(synchronization\)](https://en.wikipedia.org/wiki/Monitor_(synchronization))

Ein Monitor verwaltet z.B. Daten und eine Zugriffsfunktion (so etwa wie eine Klasse bei OO-Programmierung). Intern wird vom Monitor durch einen Thread sichergestellt, dass nur jeweils ein Aufrufer zu einem Zeitpunkt die internen Daten manipulieren kann.

Die Pthreads-Schnittstelle (13)

Verwendungszweck

- Ein Thread soll eine Aktivität ausführen, wenn eine Bedingung erfüllt ist, z.B. Variable > Grenzwert
- Mit Mutex: Thread prüft **regelmäßig** Variable
 - Deswegen sehr kostspielig, ähnlich spinlock im Betriebssystem
 - (Einzelner Mutex-Aufruf aber unkritisch)
- Mit Bedingungsvariable: Thread wird **blockiert, bis** ihm ein **anderer** Thread **signalisiert**, dass die Bedingung erfüllt ist
 - Etwas ähnlich zu Semaphor-Objekt

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

439

Die zu lösende Situation besteht z.B. darin, dass ein Thread eine Aktivität ausführen soll, wenn eine bestimmte Bedingung erfüllt ist, z.B. der Wert einer Variablen überschreitet einen Grenzwert.

Die Lösung mit Mutexen sieht so aus, dass dieser Thread immer wieder Zugriff auf die Variable erwirbt und dann prüft und gegebenenfalls seine Aktionen ausführt. Das ist sehr kostspielig.

Mit Bedingungsvariablen wartet er blockierend darauf, dass ihm **ein anderer** Thread signalisiert, dass die Bedingung erfüllt ist. Die Signalisierung wird nur dort potentiell ausgelöst, wo der Wert der Variablen durch einen anderen Thread erhöht wird.

Die Pthreads-Schnittstelle (14)

Operationen auf „Bedingungsvariablen“

das ist **nicht** die Variable der Bedingung, die uns interessiert,
sondern das Objekt der Pthreads-Bibliothek zur Steuerung !

- **pthread_cond_init**
Initialisiert Bedingungsvariable
- **pthread_cond_destroy**
- **pthread_cond_wait (*cond_var, mutex*)**
Gibt eine Sperre frei (a), blockiert dann bis Bedingung
signalisiert wird (b) und belegt Sperre wieder
- **pthread_cond_signal (*cond_var*)**
Signalisiert Bedingung; setzt (mindestens) einen wartenden
Thread fort

Die Pthreads-Schnittstelle (15)

Operationen auf Bedingungsvariablen...

- **pthread_cond_timewait**
Wie `wait` aber mit begrenzter Wartezeit
- **pthread_cond_broadcast**
Wie `signal`, aber mit Fortsetzung aller wartenden Threads

Die Pthreads-Schnittstelle (16)

Amerkungen zu Bedingungsvariablen

- Operationspaar (a), (b) in `pthread_cond_wait` (Freigabe des Mutex, Warten auf Bedingung) muss unteilbar implementiert sein
- Bedingungsvariable „merken“ sich die Signalisierung nicht
 - Wenn bei Signalisierung kein wartender Thread existiert, bleibt sie ohne Wirkung
 - Auch dann, wenn später ein Thread auf die Bedingung wartet
- Bedingungsvariable sind opake Datentypen
- Signalisierung sollte bei belegtem Mutex erfolgen

Die Pthreads-Schnittstelle (17)

```
void* producer (void)
{
    mutex_lock(&m);
    while_(true)
    {
        while (produced > n)
            cond_wait(&c, &m);

        produce();
        produced++;
        cond_signal(&c);
    }
    mutex_unlock(&m);
}
```

pthread_cond_wait (cond_var,mutex)

Gibt eine Sperre frei, blockiert dann bis Bedingung
signalisiert wird und belegt Sperre wieder

12.10.2021

```
void* consumer (void)
{
    mutex_lock(&m);
    while_(true)
    {
        while (produced == 0)
            cond_wait(&c, &m);

        consume();
        produced--;
        cond_signal(&c);
    }
    mutex_unlock(&m);
}
```

pthread_cond_signal (cond_var)

Signalisiert Bedingung; setzt(mindestens)
einen wartenden Thread fort

Hochleistungsrechnen - © Thomas Ludwig

443

4. Thread-sichere Programmierung

Definition

- Eine Funktion kann gleichzeitig von mehreren Threads ohne gegenseitige Behinderung ausgeführt werden

Implementierungsansätze ohne gemeinsame Variable

- Wiedereintrittsfähigkeit (Re-entrancy)
- Thread-lokaler Speicher
- Unveränderbare Objekte

Implementierungsansätze mit gemeinsamen Variablen

- Gegenseitiger Ausschluss
- Atomare Operationen

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

444

Siehe:

- [https://en.wikipedia.org/wiki/Threadsafe](https://en.wikipedia.org/wiki	Threadsafe)
- https://en.wikipedia.org/wiki/Reentrancy_%28computing%29
- <https://www.daniweb.com/programming/software-development/threads/489679/difference-between-thread-safe-and-reentrant>

„Thread safety is a key challenge in multi-threaded programming. It was once only a concern of operating system programmers but since the late 1990s has become a commonplace issue. In a multi-threaded program, several threads execute simultaneously in a shared address space. Every thread has access to virtually all the memory of every other thread. Thus the flow of control and the sequence of accesses to data often have little relation to what would be reasonably expected by looking at the text of the program, violating the principle of least astonishment. Thread safety is a property aimed at minimizing surprising behavior by re-establishing some of the correspondences between the actual flow of control and the text of the program.“ [<http://en.wikipedia.org/wiki/Threadsafe>] (Siehe auch: https://en.wikipedia.org/wiki/Principle_of_least_astonishment)

„A subroutine is reentrant, and thus thread-safe, if 1) the only variables it uses are from the stack, 2) execution depends only on the arguments

passed in, and 3) the only subroutines it calls have the same properties. Such a sub-routine is sometimes called a "pure function", and is much like a mathematical function." Auf Deutsch nennt man das auch „frei von Seiteneffekten“.

5. Threads und Linux

Linux Thread-Bibliothek

- The Native POSIX Thread Library (NPTL)
 - Entwickelt von Red Hat, seit Kernel 2.6 unterstützt
 - Standard-Implementierung für POSIX-Threads und Teil der glibc
 - Ein POSIX-Thread wird auf einen Linux-Thread abgebildet

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

447

Siehe: https://en.wikipedia.org/wiki/Native_POSIX_Thread_Library

Threads und Linux (2)

Linux-Threads

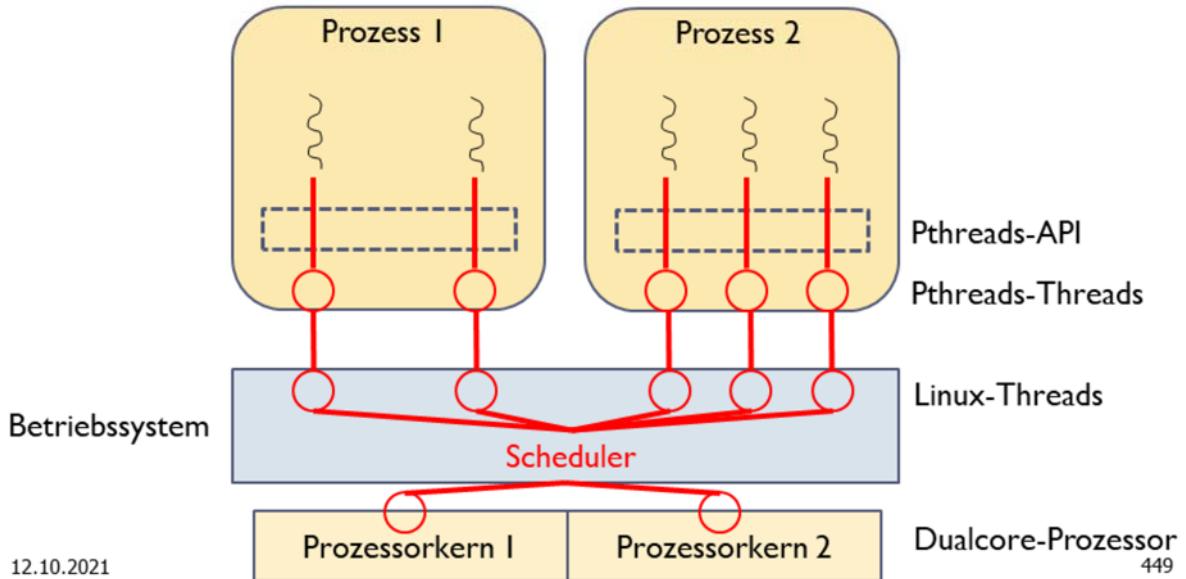
- Spezielle Variante der allgemeinen Prozesse
- Schnell erzeugbar, effiziente Nutzung
- Teilen sich alle Ressourcen mit Eltern-Prozess
- Erzeugt mit `clone()`-Aufruf (wie Kindprozesse auch)
 - Andere Parameter gestatten gemeinsame Ressourcennutzung

Kernel-Threads in Linux

- Spezielles Thread-Objekt im Betriebssystemkern
- Kein eigener Adressraum
- Arbeiten nur im Betriebssystemmodus
- Sind allerdings dem Scheduler unterworfen
- Zur Strukturierung von Aktivitäten im Kernel

Threads und Linux (3)

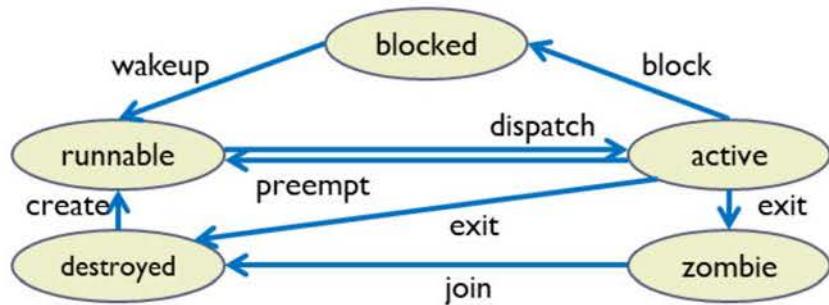
Pthreads-Threads – Linux-Threads - Prozessorkerne



Pthreads-Threads werden 1:1 auf Linux Threads abgebildet und diese vom Scheduler den Prozessorkernen zugewiesen.

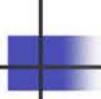
Threads und Linux (4)

- Scheduling: Zuteilung rechenbereiter Threads an Prozessorkerne
- Präemptives Scheduling: rechnender Thread kann unterbrochen werden
- Threadzustände
(POSIX-Implementierungsmodell)



6. Bewertung

	Pthreads	OpenMP	MPI
Skalierbarkeit	Begrenzt	Begrenzt	Ja
Fortran / C und C++	Ja / Ja	Ja / Ja	Ja / Ja
Hohe Abstraktion	Nein	Ja	Nein
Leistungsorientierung	Nein	Ja	Ja
Portierbarkeit	Ja	Ja	Ja
Herstellerunterstützung	Unix/SMP	Verbreitet	Verbreitet
Inkrement. Parallelisierung	Nein	Ja	Nein



Programmierung mit Threads

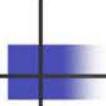
Zusammenfassung

- Threads trennen Einheiten der Ressourcenbelegung (Prozesse) von Einheiten der Prozessorzuteilung
- Threads werden für echt parallele Programme aber auch als Strukturierungsmittel eingesetzt
- Threads teilen sich den Adressraum des sie erzeugenden Prozesses
- Die Pthreads-Schnittstelle definiert einen Standard zur Nutzung von Threads
- Pthreads werden auf die Threads des Betriebssystems umgesetzt, die dann auf die Prozessorkerne des Systems abgebildet werden
- Alle Bibliotheken müssen heutzutage thread-sicher sein

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

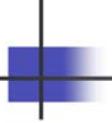
452



Programmierung mit Threads

Die wichtigsten Fragen

- Worin unterscheiden sich Threads und Prozesse?
- Warum sind Threads ein Thema beim Hochleistungsrechnen?
- Wann programmieren wir mit Threads?
- Welche Grundoperationen bietet die Pthreads-Schnittstelle?
- Wie werden Threads erzeugt?
- Wie funktioniert ein Mutex?
- Was versteht man unter thread-sicheren Funktionsaufrufen?
- Wie sind Threads im Linux-Betriebssystemkern genutzt?



Hybride Programmierung

1. Hybride Rechnerarchitekturen
2. Programmiermodelle und Multithreading
3. Beispiel Strömungsmechanik
4. OpenMP 4.0
5. Nutzung in realen Systemen
6. Beispiel Klimamodellierung

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

454

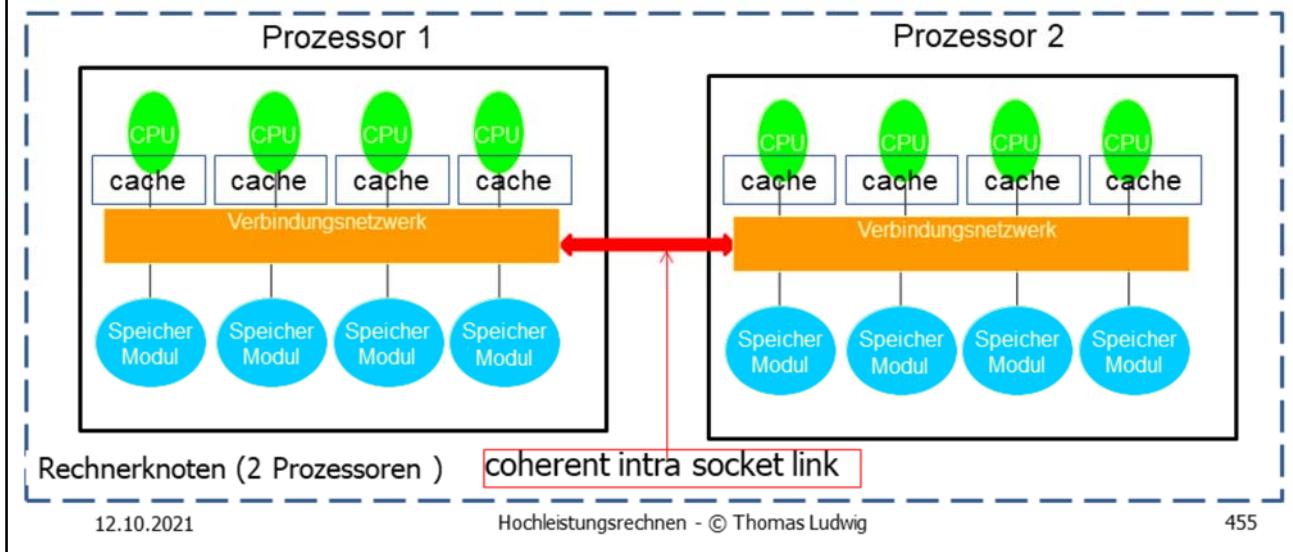
Folien Copyright Panos Adamidis, DKRZ, Raum 213, adamidis@dkrz.de

Siehe:

- <http://glaros.dtc.umn.edu/gkhome>
- <http://staffweb.cms.gre.ac.uk/~c.walshaw/partition>
- „Communication Characteristics and Hybrid MPI/OpenMP Parallel Programming in Clusters of Multi-core SMP Nodes”, Georg Hager, Gabriele Jost, Rolf Rabenseifner
- „Performance Characteristics of Hybrid MPI/OpenMP Implementations of NAS Parallel Benchmarks SP and BT on large-scale Multicore Clusters”, Xingfu Wu and Valerie Taylor

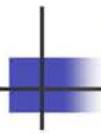
1. Hybride Rechnerarchitekturen

cache coherent non-uniform memory access
gemeinsamer Speicher



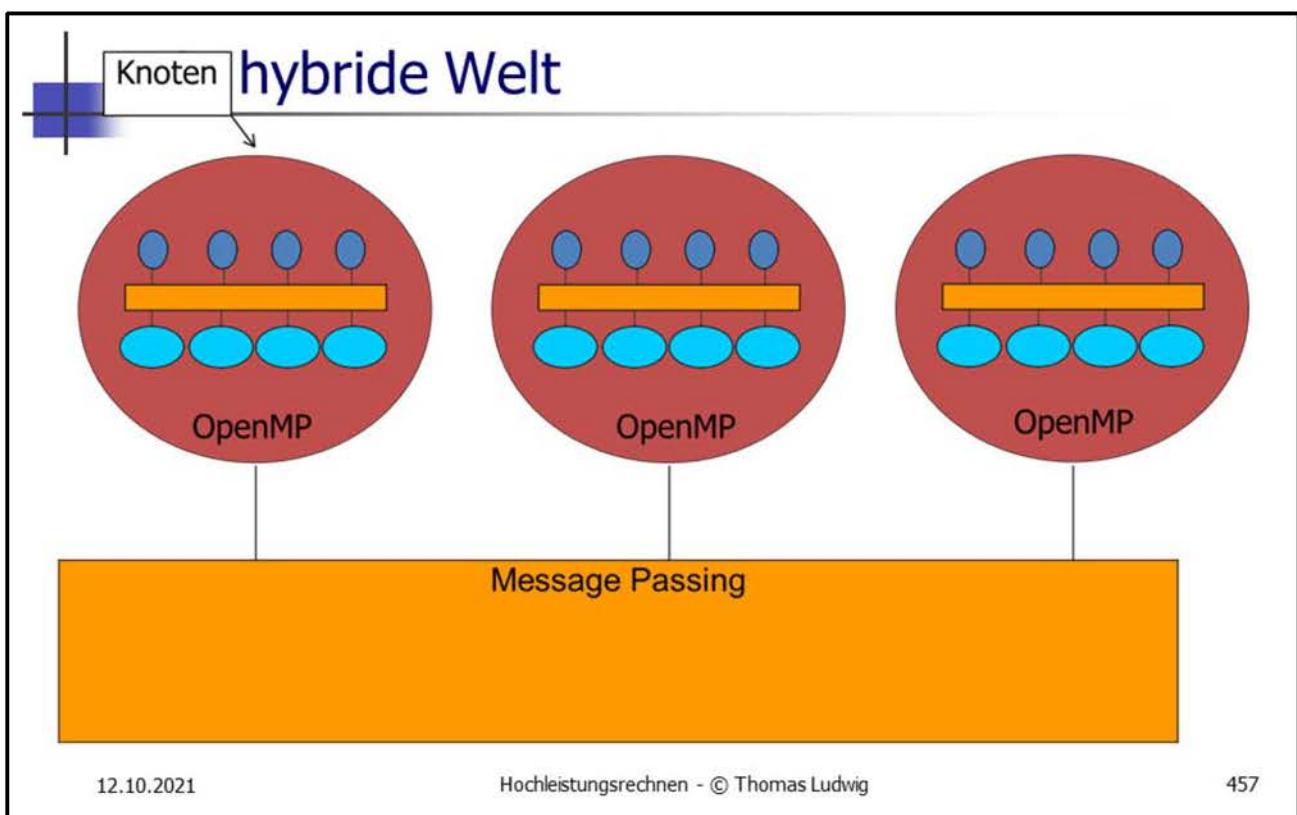
Siehe:

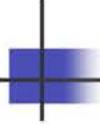
- https://en.wikipedia.org/wiki/Non-uniform_memory_access



ccNUMA – Gemeinsamer Speicher

- Hardware mit verteiltem Speicher
- Logisch ist ein gemeinsamer Speicher sichtbar
 - Ein Adressraum
- Eine Betriebssysteminstanz
- Unterschiedliche Zugriffszeiten je nachdem ob Zugriffe im lokalen oder entfernten Speicher stattfinden





Engpässe von hybriden Systemen

- Verbindungsnetzwerk
- Speicherbandbreite
- Ruhende Prozessorkerne

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

458



Warum hybride Parallelisierung

- Der logische Schritt, um hybride Architekturen zu programmieren, ist ein hybrides Programmiermodell
- Minimierung des Zusatzaufwands der MPI-Kommunikation innerhalb der Rechnerknoten
- Hybride Programmiermodelle ermöglichen Multilevel-Parallelisierung von Anwendungen

Multilevel-Parallelisierung

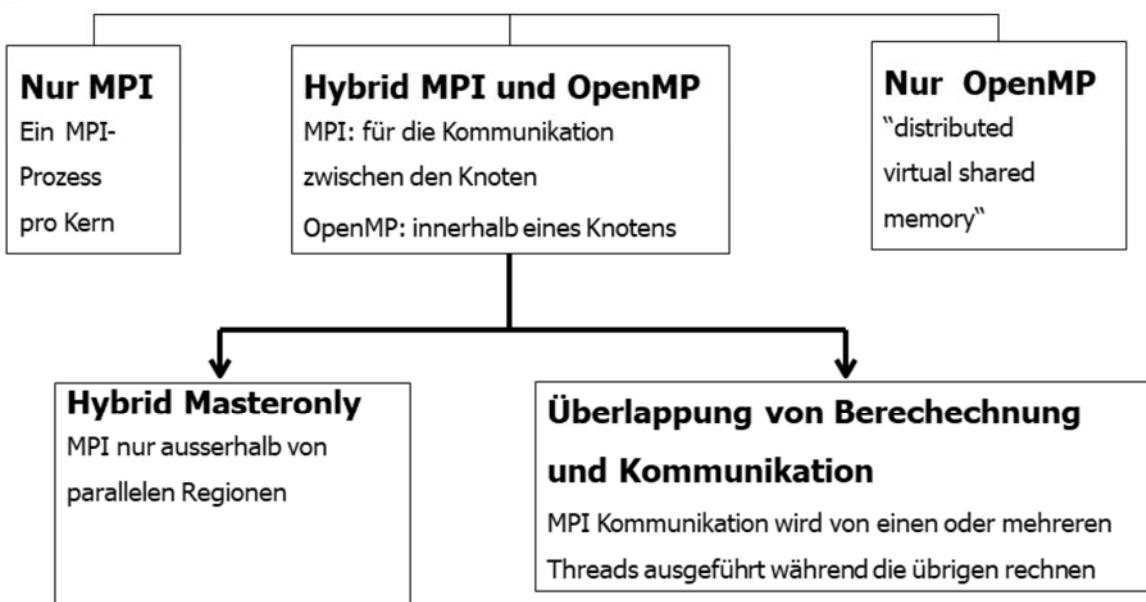
Schritt 1

- *coarse-grained parallelism*
 - Zerlegung des Problems in möglichst unabhängige Teilprobleme deren Berechnung gelegentlich Austausch von Informationen benötigt
 - Jedes Teilproblem wird auf einen MPI-Prozess abgebildet

Schritt 2

- *fine-grained parallelism*
 - Zusätzliche Parallelisierung mit OpenMP-Direktiven z.B. auf Schleifenebene

Programmiermod. für hybride Architekturen



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

461

Siehe: https://en.wikipedia.org/wiki/Distributed_shared_memory

2. Programmiermodelle und Multithreading

Unterstützung von Multithreading

- Die MPI-Bibliothek muss den Umstand berücksichtigen, dass mehrere Threads MPI-Routinen aufrufen können, ohne sich dabei gegenseitig stören zu können
 - thread-safe
- MPI-1-Standard unterstützte kein Multithreading
- MPI-2-und MPI-3-Standard haben Multithreading-Unterstützung

Prozesse und Threads

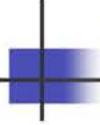
- Einzelne Threads sind nicht sichtbar außerhalb ihres Prozesses
- Die Threads führen MPI-Aufrufe im Auftrag ihres Prozesses aus
 - D.h. in den MPI-Routinen wird der **rank** des MPI-Prozesses benutzt und nicht die thread-id
MPI_Xxxx(.....,rank,.....)
 - Logisch: alle Threads haben denselben MPI-Rang

Unterstützung von Multithreading

- **MPI_Init**
 - Initialisierung ohne multithreading
- **MPI_Init_thread**
 - Initialisierung mit multithreading
- **MPI_Init_thread (int*argc, char**(*argv) [], int required, int* provided)**
 - **required**: der gewünschte Multithreading-Modus
 - **provided**: Rückgabewert, der den aktuellen Modus, der vom System unterstützt wird, enthält

Unterstützung von Multithreading...

- **MPI_THREAD_SINGLE**
 - Nur ein Thread pro Prozess
 - Ähnlich wie **MPI_Init**
- **MPI_THREAD_FUNNELED**
 - Der Prozess ist multithreaded aber nur der Master-Thread führt MPI-Aufrufe aus
- **MPI_THREAD_SERIALIZED**
 - Mehrere Threads können MPI-Routinen aufrufen, aber nur einer zu einem Zeitpunkt
- **MPI_THREAD_MULTIPLE**
 - Mehrere Threads dürfen MPI-Routinen aufrufen ohne jede Restriktion



Unterstützung von Multithreading...

Überlappung von Kommunikation und Berechnung erlauben

- **MPI_THREAD_FUNNELED**
- **MPI_THREAD_SERIALIZED**
- **MPI_THREAD_MULTIPLE**

Auswirkung auf die Korrektheit

- Je höher der Multithreading-Level desto höher die Komplexität der parallelen Algorithmen der Anwendungen
 - Einerseits kann man dadurch eine Effizienzsteigerung erreichen
 - Andererseits sind die Fehler die man dabei machen kann, schwieriger zu finden
- Die Implementierung der MPI-Bibliothek ist auch komplexer



Unterstützung von Multithreading

- OpenMPI und mpich2 unterstützen alle Multithreading-Level
- Fehlersuche in hybriden parallelen Programmen mit MPI/OpenMP mittels DDT
- Leistungsanalyse und Spuraufzeichnung ist mit VAMPIR möglich

Unterstützung von Multithreading...

- **int MPI_Query_thread (int *provided)**
Ein Thread kann den Multithreading-Modus abfragen, um sicher zu sein, dass es erlaubt ist Aufrufe der MPI-Routinen zu machen

- **int MPI_Is_thread_main (int *flag)**
 - Ein Thread kann herausfinden ob er der Master-Thread ist
 - Wichtig bei **MPI_THREAD_FUNNELED**

Beispiel

```
int thread_level, thread_is_main

MPI_Query_thread(&thread_level);
MPI_Is_thread_main(&thread_is_main);
If ((thread_level > MPI_THREAD_FUNNELED ) ||
    (thread_level == MPI_THREAD_FUNNELED &&
     thread_is_main)) {
    ... /* diese Threads duerfen MPI Routinen aufrufen */
else {
    printf("Fehler: Dieser Thread darf keine MPI
           Kommunikation machen \n")
}
...
...
```

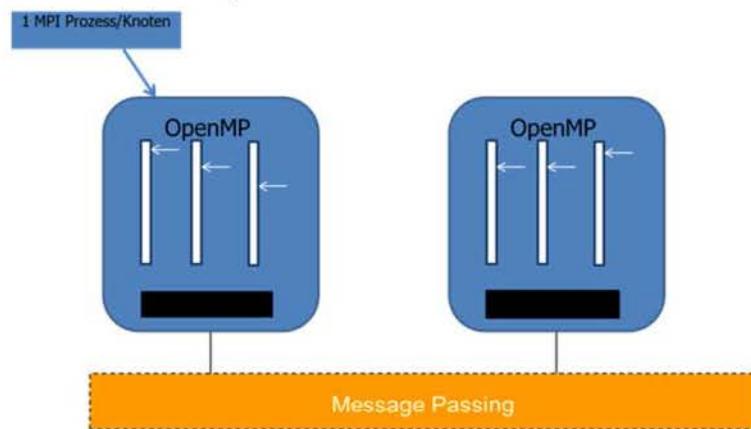
12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

470

Hybrid Master Only

- MPI-Aufrufe nur außerhalb von parallelen Regionen
- Üblicherweise 1 MPI-Prozess pro Knoten und 1 OpenMP-Thread pro Kern innerhalb des Knotens



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

471

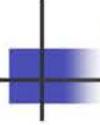
Hybrid Master Only

```
for (iteration = 1...n)
{
    #pragma omp parallel
    {
        /* compute something */
    }
    /* ON MASTER THREAD ONLY */
    MPI_Send(sendbuffer,...)
    MPI_Recv(recvbuffer,...)
}
```

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

472



Hybrid Master Only und OMP MASTER

```
#pragma omp barrier
#pragma omp master
    MPI_Xxx( . . . )
#pragma omp barrier
```

Hybrid Master Only und OMP MASTER

- **MPI_THREAD_FUNNELED** ist notwendig
- **OMP_MASTER** garantiert keine Synchronisation
- **OMP_BARRIER** ist notwendig, um sicher zu gehen, dass der Kommunikationspuffer nicht von anderen Threads benutzt wird

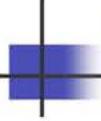
Beispiel mit MPI_Send

```
#pragma omp parallel
{
    #pragma omp for for(i=0;i<1000;i++)
        buf[i] = a[i];
    #pragma omp barrier
    #pragma omp master
        MPI_Send(buf,.....);
    #pragma omp barrier
    #pragma omp for for(i=0;i<1000;i++)
        buf[i] = c[i];
} /* omp end parallel */
```



Hybrid Master Only

- Vorteile
 - Keine MPI-Kommunikation innerhalb eines Rechnerknotens
 - Keine Topologie-Problem
- Nachteile
 - Während der Master kommuniziert, schlafen alle anderen Threads
 - Nur ein kommunizierender Thread (Master) kann in den meisten Fällen mit MPI nicht die volle Bandbreite des Netzes zwischen den Knoten ausnutzen



Hybrid mit Überlappung (Mixed Modell)

- Überlappung von Kommunikation und Berechnung
- Mehr als ein MPI-Prozess pro Knoten
- MPI-Aufrufe können von mehreren Threads ausgeführt werden
- MPI-Bibliothek muss Multithreading unterstützen

Hybrid Mixed

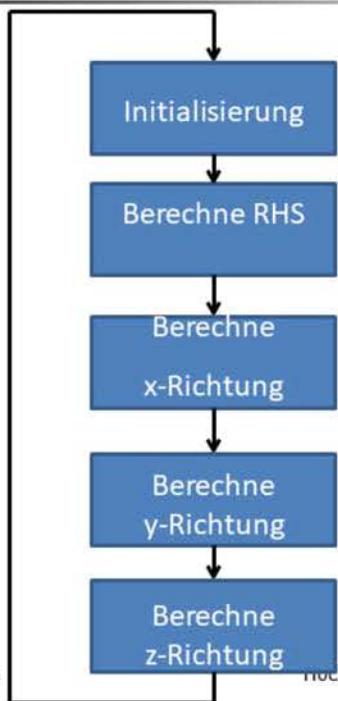
```
#pragma omp parallel
{
    if ( my_thread_id == id1 )      {
        MPI_Send(sendbuffer, ....)
    }
    else if ( my_thread_id == id2 ) {
        MPI_Recv(recvbuffer, ....)
    }
    else
    {
        /* compute something */
    }
}
```

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

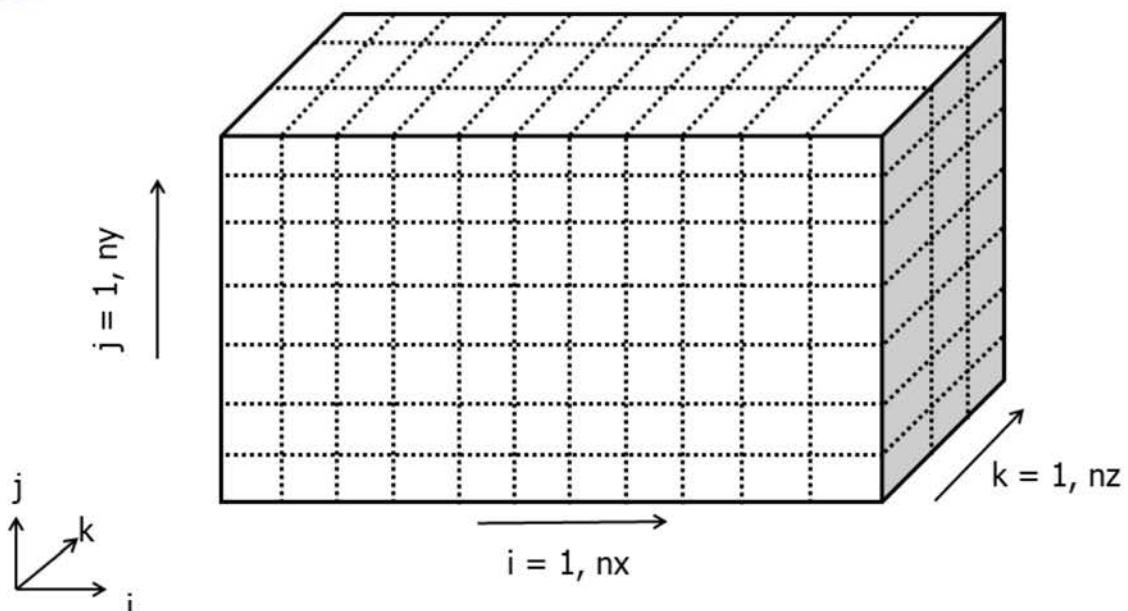
478

3. Beispiel Strömungsmechanik



3D Navier Stokes Gleichungen

3-dimensionales Rechengebiet

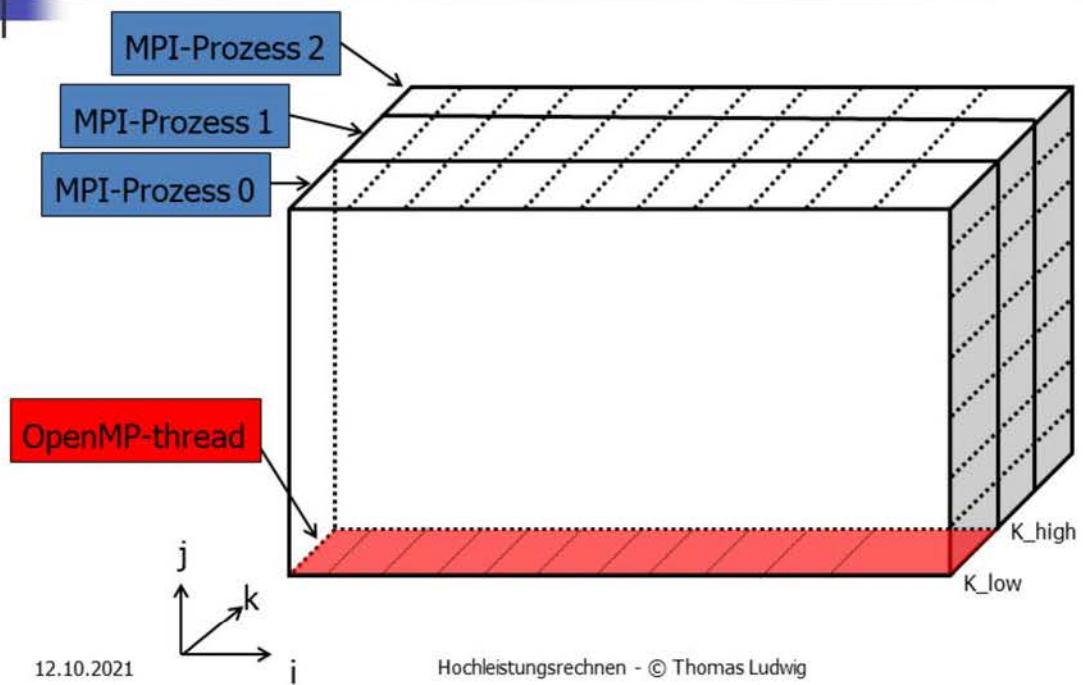


12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

480

1d-Gebietszerlegung in z-Richtung

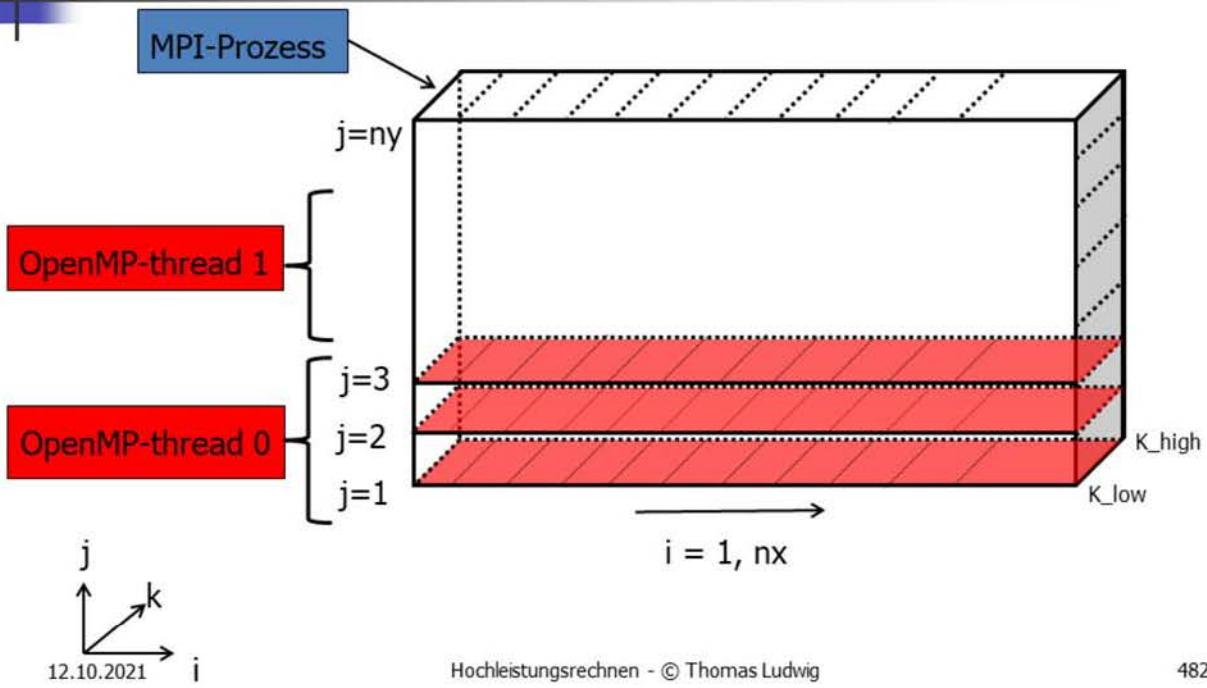


12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

481

1d-Gebietszerlegung in z-Richtung



Hybrid Mixed

```
MPI_Init_thread(MPI_THREAD_MULTIPLE,...)
#pragma omp for
for (j=1; j<ny; j++) {
    MPI_Receive(pid1,...)
    for (k=k_low; k<k_high; k++)
        for (i=1; i<nx; i++)
    {
        z[i,j,k] = z[i,j,k-1] + ...
    }
    MPI_Send(pid2,...)
}
```

4. OpenMP 4.0

SIMD für AVX

- **#pragma omp simd clauses**
 - Gruppen von Iterationen können parallel nach SIMD-Methode ausgeführt werden
 - Erzwingt Vektorisierung von Schleifen, die der Compiler nicht automatisch vektorisiert
 - Dem Compiler wird gesagt, er soll Abhängigkeiten ignorieren
 - Der Programmierer ist verantwortlich für Korrektheit

SIMD vs Scalar

SIMD Mode

Scalar Mode

Zeit

A0	A1	A2	A3	A4
+				
B0	B1	B2	B3	B4
=				
C0	C1	C2	C3	C4

A0
+
B0
=
C0

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

485

SIMD für AVX

- **#pragma omp declare simd clauses**
 - Erzwingt Vektorisierung von „*functions*“ oder „*subroutines*“
 - Traditionell akzeptieren Funktionen Skalare als Argumente und liefern Skalare als Rückgabewert zurück
 - Ein Funktionsaufruf innerhalb einer Schleife stellt somit ein Hindernis für die Vektorisierung dar
 - Der Compiler wird instruiert, spezielle Vektor-Varianten der skalaren Funktion zu erzeugen

SIMD

```
#pragma omp declare simd
float seqfun(float) {
    // function body
}

void callseqfun(float *restrict a, float *restrict x,
int n) {
#pragma omp simd
    for (int i=0; i<n; i++)
        a[i]=seqfun(x[i]);
}

Compiler generiert eine Vektorvariante von seqfun
seqfun → vecfun (<x0,x1,x2,x3>)
```

SIMD für Funktionen

Scalar

Zeit

```
call seqfun(x[0]) -> r0  
store a[0],r0  
call seqfun(x[1])-> r1  
store a[1],r1  
call seqfun(x[2])-> r2  
store a[2],r2  
call seqfun(x[3])-> r3  
store a[3],r3
```

SIMD

```
call vecfun(x[0..3])->vecreg0  
simdstore a[0..3], vecreg0
```

Threading and SIMD

- ```
#pragma omp parallel for simd clauses
for (i=0; i<n; i++)
 x = x + A[i];
```
- Die Iterationen werden auf mehrere Threads aufgeteilt
- Jeder Thread wird in SIMD-Mode ausgeführt

## Hybrid Mixed

```
MPI_Init_thread(MPI_THREAD_MULTIPLE,...)
#pragma omp for
for (j=1; j<ny; j++) {
 MPI_Receive(pid1,...)
 for (k=k_low; k<k_high; k++)
#pragma omp simd
 for (i=1; i<nx; i++)
 {
 z[i,j,k] = z[i,j,k-1] + ...
 }
 MPI_Send(pid2,...)
}
```

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

490



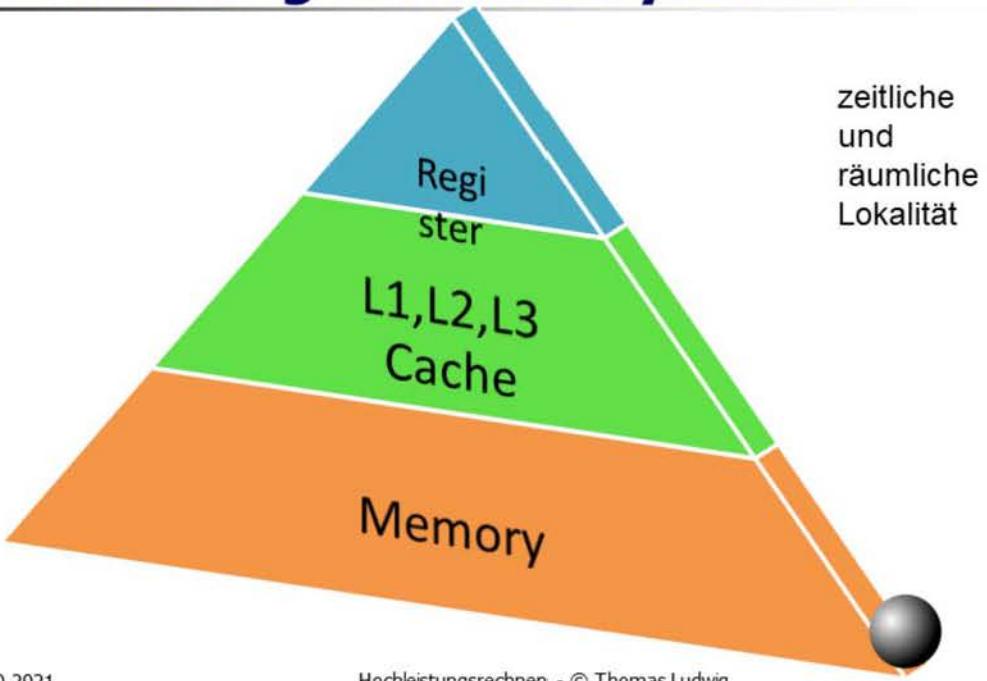
## OpenMP und MPI-Non-Blocking-Kommunikation

- Es ist nicht erlaubt
  - dass mehrere Threads ein **MPI\_Wait** oder **MPI\_Test** auf eine und dieselbe MPI-Non-Blocking-Operation durchführen
- Es ist erlaubt
  - dass ein Thread eine Non-Blocking-Operation startet und ein anderer diese abschließt. Es dürfen aber nicht zwei Threads versuchen diese zu beenden!

## OpenMP und parallele E/A

- OpenMP spezifiziert nichts über parallele E/A
- Programmierer ist verantwortlich, die parallele E/A im Multithreaded-Programm sicherzustellen
- In Fortran führt nicht-synchronisierte E/A auf demselben „unit“ zu einem nicht-spezifiziertem Verhalten
- In hybriden Programmen kann man weiterhin parallele E/A in der MPI-Welt machen

## 5. Nutzung in realen Systemen

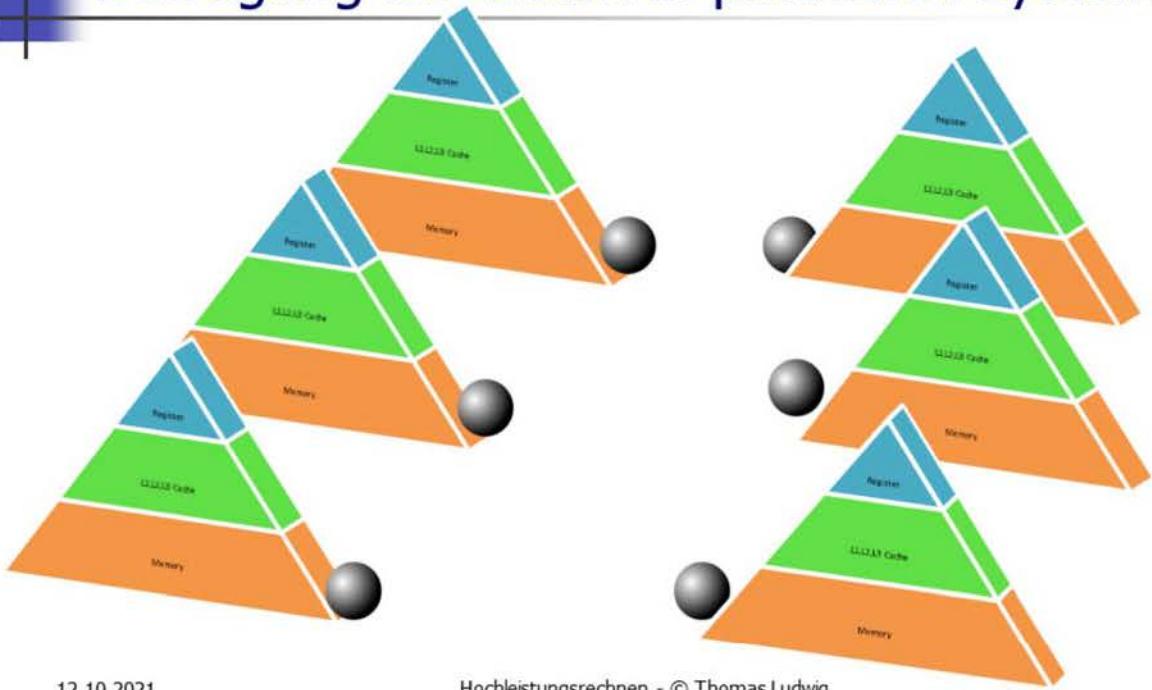


12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

493

# Bewegung der Daten in parallelen Systemen



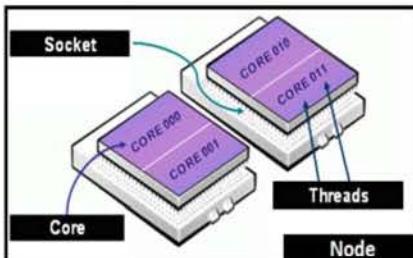
12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

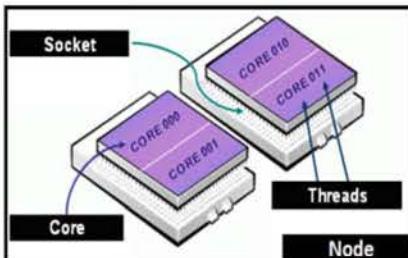
494

# SLURM Process and Thread Binding

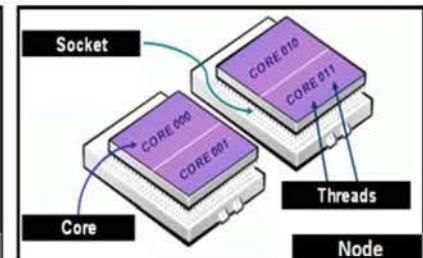
- **HSW node** : NUMA board
  - 2 sockets mit 12 cores
  - Jeder core 2 CPUs/HyperThreads
- **Abbildung der Prozesse** : Verteilung der ranks auf Knoten
  - srun --distribution=<block|cyclic[:block|cyclic]>
  - Erstes Argument: block oder cycle auf sukzessive Knoten



12.10.2021



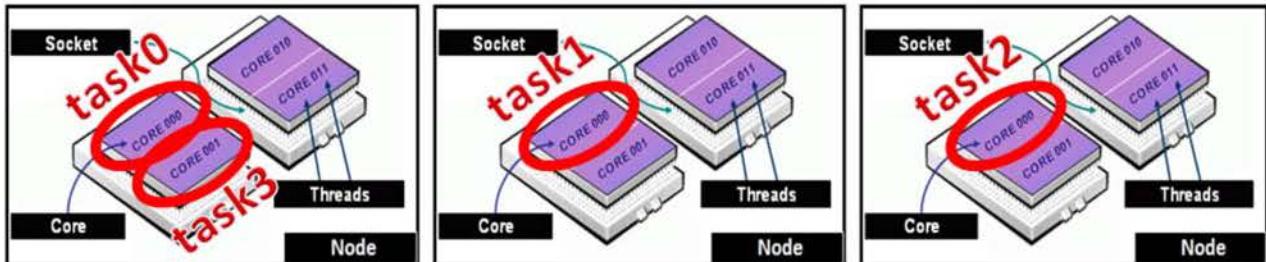
Hochleistungsrechnen - © Thomas Ludwig



495

# SLURM Process und Thread Binding

srun --distribution=cyclic:block



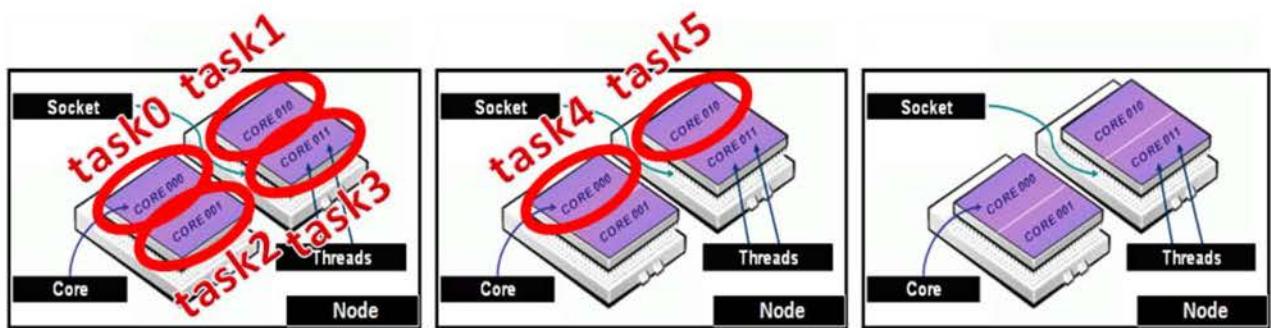
12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

496

# SLURM Process und Thread Binding

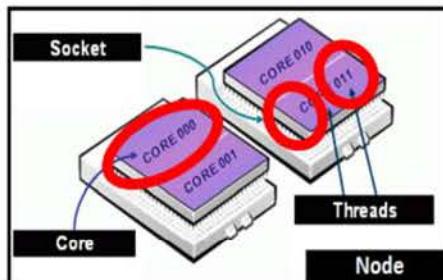
srun --distribution=block:cyclic



mistral default is block:block

## SLURM Process und Thread Binding

- Binding ranks to cores, cpus, ...  
`srun --cpu_bind=<[verbose,]type>`
  - bind to a single core: <type> = cores
  - bind to a single CPU/HyperThread: <type> = threads
  - custom bindings: <type> = map\_cpu:<list>



12.10.2021

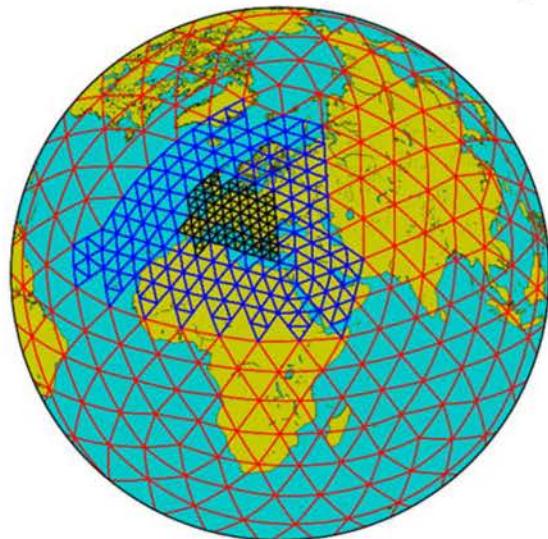
498

## Abbildungsproblem

- Finde die optimale Abbildung der Prozesstopologie der Anwendung auf die Topologie der Hardware
  - Finde optimalen Lastausgleich aller Recheneinheiten
  - Minimiere den Kommunikationsaufwand
  - Dieses Problem ist NP-vollständig
- Lösungsansätze
  - Die Prozesstopologie der Anwendung wird auf Graphen abgebildet
  - Das Lastausgleichproblem wird als Graphenpartitionierungsproblem formuliert
  - Die Partitionierungsalgorithmen basieren auf Heuristiken
  - Bekannte Bibliotheken: METIS, Jostle

## 6. Beispiel Klimamodellierung

ICON (ICOsaHedral Nonhydrostatic) auf „Mistral“



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

500

# ICON auf Mistral mit 64 Knoten

Ohne Hypertreading

| MPI_tasks x threads | Time in sec | Performance Gain % |
|---------------------|-------------|--------------------|
| 24x1                | 264,73      |                    |
| 12x2                | 191,95      | 27,4%              |
| 6x4                 | 185,17      | 30,0%              |
| 4x6                 | 174,33      | <u>34,1%</u>       |
| 2x12                | 183,96      | 30,5%              |
| 1x24                | 242,27      | 8,4%               |

Mit Hypertreading

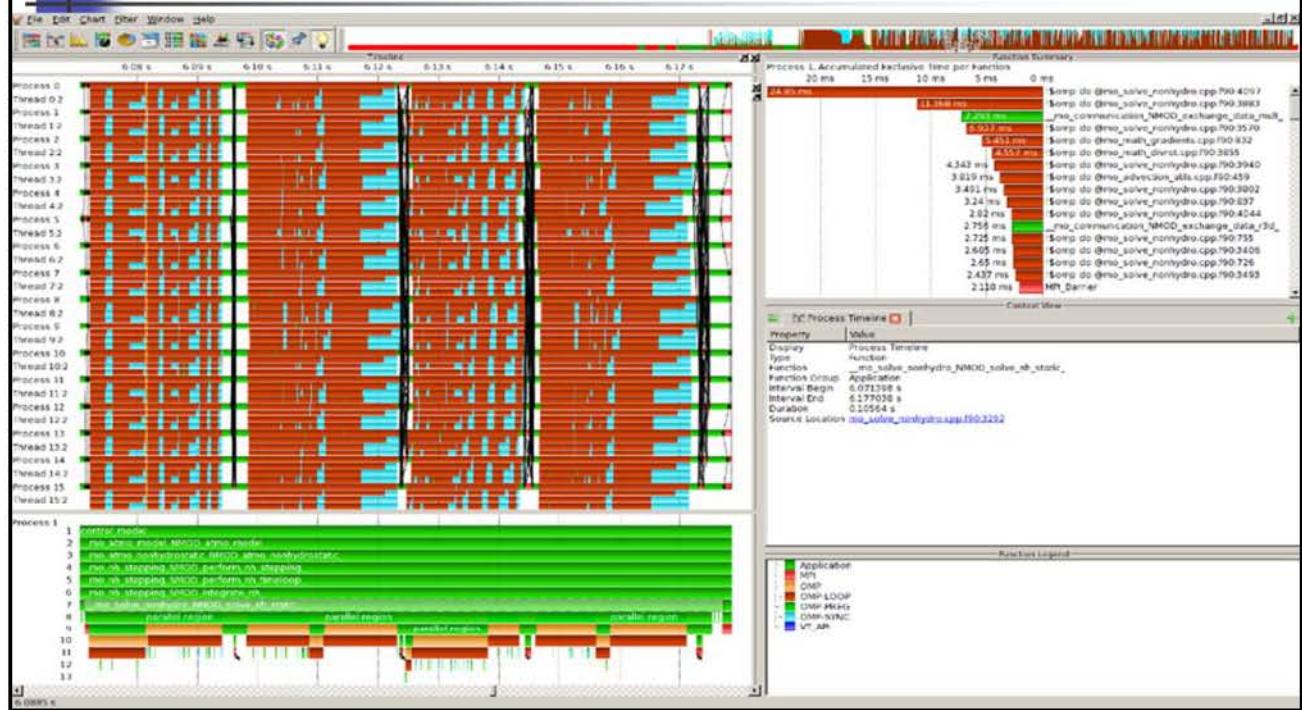
| MPI_tasks x threads | Time in sec | Performance Gain % |
|---------------------|-------------|--------------------|
| 24x2                | 216,57      | 18,2%              |
| 12x4                | 194,96      | 26,3%              |
| 6x8                 | 179,20      | 32,3%              |
| 8x6                 | 185,21      | 30,03%             |
| 4x12                | 164,76      | <u>37,7%</u>       |
| 2x24                | 175,14      | 33,8%              |
| 1x48                | 240,82      | 9,03%              |

12.10.2021

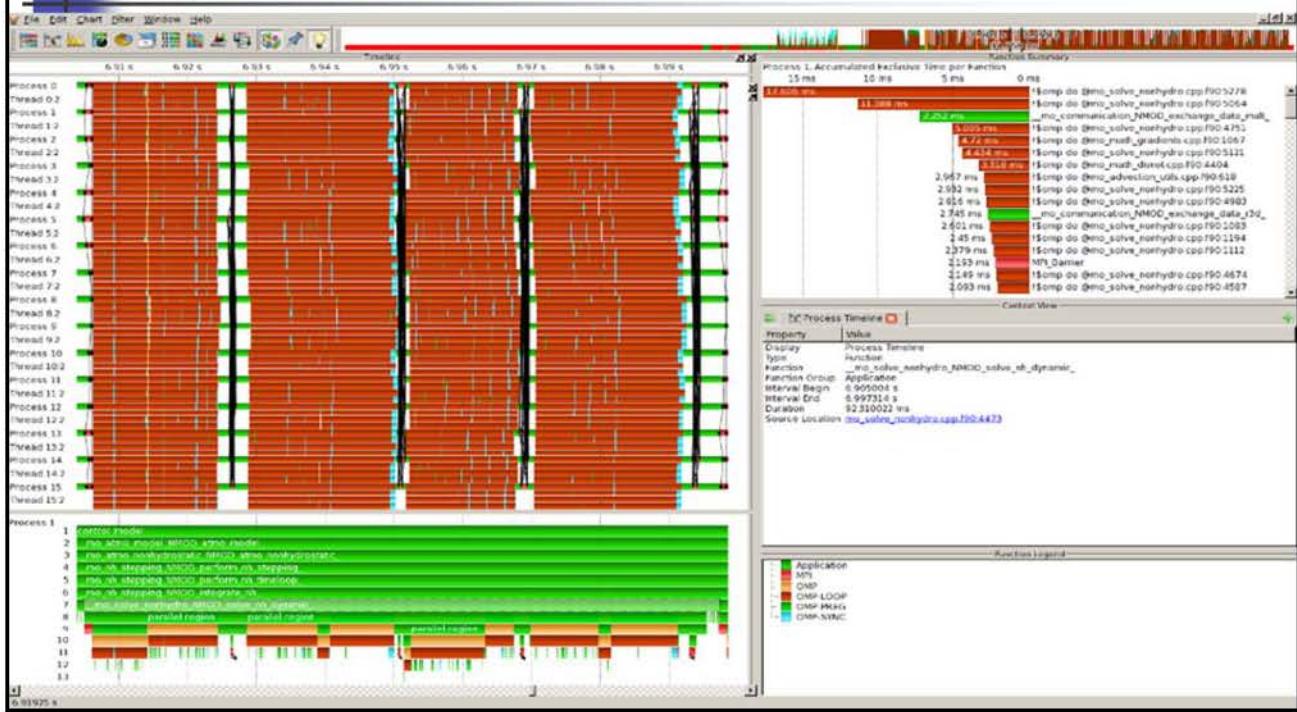
Hochleistungsrechnen - © Thomas Ludwig

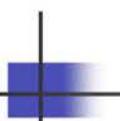
501

# Statischer Schedule



# Dynamischer Schedule

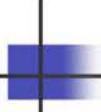




## Hybride Programmierung

### Zusammenfassung

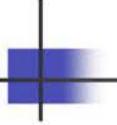
- Hybride MPI/OpenMP-Parallelisierung kann die Skalierbarkeit von Anwendungen auf modernen hybriden Architekturen steigern
- MPI-Kommunikation innerhalb Rechnerknoten wird vermieden
- Multilevel-Parallelisierung wird dadurch ermöglicht
- Die MPI-Bibliothek muss thread-safe sein
- Die Komplexität der hybrid parallelen Programme ist grösser im Vergleich zu reinen MPI- oder OpenMP-Programmen
- Auswirkungen auf Korrektheit
  - Die Fehler sind schwerer zu finden
- Auswirkungen auf Effizienz
  - Zusatzaufwand der Generierung und Synchronisation der Threads
  - Es ist schwierig die optimale Anzahl der OpenMP-Threads pro MPI-Prozess zu bestimmen
- Ein guter Lastausgleich zwischen allen Recheneinheiten des Systems hat sehr große Auswirkung auf die Effizienz des parallelen Programms



## Hybride Programmierung

Die wichtigsten Fragen

- Wie sieht die Architektur eines ccNuma-Rechenknotens aus?
- Was versteht man unter hybrider Programmierung?
- Wie ist das prinzipielle Vorgehen?
- Welche Varianten bzgl. des Multithreadings gibt es?
- Wie sieht die typische Gebietsaufteilung eines 3-dimensionalen Rechengebietes aus?
- Welche Aufgaben übernimmt der Scheduler bei der Abbildung von Prozessen und Threads?



# **Mathematische Bibliotheken**

1. Mathematische Bibliotheken: Wieso? Weshalb? Warum?
2. Numerische Lineare Algebra und andere Grundbausteine
3. Löserpakete für Wissenschaftliche Anwendungen/PDEs
4. Software-Frameworks
5. Zusammenfassung und Wiederholung



# Mathematische Bibliotheken

1. Mathematische Bibliotheken: Wieso? Weshalb? Warum?
2. Numerische Lineare Algebra u. Von Low- zu High-Level he  
Löserpakete für Wissenschaftl. Code und math.  
Software-Frameworks
3. Löserpakete für Wissenschaftl. Code und math.  
Beschreibung
4. Software-Frameworks
5. Zusammenfassung und Wiederholung

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

507

# 1. Mathematische Bibliotheken: Wieso? Weshalb? Warum? (1)

Mathematische Grundfunktionalitäten als Bausteine in Vielzahl von Problemstellungen des wissenschaftlichen Rechnens und darüber hinaus

- Numerische Simulation
  - Lösen von (nicht-)linearen Gleichungssystemen, bspw. Poisson-Gleichung zur Bestimmung des physikalischen Drucks in inkompressiblen Strömungen
  - Vektor- und Matrixoperationen zur Lösung von ODEs, bspw. chemische Reaktionsprozesse
  - Prä-Konditionierung schlecht konditionierter Probleme, bspw. in der Struktur-/Fluidmechanik
  - ...

Lösen von gekoppelten ODEs: Beispiel

# 1. Mathematische Bibliotheken: Wieso? Weshalb? Warum? (2)

Mathematische Grundfunktionalitäten als Bausteine in  
Vielzahl von Problemstellungen des wissenschaftlichen  
Rechnens und darüber hinaus

- Datenanalyse
  - Glätten von Daten durch Matrix-Vektor-Operationen
  - Datenfilter, bspw. via Fourieranalysen und Hauptkomponentenanalysen
  - ...

Lösen von gekoppelten ODEs: Beispiel

# 1. Mathematische Bibliotheken: Wieso? Weshalb? Warum? (3)

Mathematische Grundfunktionalitäten als Bausteine in Vielzahl von Problemstellungen des wissenschaftlichen Rechnens und darüber hinaus

- Maschinelles Lernen
  - Algorithmik innerhalb Deep Learning/Neuronaler Netze basierend auf Vektor/Matrix/Tensor-Operationen
  - Spezifische Gleichungssysteme in Regressionsansätzen, bspw. Least-Squares-Systeme (symmetrisch positiv definite Matrizen)
  - ...
  - Fun Fact: Schließung des Kreises  
„Lineare Algebra → Maschinelles Lernen → Lineare Algebra“, siehe bspw. Götz und Anzt (2018) für Prä-Konditionierung

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

510

Götz, Anzt: Machine learning-aided numerical linear algebra:  
Convolutional neural network for the efficient preconditioner generation,  
9th Workshop on Latest Advances in Scalable Algorithms for Large-Scale  
Systems, held in conjunction with SC18, 2018

## **1. Mathematische Bibliotheken: Wieso? Weshalb? Warum? (4)**

Vielzahl an sich schnell weiterentwickelnder Hardware

- Mathematische Bibliotheken als Baustein zwischen hardware-optimierter Grundfunktionalität und anwendungsspezifischer Problemstellung
- Hardware-spezifische Bibliotheken, bspw. für Intel-Architekturen, Nvidia GPUs



## **1. Mathematische Bibliotheken: Warum nicht?**

**Gruppen- bzw.  
Einzelarbeit ☺  
3 Min.**

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

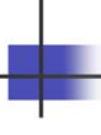
512

Reproduzierbarkeit: Fehlersuche essentiell; Problem für komplexe Software, falls bitweise Reproduzierbarkeit nicht gegeben ist (Beispiel: Klimamodelle)

Granularität: A priori ist bei Software-Entwicklung optimales Level an notwendiger Funktionalität nicht immer klar

Bedienbarkeit/Nutzbarkeit: siehe PETSc-Beispiel: viele Zeilen an Code, nicht immer trivial, aber dafür eine Vielzahl an Optionen

Nachhaltigkeit: Welche Bibliothek wird wie lange von wem unterstützt?



# 1. Mathematische Bibliotheken: Warum nicht?

- Software-Abhangigkeiten
- Bitweise Reproduzierbarkeit
- Granularitat: Low- vs. High-Level-Implementierungen
- Bedienbarkeit/Nutzbarkeit
- Nachhaltigkeit: Long-term Support fur Bibliotheken?

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

513

Reproduzierbarkeit: Fehlersuche essentiell; Problem fur komplexe Software, falls bitweise Reproduzierbarkeit nicht gegeben ist (Beispiel: Klimamodelle)

Granularitat: A priori ist bei Software-Entwicklung optimales Level an notwendiger Funktionalitat nicht immer klar

Bedienbarkeit/Nutzbarkeit: siehe PETSc-Beispiel: viele Zeilen an Code, nicht immer trivial, aber dafur eine Vielzahl an Optionen

Nachhaltigkeit: Welche Bibliothek wird wie lange von wem unterstutzt?

## 2. Numerische Lineare Algebra und andere Grundbausteine

- **BLAS, ATLAS**
- **LAPACK, Scalapack**
- **NumPy**
- Hypre: Scalable Linear Solvers and Multigrid Methods
  - Weitere Informationen:  
<https://computation.llnl.gov/projects/hypre-scalable-linear-solvers-multigrid-methods>
- ParMETIS
  - MPI-parallele Bibliothek zur Partitionierung unstrukturierter Graphen
  - Weitere Informationen:  
<http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview>
- FFTW (Fastest Fourier Transform in the West)
  - Weitere Informationen: <http://www.fftw.org/benchfft/>

## BLAS

- BLAS=Basic Linear Algebra Subprograms, eingeteilt in 3 Levels
- Implementiert grundlegende Bausteine aus der linearen Algebra
  - Skalierung eines Vektors mit Konstante
  - Matrix-Vektor-Multiplikation
  - Matrix-Matrix-Multiplikation (GEMM=general matrix-matrix multiply)
  - Spezialisierungen für symmetrische, hermitische/Dreiecks-/Bandmatrizen
  - ...
- Verschiedenste Implementierungen (inklusive Hardware-Hersteller-Implementierungen)
  - OpenBLAS
  - Intel Math Kernel library (MKL), Nvidia's cuBLAS
  - Automatically tuned linear algebra software (ATLAS)
- Weitere Informationen: [www.netlib.org/blas/](http://www.netlib.org/blas/)

# BLAS

- BLAS=Basic Linear Algebra Subprograms, eingeteilt in 3 Levels
- Implementiert grundlegende Bausteine aus der linearen Algebra
  - Skalierung eines Vektors mit Konstante
  - Matrix-Vektor-Multiplikation
  - Matrix-Matrix-Multiplikation (GEMM=general matrix-matrix multiply)
  - Spezialisierungen für symmetrische, hermitische/Dreiecks-/Bandmatrizen
  - ...
- Verschiedenste Hersteller-Implementierungen:
  - OpenBLAS
  - Intel Math Kernel Library (MKL), NVIDIA's CUBLAS
  - Automatically tuned linear algebra software (ATLAS)
- Weitere Informationen: [www.netlib.org/blas/](http://www.netlib.org/blas/)

Was nützt Selbstdaption/  
Auto-Tuning für BLAS und  
HPC?

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

516

Selbstdaption für: Anpassung an unterschiedliche Hardware (bspw. Cache-Größen, siehe nächste Folie, Anzahl und Geschwindigkeit von Rechenkernen, Befehlssätze), Optimierung der Algorithmik (Hardware- oder Problemspezifisch)

# ATLAS

- Ziel: Automatische Optimierung für spezifische Hardware
- Beispiel: Matrix-Matrix-Multiplikation  
→ Optimales Cache-Blocking
- Weitere Informationen:
  - <http://www.netlib.org/lapack/lawnspdf/lawn131.pdf>
  - R.C. Whaley, J.J. Dongarra. Automatically tuned linear algebra software. SC '98 proceedings, pp. 1-27, 1998

$$\left[ \quad \right] = \left[ \quad \right] \times \left[ \quad \right]$$

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

517

# ATLAS

- Ziel: Automatische Optimierung für spezifische Hardware
- Beispiel: Matrix-Matrix-Multiplikation  
→ Optimales Cache-Blocking
- Weitere Informationen:
  - <http://www.netlib.org/lapack/timing.html>
  - R.C. Whaley, J.J. Dongarra, A. Petitet, and S. Parashar. SCALAPACK: A library of parallel numerical linear algebra routines for distributed memory architectures. *SCALAPACK Report*, University of Tennessee, Dept. of Computer Science, Technical Report CS-97-13, 1997.

Tuning der Block-  
größe [ ] zur  
Verhinderung von  
Cache Misses

pdf  
linear  
1998

$$\begin{pmatrix} [ ] \\ [ ] \end{pmatrix} = \begin{pmatrix} [ ] & [ ] \\ [ ] & [ ] \end{pmatrix} \times \begin{pmatrix} [ ] \\ [ ] \end{pmatrix}$$

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

518

# ATLAS

- Ziel: Automatische Optimierung
- Beispiel: Matrix-Matrix-Multiplikation  
→ Optimales Cache-Blocking
- Weitere Informationen:
  - <http://www.netlib.org/atlas/>
  - R.C. Whaley, J.J. Dongarra, ATLAS: An Environment for Auto-Tuning of Linear Algebra Software. SC 1998

Auto-Tuning ist Gegenstand aktueller HPC-Forschung

Tuning der Blockgröße zur Verhinderung von Cache Misses

pdf  
linear  
1998

$$\begin{pmatrix} \text{ } \\ \text{ } \end{pmatrix} = \begin{pmatrix} \text{ } \\ \text{ } \end{pmatrix} \times \begin{pmatrix} \text{ } \\ \text{ } \end{pmatrix}$$

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

519

## LAPACK

- LAPACK=Linear Algebra Package
- Implementierungen für
  - Lösung linearer Gleichungssysteme
  - Lineare Ausgleichsprobleme
  - Eigenwertprobleme
    - QR-Zerlegung, Householder-Transformation, Singulärwertzerlegung, ...
- Verwendung von BLAS
- Verschiedene Implementierungen (MKL, ...)
- Weitere Informationen: <http://www.netlib.org/lapack/>

# BLAS und LAPACK: Parallelisierung

- Parallelisierung und Threadsicherheit abhängig von Implementierung
- OpenBLAS<sup>1</sup>: „If your application is already multi-threaded, it will conflict with OpenBLAS multi-threading. Thus, you must set OpenBLAS to use single thread as following.
  - export OPENBLAS\_NUM\_THREADS=1 in the environment variables. Or
  - Call openblas\_set\_num\_threads(1) in the application on runtime. Or
  - Build OpenBLAS single thread version, e.g. make USE\_THREAD=0 USE\_LOCKING=1 (see comment below)
- If the application is parallelized by OpenMP, please build OpenBLAS with USE\_OPENMP=1“
- Intel oneAPI MKL<sup>2</sup>: „Is a seamless upgrade ... of the Intel Math Kernel Library“
  - „Data Parallel C++ (DPC++) APIs maximize performance and cross-architecture portability
  - Introduces C and Fortran OpenMP offload for Intel GPU acceleration“

1<https://github.com/xianyi/OpenBLAS/wiki/faq#multi-threaded>

2[https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/onemkl.html?wapkw=\(Intel\)](https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/onemkl.html?wapkw=(Intel))

## ScaLAPACK

- ScaLAPACK=Scalable LAPACK, entworfen für MIMD-Systeme
- Weitere Informationen:
  - [http://www.netlib.org/utk/people/JackDongarra/PAPERS/077\\_1996\\_scalapack-a-portable-linear-algebra-library-for-distributed-memory.pdf](http://www.netlib.org/utk/people/JackDongarra/PAPERS/077_1996_scalapack-a-portable-linear-algebra-library-for-distributed-memory.pdf)
  - J. Choi et al. ScaLAPACK: a portable linear algebra library for distributed memory computers — design issues and performance. Computer Physics Communications 97(1-2): 1-15, 1996
- Ähnliche Entwicklungen:
  - PLASMA=Parallel Linear Algebra Software for Multicore Architectures (<http://icl.cs.utk.edu/plasma/software/>)
  - MAGMA=Matrix Algebra on GPU and Multicore Architectures (<http://icl.cs.utk.edu/magma/>)

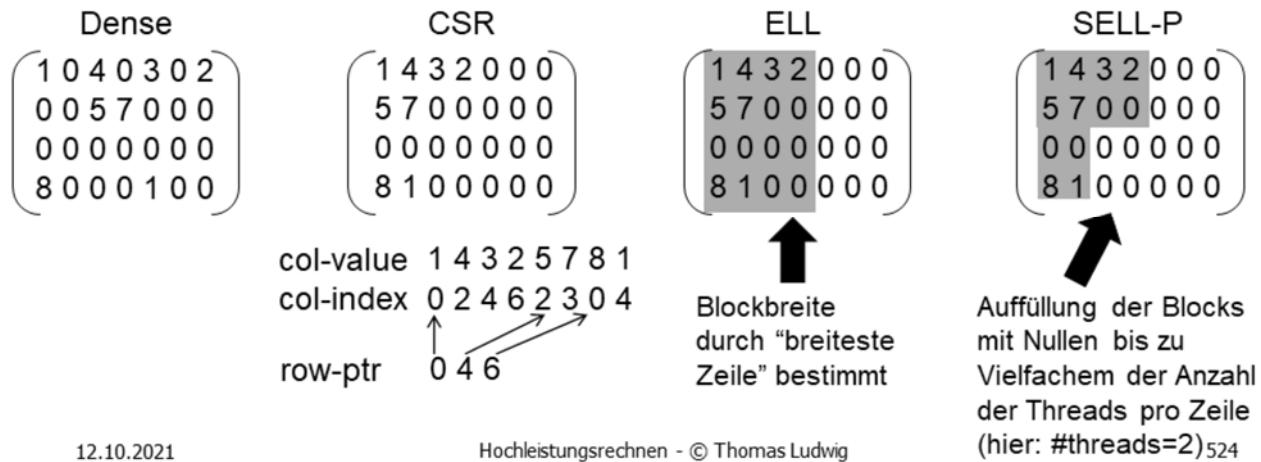
# ScaLAPACK

- ScaLAPACK=Scalable LAPACK, entworfen für MIMD-Systeme
- Weitere Informationen:
  - [http://www.netlib.org/utk/people/JackDongarra/PAPERS/077\\_1996\\_scalapack-a-portable-linear-algebra-library-for-distributed-memory.pdf](http://www.netlib.org/utk/people/JackDongarra/PAPERS/077_1996_scalapack-a-portable-linear-algebra-library-for-distributed-memory.pdf)
  - J. Choi et al. ScaLAPACK: a portable linear algebra library for distributed memory computers — design. Computer Physics Communications 97(1), 1996.
- Ähnliche Entwicklungen:
  - PLASMA=Parallel Linear Algebra Software for Architectures (<http://icl.cs.utk.edu/plasma/>)
  - MAGMA=Matrix Algebra on GPU and Multicore Architectures (<http://icl.cs.utk.edu/magma/>)

Entwickelt für  
heterogene  
Architekturen

# Exkurs: Dünnbesetzte Matrizen/SPMV (1)

- SPMV=Sparse Matrix Vector Multiplication
- Datenstrukturen (bspw. MAGMA):
  - Compressed Storage Row (CSR), ELLPACK (ELL), padded sliced ELLPACK (SELL-P)



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

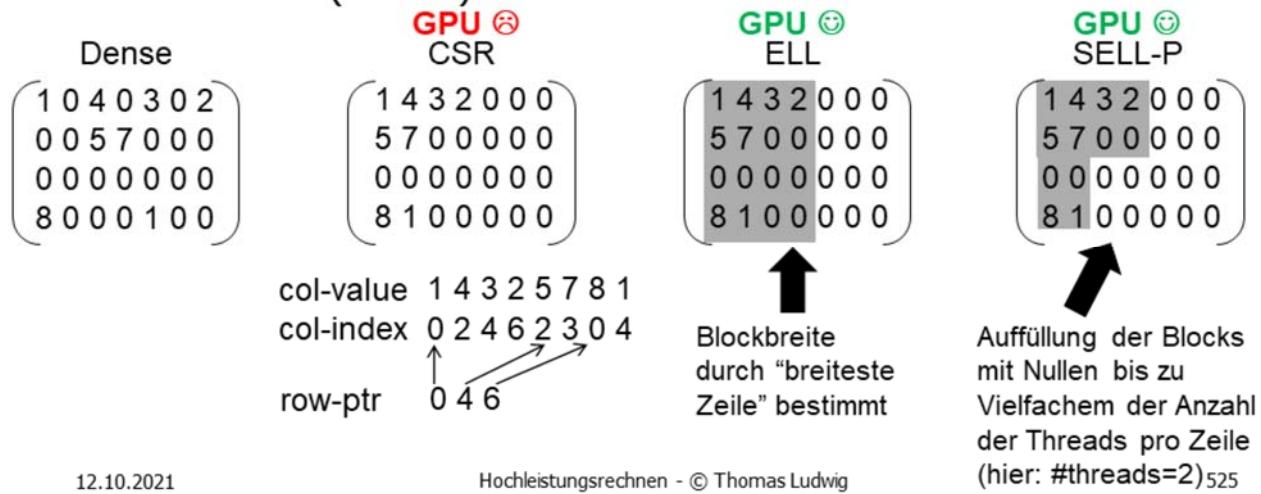
## SPMV: Sparse Matrix Vector Multiplication

Details zur Grafik und zu Performance-Vergleichen:

Anzt, H., Tomov, S., and Dongarra, J. Implementing a Sparse Matrix Vector Product for the SELL-C/SELL-C- $\sigma$  formats on NVIDIA GPUs. Tech. Rep. ut-eecs-14-727, University of Tennessee, March 2014

## Exkurs: Dünnbesetzte Matrizen/SPMV (1)

- SPMV=Sparse Matrix Vector Multiplication
- Datenstrukturen (bspw. MAGMA):
  - Compressed Storage Row (CSR), ELLPACK (ELL), padded sliced ELLPACK (SELL-P)



Details zur Grafik und zu Performance-Vergleichen:

Anzt, H., Tomov, S., and Dongarra, J. Implementing a Sparse Matrix Vector Product for the SELL-C/SELL-C- $\sigma$  formats on NVIDIA GPUs. Tech. Rep. ut-eecs-14-727, University of Tennessee, March 2014

## Exkurs: Dünnbesetzte Matrizen/SPMV (2)

- Shared-Memory Parallelisierung
  - Zeilenweise Verteilung und Thread-Scheduling
    - Pinning von #Zeilen pro Thread
    - Anfällig für Last Imbalanz
    - Dynamische Verteilung der Zeilen an Threads
  - Segmentverteilung (basierend auf Nicht-Null-Einträgen)
    - CSR: Pinning der drei Arrays auf einen Memory-Controller ☺
      - Partitioniertes Speichern der Arrays
    - CSR: Konträr zu SIMD-Nutzung
      - Alternative: Submatrizen/-zeilen verteilen
  - ELL/SELL-P: Effizientes Handling von Matrix-Blöcken
    - erhöhte Last durch Nulleinträge

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

526

Weiterführende Literatur:

S. Williams, N. Bell, J.W. Choi, M. Garland, L. Oliker, R. Vuduc. Chapter 5: Sparse Matrix-Vector Multiplication on Multicore and Accelerators. In J. Dongarra, D.A. Bader, J. Kurzak (eds) Scientific Computing With Multicore and Accelerators, Taylor Francis, 2010

## Exkurs: Dünnbesetzte Matrizen/SPMV (3)

- Distributed-Memory Parallelisierung
  - (Gleichmäßige) Verteilung der Input/Output-Vektoren auf Prozesse
  - Verteilung von Zeilenblöcken auf Prozesse
    - Aufteilung der Zeilenblöcke: On-Process vs. Off-Process Block/Vektor  
On-Process Block x On-Process Vektor → On-Process Vektor  
Off-Process Block x Off-Process Vektor → On-Process Vektor
  - Kommunikation von Off-Process Vektoreinträgen zwischen Prozessen  
→ Datenlokalität? Kommunikationsreduktion?



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

527

Weiterführende Literatur:

A. Bienz, W.D. Gropp, L.N. Olson. Node Aware Sparse Matrix-Vector Multiplication. arXiv:1612.08060, 2016. URL:  
<https://arxiv.org/abs/1612.08060>

## NumPY

- Implementierung basierend auf C und Python für Verwendung in Python
- Features:
  - N-Dimensionale Arrays
  - Tools zur Integration von C++- und Fortran-Code
  - Funktionen für lineare Algebra, FFT, Zufallszahlen
  - Universal Functions
  - Funktionen für Broadcasting
- Weitere Informationen: <http://www.numpy.org/>

## NumPY: Universal Functions

```
import numpy as np
import timeit
import sys

def recip(v):
 w= np.empty(len(v))
 for i in range(len(v)):
 w[i] = 1.0/v[i]
 return w

if __name__ == "__main__":
 np.random.seed(42)
 v = np.random.randint(1,1000, \
 size=1000000)
 if (sys.argv[1]=="recip"):
 recip(v)
 elif (sys.argv[1]=="norecip"):
 1.0/v
```

```
time python ./ufunc.py recip
real 0m2.387s
```

```
...
time python ./ufunc.py norecip
real 0m0.367s
...
```

- Python-Programm langsam  
(dynamisches Type-Checking,  
Interpreter etc.) ☹
- Ufuncs: „Vektorisierte“  
Operationen auf ndarray-  
Elementen

## NumPY: Broadcasting

```
import numpy as np

np.random.seed(42)
v=np.random.rand(10)
avg=v.mean()
print(v-avg)
```

- Spezifische Behandlung von Arrays unterschiedlicher Größe
- Simpel und effizient für schnelles Skripting
- Implementierung je nach Kontext fehleranfällig
- Beispiel-Code: Verschiebung von Zufallszahlen-Verteilung auf Null-Mittelwert



## **3. Löserpakete für Wissenschaftliche Anwendungen/PDEs**

- **PETSc**
- **Trilinos**
- **SciPy**

## PETSc/Tao (1)

- PETSC=Portable, Extensible Toolkit for Scientific Computation
- Unterstützt MPI, GPUs durch CUDA/OpenCL, MPI-GPU hybrid
- Funktionalitäten
  - Vektor- und Matrix-Operationen
  - Daten- und Gittermanagement (Strukturierte und unstrukturierte Gitter, Graphen und Netzwerke, ...)
  - Lineare Löser (Vorkonditionierer, Krylov-Methoden, Geometrisches Mehrgitterverfahren)
  - Nichtlineare Löser
  - Zeitschrittverfahren für ODEs
  - Optimierungsverfahren (Tao optimization library)
- Weitere Informationen:

<https://www.mcs.anl.gov/petsc/index.html>

## PETSc/Tao (2)

- Software, die von PETSc (optional) genutzt wird:
  - BLAS, LAPACK
  - FFTW
  - ParMeTiS
  - ...
- Software, die PETSc nutzt:
  - MOOSE (Multiphysics Object-Oriented Simulation Environment)
  - SLEPc (Scalable Library for Eigenvalue Problems)
  - FEnICS (Finite-Elemente-Software zur Lösung von PDEs)
  - Firedrake (Finite-Elemente-Software zur Lösung von PDEs)
  - deal.II (Finite-Elemente-Software zur Lösung von PDEs)
  - PyClaw (Python-basierte Löser für hyperbolische PDEs)
  - ...

## PETSc/Tao: Parallelisierung (1)

- Excerpt from (11. Dez 2018 25. Nov 2019 09. Jan 2020):

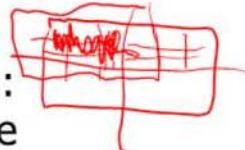
<https://www.mcs.anl.gov/petsc/miscellaneous/petscthreads.html>

"The core PETSc team has come to the consensus that **pure MPI using neighborhood collectives and the judicious using of MPI shared memory** (for data structures that you may not wish to have duplicated on each MPI process due to memory constraints) **will provide the best performance for HPC simulation needs on current generation systems, next generation systems and exascale systems.** It is also a much simpler programming model than MPI + threads (leading to simpler code)."

- Shared-Memory Parallelisierung möglich in Vec/Mat-Klassen oder Management einzelner PETSc-Objekte pro Thread, etc.

## PETSc/Tao: Parallelisierung (2)

- Local-to-Global Mapping: Differenzierung zwischen lokaler und globaler Nummierung von Vektoreinträgen, Zellen, Knoten, ...  
→ Berücksichtigung von Ghost Points
- Data Management for Distributed Arrays (DMDA):  
Parallelisierungskonzept für konzeptionell reguläre Rechtecksgitter  
→ Beispiel: siehe nächste Folien
- Unterstützung von Gather- und Scatter-Operationen, bspw. via PETSc-spezifische VecScatter\*-Befehle



## PETSc: Beispiel Poisson-Löser (1)

- Excerpt aus NS-EOF (Inkompressible Strömungssimulation für Lehrzwecke); Nutzung von PETSc, V. 3.3
  - Details: P. Neumann, C. Kowitz, F. Schranner, D. Azarnykh. J. Parallel Distrib. Comput. 105:83-91, 2017
- Algorithmus pro Zeitschritt
  - Berechne partielle Ableitungen aus Impulsgleichungen und Prognose für neue Strömungsgeschwindigkeiten u,v,w exklusive Druckanteil
  - **Berechne Druck p zum nächsten Zeitschritt aus Poisson-Gleichung (7-Punkt-Stencil im Diskreten):**
$$\partial^2 p / \partial x^2 + \partial^2 p / \partial y^2 + \partial^2 p / \partial z^2 = r(u, v, w)$$
rechte Seite der Gleichung enthält Geschwindigkeitsprognosen
  - Korrigiere Geschwindigkeitsfeld mit Hilfe neuer Druckwerte

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

536

## PETSc: Beispiel Poisson-Löser (1)

- Excerpt aus NS-EOF (Inkompressible Strömungssimulation für Lehrzwecke); Nutzung von PETSc, V. 3.3
  - Details: P. Neumann, C. Kowitz, F. Schranner, D. Azarnykh. J. Parallel Distrib. Comput. 105:83-91, 2017
- Algorithmus pro Zeitschritt
  - Berechne partielle Ableitungen aus Impulsgleichungen und Prognose für neue Strömungsgeschwindigkeiten u,v,w exklusive Druckanteil
  - **Berechne Druck p zum nächsten Zeitschritt aus Poisson-Gleichung (7-Punkt-Stencil im Diskreten):**
$$p_{i-1jk} + p_{ij-1k} + p_{ijk-1} - 6p_{ijk} + p_{i+1jk} + p_{ij+1k} + p_{ijk+1} = r_{ijk}$$
rechte Seite der Gleichung enthält Geschwindigkeitsprognosen
  - Korrigiere Geschwindigkeitsfeld mit Hilfe neuer Druckwerte

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

537

## PETSc: Beispiel Initialisierung (1)

```
KSPCreate(PETSC_COMM_WORLD,&_ksp);
PCCreate(PETSC_COMM_WORLD,&_pc);
...
PetscErrorCode (*computeMatrix)(KSP, Mat, Mat, void*) = NULL;
computeMatrix = computeMatrix3D;
DMCreate3d(
 PETSC_COMM_WORLD, bx, by, bz, DMDA_STENCIL_STAR,
 parameters.geometry.sizeX+2, parameters.geometry.sizeY+2,
 parameters.geometry.sizeZ+2,
 parameters.parallel.numProcessors[0],
 parameters.parallel.numProcessors[1],
 parameters.parallel.numProcessors[2],
 1, 2,
 parameters.parallel.sizes[0], parameters.parallel.sizes[1],
 parameters.parallel.sizes[2],
 &_da);
```

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

538

## PETSc: Beispiel Initialisierung (1)

```
KSPCreate(PETSC_COMM_WORLD, &_ksp);
PCCreate(PETSC_COMM_WORLD, &_pc);
...
PetscErrorCode (*computeMatrix)(KSP, PC, Mat, void*) = NULL;
computeMatrix = computeMatrix3D;
DMDACreate3d(
 PETSC_COMM_WORLD, bx, by, b,
 parameters.geometry.sizeX+2, parameters.geometry.sizeY+2,
 parameters.geometry.sizeZ+2,
 parameters.parallel.numProcessors[0],
 parameters.parallel.numProcessors[1],
 parameters.parallel.numProcessors[2],
 1, 2,
 parameters.parallel.sizes[0], parameters.parallel.sizes[1],
 parameters.parallel.sizes[2],
 &_da);
```

Initialisierung von Krylov  
Subspace Solver Context  
und Vorkonditionierer

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

539

## PETSc: Beispiel Initialisierung (1)

```
KSPCreate(PETSC_COMM_WORLD, &_ksp);
PCCreate(PETSC_COMM_WORLD, &_pc);
...
PetscErrorCode (*computeMatrix)(KSP, Mat, Mat, void*) = NULL;
computeMatrix = computeMatrix3D;
DMDACreate3d(
 PETSC_COMM_WORLD, bx, by, bz, MDA_STENCIL_STAR,
 parameters.geometry.sizeX+2, parameters.geometry.sizeY+2,
 parameters.geometry.sizeZ+2,
 parameters.parallel.numProcessors[0], parameters.parallel.numProcessors[1],
 parameters.parallel.numProcessors[2], 1, 2,
 parameters.parallel.sizes[0], parameters.parallel.sizes[2],
 &_da);
```

Setze Funktionspointer,  
um PETSc's  
Matrixinitialisierung später  
durchführen zu können

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

540

## PETSc: Beispiel Initialisierung (1)

```
KSPCreate(PETSC_COMM_WORLD)
PCCreate(PETSC_COMM_WORLD, ...
...
PetscErrorCode (*computeMatrix)(...)
computeMatrix = computeMatrix_C,
DMDACreate3d(
 PETSC_COMM_WORLD, bx, by, bz, DMDA_STENCIL_STAR,
 parameters.geometry.sizeX+2, parameters.geometry.sizeY+2,
 parameters.geometry.sizeZ+2,
 parameters.parallel.numProcessors[0],
 parameters.parallel.numProcessors[1],
 parameters.parallel.numProcessors[2],
 1, 2,
 parameters.parallel.sizes[0], parameters.parallel.sizes[1],
 parameters.parallel.sizes[2],
 &_da);
```

Initialisierung der  
eigentlichen PETSc-  
Datenstruktur

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

541

## PETSc: Beispiel Initialisierung (1)

```
KSPCreate(PETSC_COMM_WORLD,&_ksp);
PCCreate(PETSC_COMM_WORLD,&_pc);
...
PetscErrorCode (*computeMatrix)(KS
computeMatrix = computeMatrix3D;
DMDACreate3d(
 PETSC_COMM_WORLD, bx, by, bz, DMDA_STENCIL_STAR,
 parameters.geometry.sizeX+2, parameters.geometry.sizeY+2,
 parameters.geometry.sizeZ+2,
 parameters.parallel.numProcessors[0],
 parameters.parallel.numProcessors[1],
 parameters.parallel.numProcessors[2],
 1, 2,
 parameters.parallel.sizes[0], parameters.parallel.sizes[1],
 parameters.parallel.sizes[2],
 &_da);
```

Parameter zur  
Randbehandlung

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

542

## PETSc: Beispiel Initialisierung (1)

```
KSPCreate(PETSC_COMM_WORLD, &_ksp);
PCCreate(PETSC_COMM_WORLD, &_pc);
...
PetscErrorCode (*computeMatrix)(KSP, Mat, Mat, void*) = NULL;
computeMatrix = computeMatrix3D;
DMCreate3d(
 PETSC_COMM_WORLD, bx, by, bz, DMDA_STENCIL_STAR,
 parameters.geometry.sizeX+2, parameters.geometry.sizeY+2,
 parameters.geometry.sizeZ+2,
 parameters.parallel.numProcessors[0],
 parameters.parallel.numProcessors[1],
 parameters.parallel.numProcessors[2],
 1, 2,
 parameters.parallel.sizes[0], parameters.parallel.sizes[1],
 parameters.parallel.sizes[2],
 &_da);
```

Stencilformat für  
Matrixzeilen-Beschreibung;  
legt Ghost Points und  
Nachrichtentransfer fest:  
**STAR**: weniger Nachrichten  
**BOX**: mehr Nachrichten

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

543

## PETSc: Beispiel Initialisierung (1)

```
KSPCreate(PETSC_COMM_WORLD,&_ksp);
PCCreate(PETSC_COMM_WORLD,&_pc);
...
PetscErrorCode (*computeMatrix)(KS
computeMatrix = computeMatrix3D;
DMDACreate3d(
 PETSC_COMM_WORLD, bx, by, bz, MDA_STENCIL_STAR,
parameters.geometry.sizeX+2, parameters.geometry.sizeY+2,
parameters.geometry.sizeZ+2,
parameters.parallel.numProcessors[0],
parameters.parallel.numProcessors[1],
parameters.parallel.numProcessors[2],
1, 2,
parameters.parallel.sizes[0], parameters.parallel.sizes[1],
parameters.parallel.sizes[2],
&_da);
```

Globale Größe des 3D-  
Systems

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

544

## PETSc: Beispiel Initialisierung (1)

```
KSPCreate(PETSC_COMM_WORLD,&_ksp);
PCCreate(PETSC_COMM_WORLD,&_pc);
...
PetscErrorCode (*computeMatrix)(KS
computeMatrix = computeMatrix3D;
DMCreate3d(
 PETSC_COMM_WORLD, bx, by, bz, J,
 A_STENCIL_STAR,
 parameters.geometry.sizeX+2, parameters.geometry.sizeY+2,
 parameters.geometry.sizeZ+2,
parameters.parallel.numProcessors[0],
parameters.parallel.numProcessors[1],
parameters.parallel.numProcessors[2],
 1, 2,
 parameters.parallel.sizes[0], parameters.parallel.sizes[1],
 parameters.parallel.sizes[2],
 &_da);
```

Anzahl der MPI-Prozesse  
pro Dimension

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

545

## PETSc: Beispiel Initialisierung (1)

```
KSPCreate(PETSC_COMM_WORLD,&_ksp);
PCCreate(PETSC_COMM_WORLD,&_pc);
...
PetscErrorCode (*computeMatrix)(KS
computeMatrix = computeMatrix3D;
DMDACreate3d(
 PETSC_COMM_WORLD, bx, by, DMDA_STENCIL_STAR,
parameters.geometry.sizeX+2, parameters.geometry.sizeY+2,
parameters.geometry.sizeZ+2,
parameters.parallel.numProcessors[0],
parameters.parallel.numProcessors[1],
parameters.parallel.numProcessors[2],
1, 2,
parameters.parallel.sizes[0], parameters.parallel.sizes[1],
parameters.parallel.sizes[2],
&_da);
```

Anzahl der Variablen pro  
Zelle und Stencilbreite

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

546

## PETSc: Beispiel Initialisierung (1)

```
KSPCreate(PETSC_COMM_WORLD,&_ksp);
PCCreate(PETSC_COMM_WORLD,&_pc);
...
PetscErrorCode (*computeMatrix)(KS
 computeMatrix = computeMatrix3D;
 DMDACreate3d(
 PETSC_COMM_WORLD, bx, by, MDA_STENCIL_STAR,
 parameters.geometry.sizeX+2, parameters.geometry.sizeY+2,
 parameters.geometry.sizeZ+2,
 parameters.parallel.numProcessors[0],
 parameters.parallel.numProcessors[1],
 parameters.parallel.numProcessors[2],
 1, 2,
parameters.parallel.sizes[0], parameters.parallel.sizes[1],
parameters.parallel.sizes[2],
&_da);
```

Arrays mit Anzahl der  
Zellen

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

547

## PETSc: Beispiel Initialisierung (1)

```
KSPCreate(PETSC_COMM_WORLD,&_ksp);
PCCreate(PETSC_COMM_WORLD,&_pc);
...
PetscErrorCode (*computeMatrix)(KS
computeMatrix = computeMatrix3D;
DMDACreate3d(
 PETSC_COMM_WORLD, bx, by, bz,
 parameters.geometry.sizeX+2, parameters.geometry.sizeY+2,
 parameters.geometry.sizeZ+2,
 parameters.parallel.numProcessors[0],
 parameters.parallel.numProcessors[1],
 parameters.parallel.numProcessors[2],
 1, 2,
 parameters.parallel.sizes[0], parameters.parallel.sizes[1],
 parameters.parallel.sizes[2],
 &_da);
```

Distributed-Array Objekt,  
welches durch diese  
Methode initialisiert wird

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

548

## PETSc: Beispiel Initialisierung (2)

```
...
PCSetType(_pc,PCILU);
PCFactorSetLevels(_pc,1);
KSPSetPC(_ksp,_pc);
...
```

Definition des Vorkonditionierers

```
KSPSetFromOptions(_ksp);
KSPSetInitialGuessNonzero(_ksp,PETSC_TRUE);
KSPSetUp(_ksp);
...
```

Konfiguration weiterer Optionen über Kommandozeile

Start von Nicht-Null-Lösung für Druckfeld

Anlegen der Löserinstanz

## PETSc: Beispiel computeMatrix3D(...) (1)

```
for (k = limitsZ[0]; k < limitsZ[1]; k++) {
 for (j = limitsY[0]; j < limitsY[1]; j++) {
 for (i = limitsX[0]; i < limitsX[1]; i++) {
 ...
 stencilValues[1] = 2.0/(dx_L *(dx_L+dx_R)); // left
 stencilValues[0] = 2.0/(dx_R *(dx_L+dx_R)); // right
 stencilValues[2] = 2.0/(dx_T *(dx_T+dx_Bo)); // top
 stencilValues[3] = 2.0/(dx_Bo*(dx_T+dx_Bo)); // bottom
 stencilValues[4] = 2.0/(dx_B *(dx_B+dx_F)); // back
 stencilValues[5] = 2.0/(dx_F *(dx_B+dx_F)); // front
 stencilValues[6] = -2.0/(dx_R*dx_L)-2.0/(dx_T*dx_Bo)-
 2.0/(dx_F*dx_B); // center
 }
 }
}
```

Schleife über alle Zellen  
und Initialisierung der  
Matrix-Zeile

## PETSc: Beispiel computeMatrix3D(...) (2)

```
// Definition of positions. Order must correspond to values
column[0].i = i+1; column[0].j = j; column[0].k = k;
column[1].i = i-1; column[1].j = j; column[1].k = k;
column[2].i = i; column[2].j = j+1; column[2].k = k;
column[3].i = i; column[3].j = j-1; column[3].k = k;
column[4].i = i; column[4].j = j; column[4].k = k+1;
column[5].i = i; column[5].j = j; column[5].k = k-1;
column[6].i = i; column[6].j = j; column[6].k = k;
```

```
MatSetValuesStencil(A, 1, &row, 7, column, stencilValues,
INSERT_VALUES);
```

... Alle Sonderfälle:  
Randbehandlung  
etc.

Definition der Indexeinträge in  
Zeile und Setzen aller  
Informationen in PETSc

## PETSc: Beispiel computeMatrix3D(...) (3)

```
MatAssemblyBegin(A,MAT_FINAL_ASSEMBLY);
MatAssemblyEnd(A,MAT_FINAL_ASSEMBLY);
```

Assemblierung der  
globalen Matrix-  
Information

```
MatNullSpace nullspace;
MatNullSpaceCreate(PETSC_COMM_WORLD,PETSC_TRUE,
0,0,&nullspace);
MatSetNullSpace(A,nullspace);
MatNullSpaceDestroy(&nullspace);
```

Definition des Kerns  
(=NullSpace) der Matrix:  
Konstante Druckfelder

## PETSc: Beispiel solve()

```
KSPSetComputeRHS(_ksp, computeRHS3D, & _ctx);
KSPSetComputeOperators(_ksp, computeMatrix3D, & _ctx);
KSPSolve(_ksp, PETSC_NULL, _x);
```

Definition von rechter Seite und Matrix

```
// Then extract the information
PetscScalar ***array;
DMDAVecGetArray(_da, _x, &array);
```

Endlich wird gerechnet 😊

```
Zurückschreiben der Lösung in Simulations-Datenstruktur
```

```
for (int k = _firstZ; k < _firstZ + lengthZ; k++){
 for (int j = _firstY; j < _firstY + lengthY; j++){
 for (int i = _firstX; i < _firstX + lengthX; i++){
 pressure.getScalar(i - _firstX + _offsetX, j - _firstY + _offsetY,
 k - _firstZ + _offsetZ) = array[k][j][i];
 }
 }
}
```

...

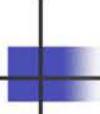
12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

553

# Trilinos

- Pakete-Sammlung zur Lösung von „large-scale, complex multi-physics engineering and scientific problems“
- Weitere Informationen: <https://trilinos.org/>
- Einteilung der Pakete in „Capability Areas“:
  - User Experience
  - Parallel Programming Environments
  - Framework & Tools
  - Software Engineering Technologies and Integration
  - I/O Support
  - Meshes, Geometry & Load Balancing
  - Discretization
  - Scalable Linear Algebra (Pakete Eptra, Tpetra, etc.; Paket Teuchos wrappt BLAS/LAPACK)
  - Linear & Eigen Solvers
  - Embedded Nonlinear Analysis Tools



## SciPy

- Nutzt NumPy
- Funktionalitäten
  - Numerische Integration (Quadratur)
  - Optimierung
  - Interpolation
  - FFT
  - Symbolisches Rechnen
  - Visualisierung und Datenanalyse (Pandas)
  - ...
- Weitere Informationen: <https://www.scipy.org/>

## 4. Software-Frameworks

- Finite Elemente Frameworks
  - **FEniCS**
  - **deal.II**
- Andere Frameworks
  - DUNE
    - DUNE=Distributed and Unified Numerics Environment
    - Basiert auf generischer Programmierung (Expression Templates)
    - Kernmodule:
      - dune-common (allg. Infrastruktur)
      - dune-geometry
      - dune-grid
      - dune-istl (iterative solver template library)
      - dune-local functions (Ansatzfunktionen)
  - Numerische Strömungsmechanik: OpenFOAM
  - Wettersimulation: OpenIFS

# Finite Elemente Methode



$$-\frac{d^2u}{dx^2} = f(x), \quad u(0) = u(1) = 0$$

Testfunktionen

$$-\int_0^1 \frac{d^2u}{dx^2} \cdot v \, dx = \int_0^1 f \cdot v \, dx \quad \forall v \in V$$

Schwache  
Formulierung

$$\int_0^1 \frac{du}{dx} \cdot \frac{dv}{dx} \, dx = \int_0^1 f \cdot v \, dx \quad \forall v \in V$$

Ansatzfunktionen

$$u(x) := \sum_i \alpha_i w_i(x)$$

$$\sum_i \alpha_i \int_0^1 \frac{dw_i}{dx} \cdot \frac{dv}{dx} \, dx = \int_0^1 f \cdot v \, dx \quad \forall v \in V$$

# FEnICS

- Ziel: „creating easy, intuitive, efficient, and flexible software for solving partial differential equations (PDEs) using finite element methods“
  - H.P. Langtangen, A. Logg. Solving PDEs in Python. The FEniCS Tutorial I, Springer, 2016
  - Weitere Informationen: <https://fenicsproject.org/>
- C++-Backend (DOLFIN)
- C++- und Python-Schnittstellen
- Schnittstellen zu (u.a.) PETSc, Trilinos, ParMETIS
- Features:
  - Automatische Generierung von Basisfunktionen
  - Automatische Evaluierung von schwachen Formulierungen
  - Automatische Finite-Element-Assemblierung
  - Automatische adaptive Fehlerkontrolle
  - Siehe: [https://fenicsproject.org/pub/course/lectures/2017-nordic-phdcourse/lecture\\_01\\_fenics\\_introduction.pdf](https://fenicsproject.org/pub/course/lectures/2017-nordic-phdcourse/lecture_01_fenics_introduction.pdf)
- FEniCS-HPC: Hybrid MPI/OpenMP-parallelisierte FEniCS-Komponenten

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

559

Literatur zu FEniCS-HPC:

J. Hoffman, J. Jansson, N. Jansson. FEniCS-HPC: Automated Predictive High-Performance Finite Element Computing with Applications in Aerodynamics. In Wyrzykowski R., Deelman E., Dongarra J., Karczewski K., Kitowski J., Wiatr K. (eds) Parallel Processing and Applied Mathematics. PPAM 2015. Lecture Notes in Computer Science, vol 9573, Springer, 2016

# FEniCS

- Ziel: „creating easy, intuitive, efficient, and flexible software for solving partial differential equations (PDEs) using finite element methods“
  - H.P. Langtangen, A. Logg. Solving PDEs in Python. The FEniCS Tutorial I, Springer, 2016
  - Weitere Informationen: <https://fenicsproject.org/>
- C++-Backend (DOLFIN)
- C++- und Python-Schnittstellen
- Schnittstellen zu (u.a.) PETSc, Trilinos, ParMETIS
- Features:
  - Automatische Generierung von Meshes
  - Automatische Evaluierung von Integralen
  - Automatische Finite-Element-Meshing
  - Automatische adaptivem Meshing
  - Siehe: [https://fenicsproject.org/phdcourse/lecture\\_01\\_fenics\\_intro.pdf](https://fenicsproject.org/phdcourse/lecture_01_fenics_intro.pdf)
- FEniCS-HPC: Hybrid MPI/OpenMP-parallelisierte FEniCS-Komponenten

u.a. Turbulente  
Strömungssimulation auf 5000  
Rechenkernen mit PETSc-Backend

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

560

Literatur zu FEniCS-HPC:

J. Hoffman, J. Jansson, N. Jansson. FEniCS-HPC: Automated Predictive High-Performance Finite Element Computing with Applications in Aerodynamics. In Wyrzykowski R., Deelman E., Dongarra J., Karczewski K., Kitowski J., Wiatr K. (eds) Parallel Processing and Applied Mathematics. PPAM 2015. Lecture Notes in Computer Science, vol 9573, Springer, 2016

## FEniCS: Beispiel Poisson-Gleichung (1)

- Weitere Informationen:

[https://fenicsproject.org/olddocs/dolfin/1.3.0/python/demo\\_documented/poisson/python/documentation.html](https://fenicsproject.org/olddocs/dolfin/1.3.0/python/demo_documented/poisson/python/documentation.html)

- Transformation in schwache Formulierung:

$$\begin{aligned} -\nabla^2 u &= f \quad \text{in } \Omega, \\ u &= 0 \quad \text{on } \Gamma_D, \\ \nabla u \cdot n &= g \quad \text{on } \Gamma_N. \end{aligned} \quad \Rightarrow \quad \begin{aligned} a(u, v) &= L(v) \quad \forall v \in V, \\ a(u, v) &= \int_{\Omega} \nabla u \cdot \nabla v \, dx, \\ L(v) &= \int_{\Omega} fv \, dx + \int_{\Gamma_N} gv \, ds. \end{aligned}$$

## FEniCS: Beispiel Poisson-Gleichung (2)

```
https://fenicsproject.org/olddocs/dolfin/1.3.0/python/
_downloads/demo_poisson.py
...
mesh = UnitSquareMesh(32, 32) Geometrie: Einheits-
V = FunctionSpace(mesh, "Lagrange", 1) quadrat, 32x32 Elemente
 Lagrange-Elemente
 erster Ordnung
Define Dirichlet boundary (x = 0 or x = 1)
def boundary(x):
 return x[0] < DOLFIN_EPS or x[0] > 1.0 -
DOLFIN_EPS

u0 = Constant(0.0)
bc = DirichletBC(V, u0, boundary)
```

## FEniCS: Beispiel Poisson-Gleichung (3)

```
Define variational problem
u = TrialFunction(V)
v = TestFunction(V)
f = Expression("10*exp(-(pow(x[0] - 0.5, 2) + pow(x[1] - 0.5, 2)) / 0.02)")
g = Expression("sin(5*x[0])")
a = inner(grad(u), grad(v))*dx
L = f*v*dx + g*v*ds

Compute solution
u = Function(V)
solve(a == L, u, bc)
```

Ansatzfunktionen

Testfunktionen

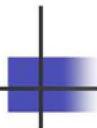
Löse:

$$a(u, v) = L(v) \quad \forall v \in V,$$
$$a(u, v) = \int_{\Omega} \nabla u \cdot \nabla v \, dx,$$
$$L(v) = \int_{\Omega} fv \, dx + \int_{\Gamma_N} gv \, ds.$$



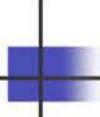
## deal.II

- deal.II=successor of Differential Equations Analysis Library
- C++-Bibliothek
- Features:
  - Lokal verfeinerte Gitter, adaptive Verfeinerung mit Fehlerschätzern
  - h-,p-,hp-Verfeinerung
  - Continuous vs. Discontinuous elements
  - Eigenständige Bibliothek für lineare Algebra
  - Schnittstellen zu Trilinos, PETSc, METIS
- Weitere Informationen: <https://www.dealii.org/>



## Wer ist Super-HPCler? ☺

- A Stampfen
- B Klatschen
- C „Hell, Yeah!“
- D Winken



## Wer ist Super-HPCler? ☺

- A Stampfen
- B Klatschen
- C „Hell, Yeah!“
- D Winken

Wie funktioniert das Speicherformat ELLPACK?

- A Komprimieren der Matrix in einen großen Block, Padding mit Nullen
- B Komprimieren der Matrix in verschiedenen große Blöcke, Padding mit Nullen
- C Indirekte Indexierung der Matrixeinträge
- D Das ist ein ELLbogen-SchutzPACK beim Skaten



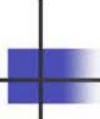
## Wer ist Super-HPCler? ☺

- A Stampfen
- C „Hell, Yeah!“

- B Klatschen
- D Winken

Was ist BLAS?

- A Eine Aufforderung zum Pusten
- C Ausgestoßene Atemluft von Walen
- B Ein Bib-Ansatz für LinAlg-Komponenten
- D Ein Löseralgorithmus für Gleichungssysteme

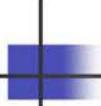


## Wer ist Super-HPCler? ☺

- A Stampfen
- B Klatschen
- C „Hell, Yeah!“
- D Winken

Welche der folgenden Abkürzungen  
spielt in PETSc eine Rolle?

- A DAAD
- B MDAD
- C DAMD
- D DMDA



## Mathematische Bibliotheken

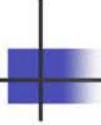
### Zusammenfassung

- Mathematische Bibliotheken für numerische lineare Algebra, zur Lösung von PDEs/ODEs, Datenanalyse, etc.
- Numerische lineare Algebra
  - BLAS für grundlegende Operationen
  - LAPACK/Scalapack für Gleichungssystemlöser, Eigenwertprobleme, etc.
  - Exkurs: Dünnbesetzte Matrizen versus HPC-Implementierung
- Löserpakete für PDEs (Beispiel: PETSc)
  - Nutzung von Low-Level Bibs (wie bspw. BLAS)
  - Breite Funktionalität: Zeitschrittverfahren, (nicht-)lineare Löser, Gebietszerlegungen, etc.
- High-Level Frameworks (Beispiel: FEniCS): Anwendungs- und problemnahe Formulierung und Implementierung

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

572



# Reproduzierbarkeit

1. Einführung und Überblick
2. Zahldarstellung
3. Einflussgrößen
4. Beispiel Klimamodell ICON

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

573

# 1. Einführung und Überblick

## "Open Science": Forscher brechen aus dem Elfenbeinturm aus

 heise online 18.02.2017 11:16 Uhr

 vorlesen

Die Digitalisierung verändert das Selbstverständnis von Wissenschaftlern. Sie öffnen ihre Daten und Ergebnisse dem Zugang für andere Forscher und für die Öffentlichkeit. Wichtige Impulse dafür gehen von der Stadt Gutenberg aus.

Dass "Open Science" gerade in der Psychologie sehr forciert wird, hat seinen Grund in der sogenannten Replikationskrise des Fachs: In einem internationalen Projekt wurden 100 psychologische Experimente wiederholt, deren Ergebnisse in renommierten Fachzeitschriften veröffentlicht wurden. Dabei konnten die Ergebnisse von mehr als jeder zweiten Studie nicht "repliziert", also nicht bestätigt werden.

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

574

Siehe: <https://www.heise.de/newsticker/meldung/Open-Science-Forscher-brechen-aus-dem-Elfenbeinturm-aus-3630380.html>



## Reproduzierbarkeit und Computersimulation

Verschiedene Abstraktionsgrade

- Reproduziere das wissenschaftliche Ergebnis
  - Komme zur selben Erkenntnis
- Reproduziere Computerexperimente
  - Code neu laufen lassen und Ergebnisse prüfen
  - "Gute wissenschaftliche Praxis"
- **Wiederhole ein Experiment mit bitweisem Endergebnis**
  - **Bei identischen Eingabedaten**

Alle Ebenen gleichzeitig relevant

# Reproduzierbarkeitswettbewerb (SC'16)

## SCC Reproducibility Initiative Winner

Replication and reproducibility of experimental computer science results in peer-reviewed paper is gaining relevance in the HPC community. SC, the leading conference in the field, wants to promote and support replication and reproducibility through a new initiative that aims to integrate aspects of past technical papers into the Student Cluster Competition (SCC). The SCC is excited to announce “A parallel connectivity algorithm for de Bruijn graphs in metagenomic applications” as the winning paper for the inaugural reproducibility initiative. This paper and accompanying application will be reproduced in the SCC at SC16. **This is the first time that students have been challenged to reproduce a paper rather than run prescribed data sets.** Although they are doing similar tasks from previous competitions, they are seeing it from an entirely new perspective, as a component to the scientific process. **“We want students to understand, early in their careers, the important role reproducibility plays in research.”** explains the SCC Chair Stephen Harrell

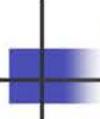
12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

576

### SC'16: SCC Reproducibility Initiative

Siehe: <http://sc16.supercomputing.org/studentssc/scc-reproducibility-initiative-winner/>



## Reproduzierbarkeit und Klimamodelle

- Klimamodellierung
  - Hochgradig nichtlineare Mathematik
  - Simulationen über lange Zeiträume
- Computer
  - Assoziativität nicht gewährleistet
  - Maschinen sind nicht deterministisch

## 2. Zahldarstellung

IEEE 745: 64-Bit-Zahldarstellung (double)

- 1 Bit für Vorzeichen +/-
- 11 Bits für Exponent
- 52 Bits für Mantisse
  - Ca. 16 Bits dezimal

Man kann große und kleine Werte mit hoher  
Genauigkeit speichern, ABER nicht ohne  
Genauigkeitsverlust zusammenführen (z.B. addieren)

## Assoziativität

Reihenfolge ist relevant!

- Assoziativität nicht in realen Computern  
 $a + ( b + c ) \neq ( a + b ) + c$

Beispiel

$$a=11.0000b \quad b=0.000011b \quad c=0.000001b$$

- Jeder Wert kann im Computer mit 6 Bits Mantisse repräsentiert werden
- Was ist mit  $a+b+c$  ?

## Assoziativität

$$\begin{array}{rcl} a & = 11,0000 & (a+b) = 11,0000 \text{ 11} \\ + (b+c) = 0,0001 \text{ 00} & & +c = 0,0000 \text{ 01} \\ \hline a+(b+c) = 11,0001 & & (a+b)+c = 11,0000 \end{array}$$

- Jede arithmetische Operation kann dieses Problem aufweisen
- Beeinflusst nur die hintersten Bits

### 3. Einflussgrößen

Jede Reihenfolgenänderung in der Arithmetik kann das Ergebnis nichtreproduzierbar ändern

- Region unterschiedlich partitioniert
- Anzahl Rechnerknoten geändert
- Nichtdeterminismus in Bibliotheken
- Nichtdeterminismus durch Lastbalancierung
- Compileroptionen geändert
- Prozessoren ausgetauscht
- Bibliotheken ausgetauscht
- Compiler ausgetauscht

*Wissenschaftler*

*Systemverwalter*

## 4. Beispiel Klimamodell ICON

ICON (icosahedral non-hydrostatic general circulation model)

- Anforderung
  - Muss bitweise reproduzierbares Ergebnis für alle möglichen Aufteilungen liefern (starke Anforderung)
- Tatsächlich: sequentielles und paralleles Ergebnis müssen identisch sein

Siehe: <https://code.mpimet.mpg.de/projects/iconpublic>

## Beispiel von Abweichungen

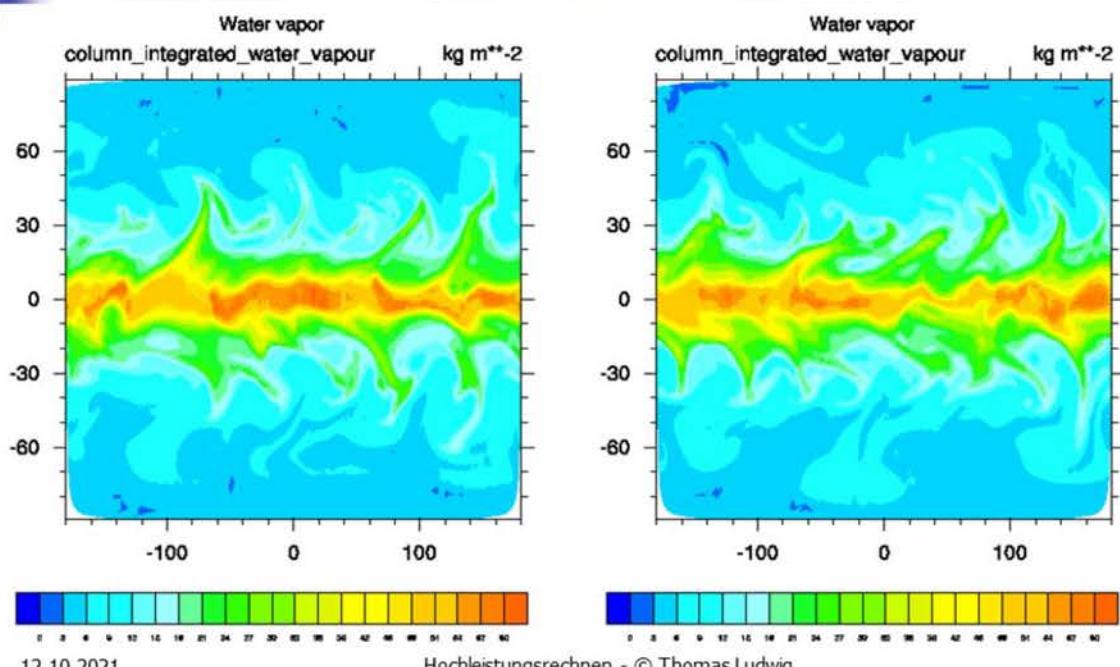
### Vorgehen

- Wissenschaftler nutzt Compileroptionen -O2 und -O3 mit Vektorisierung

Ergebnisse ändern sich bei

- -O2 novec und -O3 vec
- -O3 AVX von Lauf zu Lauf
  - AVX + FMA + dynamische Speicherverwaltung
  - = nichtdeterministisch

$t_{\text{sim}} = 100\text{d}$  O2: 10min30sec  $\rightarrow$  O3: 8m41sec  $\Rightarrow (+17,3\%)$

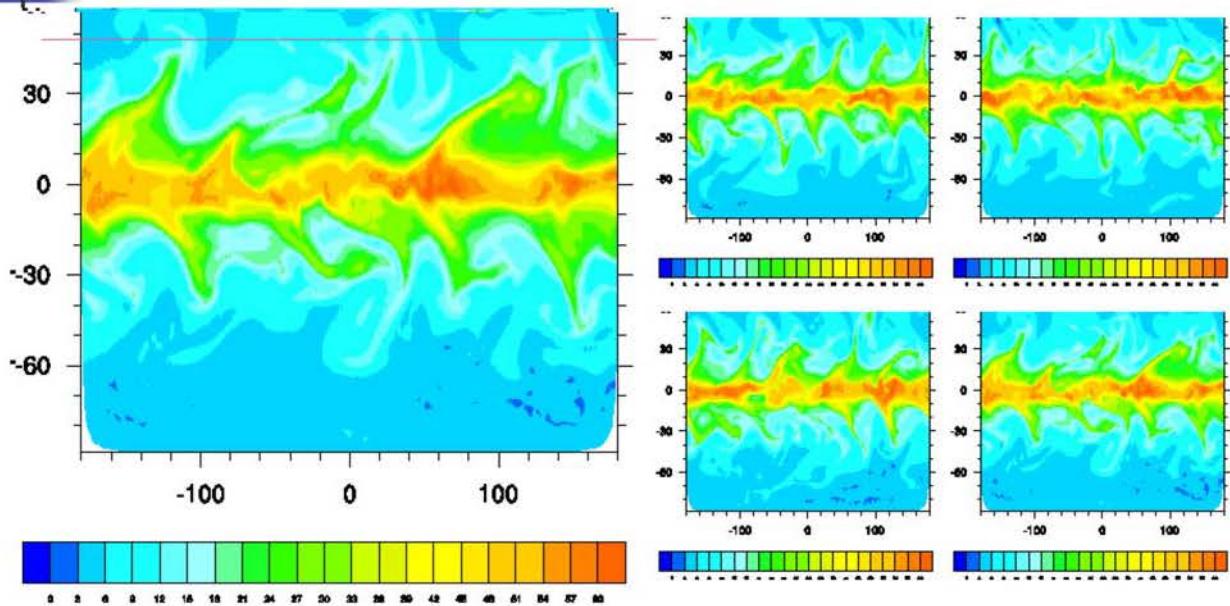


12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

584

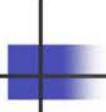
$t_{\text{sim}} = 100 \text{d}$  O3 -x AVX – 4 runs with identical input



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

585



## **Reproduzierbarkeit**

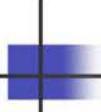
### Zusammenfassung

- Nichtreproduzierbarkeit von Einzelprogramm-ergebnissen entsteht durch fehlende Assoziativität bei der Zahldarstellung im Computer
- Nichtreproduzierbarkeit kann auf verschiedenen Abstraktionsebenen entstehen
- Nichtreproduzierbarkeit gezielt auszuschließen kostet viel Rechenleistung
- Nichtreproduzierbarkeit zuzulassen ist auch keine gute Variante
- Es bedarf hier weiterer Forschungen

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

586



## **Reproduzierbarkeit**

Die wichtigsten Fragen

- Unter welchen Bedingungen lässt sich ein Ergebnis einer Computersimulation nicht bitgenau reproduzieren?
- Was kann man dagegen tun?
- Wie sind Kosten und Nutzen in diesem Zusammenhang?

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

587

# Optimierung sequentieller Programme

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

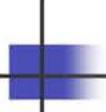
12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

588

Siehe:

- [https://de.wikipedia.org/wiki/Effizienz\\_%28Informatik%29](https://de.wikipedia.org/wiki/Effizienz_%28Informatik%29)
- <https://de.wikipedia.org/wiki/Komplexit%C3%A4tstheorie>
- <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

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

590

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 zur Beschleunigung

## 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 bis 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/algorithmische 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 (computergestütztes Experimentieren)
  - Wenig Produktionsbetrieb mit unverändertem Code
  - Wissenschaftler hat keine Zeit zur Codeoptimierung
  - Wissenschaftler hat keine Kenntnis über Möglichkeiten

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

598

Siehe aber z.B. hier:

- <https://insidehpc.com/2017/11/ultimate-trading-machine-penguin-computing-sets-record-low-latency/>

## (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 1-10% der **Rechenleistung** genutzt
- Gründe beispielhaft:
  - Sprünge im Code, nicht genügend Mathematik im Code
  - Indirektionen beim Speicherzugriff und schlechte Cache-Nutzung

## (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

## 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, davon die einfachen zuallererst

## 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<br>(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<br>(Vergleiche)<br>(Kopieraktionen) | $\mathcal{O}(n)$<br>$(n - 1)$<br>$(0)$ | $\mathcal{O}(n^2)$<br>$\approx \left(\frac{n^2}{4}\right)$<br>$\approx \left(\frac{n^2}{4}\right)$ | $\mathcal{O}(n^2)$<br>$\approx \left(\frac{n^2}{2}\right)$<br>$\approx \left(\frac{n^2}{2}\right)$ |
| 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?

## 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 Gesamlaufzeit des Programms
- Kommando **gprof**
  - Programm mit Compileroption für Profiling übersetzen (gcc: -pg)
    - Profiling: Leistungsprofil des Programms erstellen
  - Laufenlassen des Programms erzeugt Datei gmon.out
  - Kann mit gprof angesehen werden
- Kommando **perf**

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

609

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
- Erfolg einer Parallelisierung? ☺

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

610

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, wieviel 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)

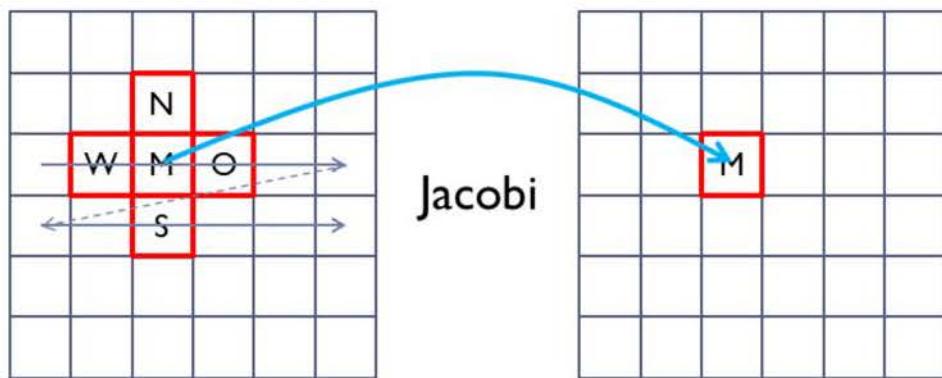
Beispiel: partielle Differentialgleichungen mittels Jacobi- oder Gauß/Seidel-Verfahren

- 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)



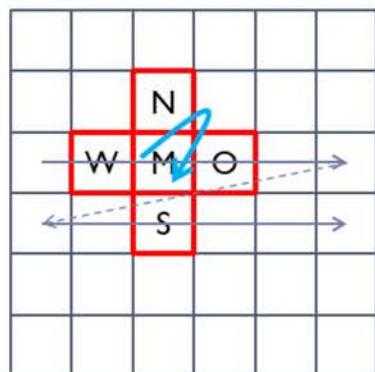
- Es werden zwei Matrizen verwendet: eine mit den aktuellen Werten und eine für die Werte der nächsten Iteration
- Der neue Wert M wird aus den alten Nachbarwerten von M (N, W, S und O) ermittelt
- Wenn alle neuen Werte bestimmt sind, werden die beiden Matrizen getauscht und die nächste Iteration beginnt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

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

614

# 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

# 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, Thalia u.a.: „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?

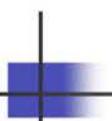
## To-Do-Liste ***Wichtig!***

### 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

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

628

## **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?



# Werkzeugarchitekturen

1. Werkzeuge allgemein
2. Grobstruktur von Werkzeugen
3. Beispiele für Werkzeuge

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

630

Leider keine Literatur. ☹

# 1. Werkzeuge allgemein

Was verstehen wir unter Werkzeugen?

Hier: Werkzeuge in den späteren Phasen des Lebenszyklus eines parallelen Programms

- Fehlersuche
- Optimierung
- Wartung
- Produktionsbetrieb

Wir betrachten nur Werkzeuge ab dem Zeitpunkt, zu dem ein halbwegs lauffähiges Programm vorliegt



# Was sind Werkzeuge?

## Begriffe

- Offline-Werkzeuge

Zeigen Informationen zum Programm nach dessen Ende oder zumindest deutlich verzögert an

- Online-Werkzeuge

Zeigen Informationen zum Programm gleichzeitig zum Ablauf an

Die beiden Klassen haben völlig unterschiedliche Implementierungskonzepte

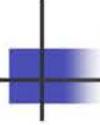
# Was sind Werkzeuge?

## Begriffe...

- Interaktive Werkzeuge
  - Gestatten eine direkte Interaktion mit dem Programm
  - beobachten/manipulieren
- Automatische Werkzeuge
  - Bearbeiten ohne Benutzereinwirkung das Programm zur Laufzeit
  - beobachten/manipulieren

Interaktiv und automatisch nur bei Online-Werkzeugen von Bedeutung

- Ausnahmen sind denkbar, z.B.: automatische Änderung der Parallelisierung aufgrund gemessenen Programmverhaltens

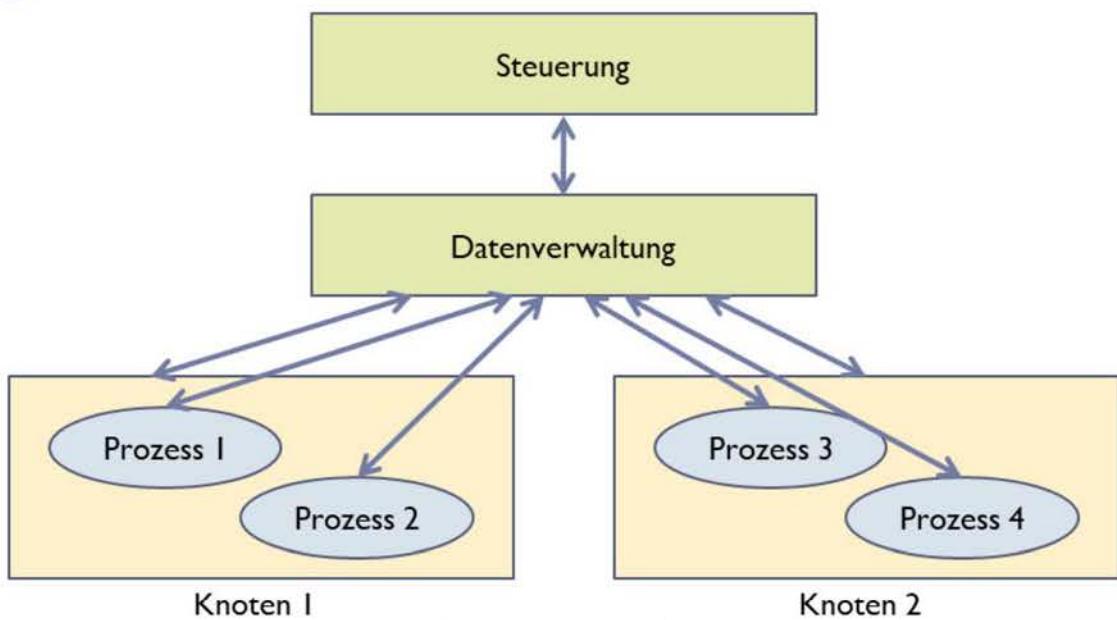


## Was sind Werkzeuge?

Hier nicht betrachtete Werkzeuge

- Werkzeuge zur Code-Erstellung
- Werkzeuge zur Parallelisierung
  - Interaktives Parallelisieren von Daten und Codebereichen
- Werkzeuge zur Gebietszerlegung der Daten
  - Gittergeneratoren bei numerischen Anwendungen

## 2. Struktur von Werkzeugen



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

635



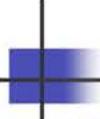
## Struktur von Werkzeugen...

Abstraktes Strukturbild klar

Aber: Realität sehr komplex

Gründe:

- Programm ist verteilt, also muss auch das Werkzeug verteilte Komponenten haben  
Werkzeug ist MPI-Programm?
- Die einzelnen Komponenten laufen auch auf Knoten des Rechnersystems  
Zusatzaufgabe, Ausfallsicherheit



## Struktur von Werkzeugen...

### Begriffe

- Monitor (knotenlokal)  
Komponenten auf einem Rechnerknoten, die HW und SW des Knotens überwachen können
- Monitorsystem  
Summe der Komponenten im System zur Beobachtung und Manipulation von HW und SW der Zielmaschine

# Struktur von Werkzeugen...

## Begriffe...

- Instrumentierung

Einbringen von zusätzlichem Code, der die Erfassung von Daten steuert und die Beeinflussung des Programms gestattet

- Sourcecode-Instrumentierung

Primitivste Variante: Einbau von `printf(...)`

- Bibliotheks-Instrumentierung

Z.B. mittels der PMPI-Schnittstelle in MPI

- Binärcode-Instrumentierung

Dynamisch eingebaute Sprünge in den Code des Monitorsystems



## Struktur von Werkzeugen...

### Offline-Werkzeuge

- Erfassung von Kenndaten zur Laufzeit des Programms
- Visualisierung nach Programmende
- Keine Beeinflussungsmöglichkeit des Programmlaufs
  - Aber nachträglich des Programms
- Überwachung wird aktiviert und Monitorsystem speichert Spurdaten ab (*trace*)
- Visualisierung der Spur nach Programmende



## Struktur von Werkzeugen...

### Online-Werkzeuge

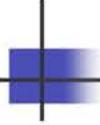
- Erfassung von Kenndaten zur Laufzeit
- Sofortige Visualisierung
- Sofortige Beeinflussung des Programmlaufs möglich
  
- Direkte Interaktion zwischen Steuerung und Datenverwaltung
- Optional Generierung von Spurdaten



## Struktur von Werkzeugen...

### Interaktive (Online-)Werkzeuge

- Benutzungsschnittstelle  
(graphisch oder Kommandozeile)
- Sofortige Darstellung  
(Skalierung der Darstellung schwierig)
- Steueranweisungen des Benutzers an Programm  
weiterleiten  
(Problem der zeitlichen Nähe)
- Nicht bei Stapelverarbeitung einsetzbar



## Struktur von Werkzeugen...

### Automatische (Online-)Werkzeuge

- Keine Benutzerinteraktion vorgesehen
- Steuerung bewertet Situation aufgrund von Heuristiken
- Steuerung manipuliert laufendes Programm und startet/stoppt einzelne Überwachungen

# Struktur von Werkzeugen...

## Allgemeine Probleme

- Skalierbarkeit der Datenerfassung  
Wie kann ich von den vielen Prozessoren die Daten effizient einsammeln?
- Konsistenz der Daten  
Sind sie alle zum selben Zeitpunkt entstanden?
- Skalierbarkeit der Darstellung  
Wie kann ich von den vielen Prozessen und Threads die Ergebnisdaten übersichtlich darstellen?
- Beeinflussungsfreiheit  
Das Programm soll nicht im Ablauf gestört werden
- Dynamik im Ablauf  
Variierende Knotenmenge / Prozessmenge / Threads-Menge

12.10.2021

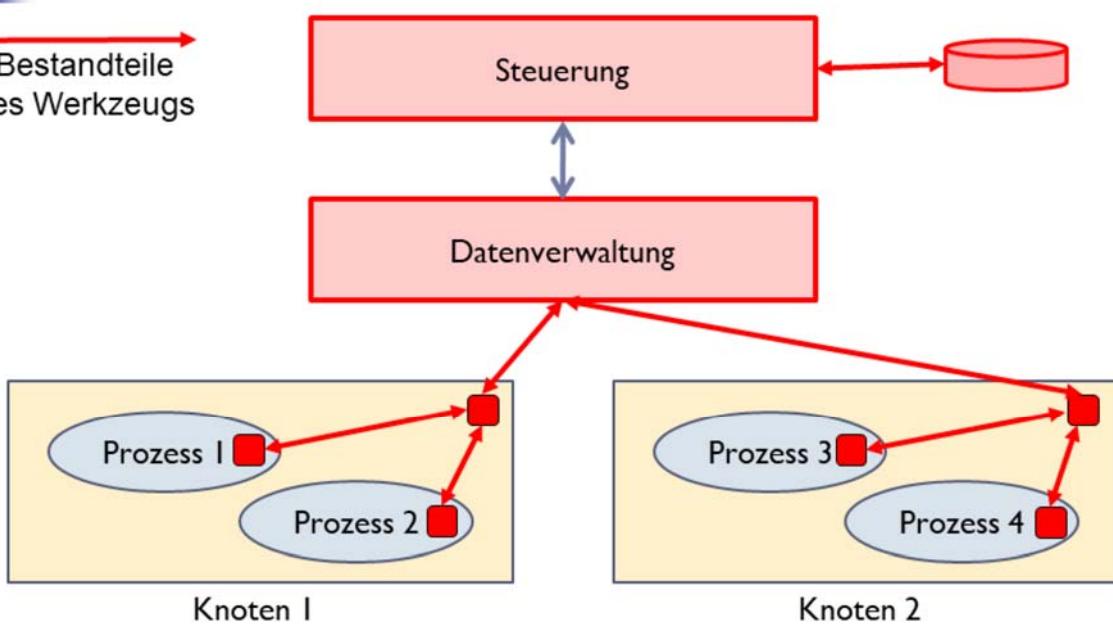
Hochleistungsrechnen - © Thomas Ludwig

643

Es gibt nur wenige Programme, bei denen die Anzahl der eingesetzten Knoten sich dynamisch ändert. Es gibt etwas mehr, aber immer noch wenige Programme, bei denen sich die Anzahl der Prozesse dynamisch ändert. Bei Einsatz von Threads ändert sich deren Anzahl naturgemäß laufend.

# Struktur von Werkzeugen...

Bestandteile  
des Werkzeugs



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

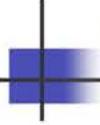
644

Wir finden Instrumentierungen der Prozesse, neue Komponenten auf jedem Rechnerknoten und zentrale Komponenten, die zusammen das Werkzeug konstituieren.

## 3. Beispiele für Werkzeuge

Die typischen interaktiven Werkzeuge:

- **Fehlersuche (*debugging*)**
- **Leistungsanalyse (*performance analysis*)**
- Programmalaufvisualisierung
- Ergebnisvisualisierung
- Ablaufsteuerung (*computational steering*)



## Beispiele für Werkzeuge...

### Die typischen automatischen Werkzeuge

- Ressourcenverwaltung
- Lastausgleich
- Sicherungspunktverwaltung
- Fehlertoleranzmechanismen

# Werkzeuge zur Fehlersuche

## Zweck

- Erkennen von Fehlerzuständen
- Auffinden von Fehlerursachen

## Probleme

- Anzahl der überwachten Prozesse/Threads
- Nichtdeterminismus im Ablauf
- Beeinflussungsfreie Beobachtung



# Werkzeuge zur Leistungsanalyse

## Zweck

- Visualisierung wichtiger Leistungsdaten
- Erkennen von Leistungsengpässen

## Probleme

- Erfassung der Daten produziert selber Last
- Zusatzlast minimieren oder herausrechnen
- Abweichende Abstraktionsebenen der Programmierung und der Messdatenerfassung



# Werkzeuge zur Ergebnisvisualisierung

## Zweck

- Visualisierung wichtiger Datenstrukturen  
Vor allem bei numerischen Anwendungen

## Probleme

- Falls online: Datenkonsistenz  
Z.B. alle Daten aus derselben Iterationsstufe des Programms
- Falls online: Datenmenge



# Werkzeuge zur Ablaufsteuerung

## Zweck

- Manipulation algorithmischer Kenngrößen zur Laufzeit  
Z.B. Algorithmus konvergiert schlecht; dann Korrektur einzelner Parameter
- Wichtig bei langlaufenden Programmen

## Probleme

- Wie bei Online-Ergebnisvisualisierung und Fehlersuche zusammen

## Status

- Hätte man gerne, gibt's praktisch nicht

# Werkzeuge zur Ressourcenverwaltung

## Zweck

- Zuteilung von parallelen Programmen zu Mengen von Knoten aufgrund definierter Strategien
- Verwaltung der Anfragen um Rechenzeit

## Probleme

- Erfassen der Lastverhältnisse im System
- Berechnen einer optimalen Zuteilung (NP-hard)

# Werkzeuge zum Lastausgleich

## Zweck

- Korrektur von Ungleichbelastungen beliebiger Ressourcen (meist CPU) zur Laufzeit
- Erzielt optimale Ressourcennutzung  
= meist minimale Programmlaufzeit

## Probleme

- Welche lasterzeugende Komponente soll verlagert werden
- Wann? Wie?

## Status

- Nahezu keine automatischen Werkzeuge, stattdessen Integration in den Code durch den Anwender

# Werkzeuge zur Sicherungspunktverwaltung

## Zweck

- Bei langlaufenden Programmen automatische Abspeicherung von Sicherungspunkten
- Nach z.B. Rechnerausfall Fortsetzung des Programms vom Sicherungspunkt aus

## Probleme

- Konsistente Sicherungspunkte hinsichtlich z.B. nicht abgeschlossener Kommunikationen
- Wiederanlauf mit veränderter Knotenzahl

## Status

- Nahezu keine automatischen Werkzeuge, stattdessen Integration in den Code durch den Anwender

# Werkzeuge zur Fehlertoleranz

## Zweck

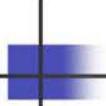
- Automatisches Tolerieren von Knotenausfällen
- Programm läuft auf anderer/kleinerer Konfiguration automatisch weiter
- Meist mit Sicherungspunkten realisiert

## Probleme

- Hoher Implementierungsaufwand

## Status

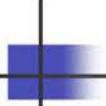
- Unterstützungskonzepte existieren



## Werkzeugarchitekturen

Zusammenfassung

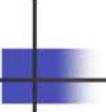
- Uns interessieren Werkzeuge für die späteren Lebenszyklusphasen der parallelen Programme
- Wir unterscheiden Offline- und Online-Werkzeuge
- Wir unterscheiden interaktive und automatische Werkzeuge
- Typisch: Fehlersuche online, Leistungsanalyse offline
- Online-Werkzeuge haben eine komplexe Struktur wegen der verwendeten Monitorsysteme
- **Monitorsystem ist selber verteiltes Programm**



## **Werkzeugarchitekturen**

Die wichtigsten Fragen

- Was unterscheidet Offline- und Online-Werkzeuge?
- Was unterscheidet interaktive und automatische Werkzeuge?
- Was versteht man unter interoperablen Werkzeugen?
- Welche Werkzeugtypen gibt es zur Unterstützung der Programmierung?
- Erläutern Sie die Struktur von Werkzeugen
- Was versteht man unter Instrumentierung?
- Beschreiben Sie die Struktur von Offline-Werkzeugen
- Beschreiben Sie die Struktur von Online-Werkzeugen



# Fehlersuche

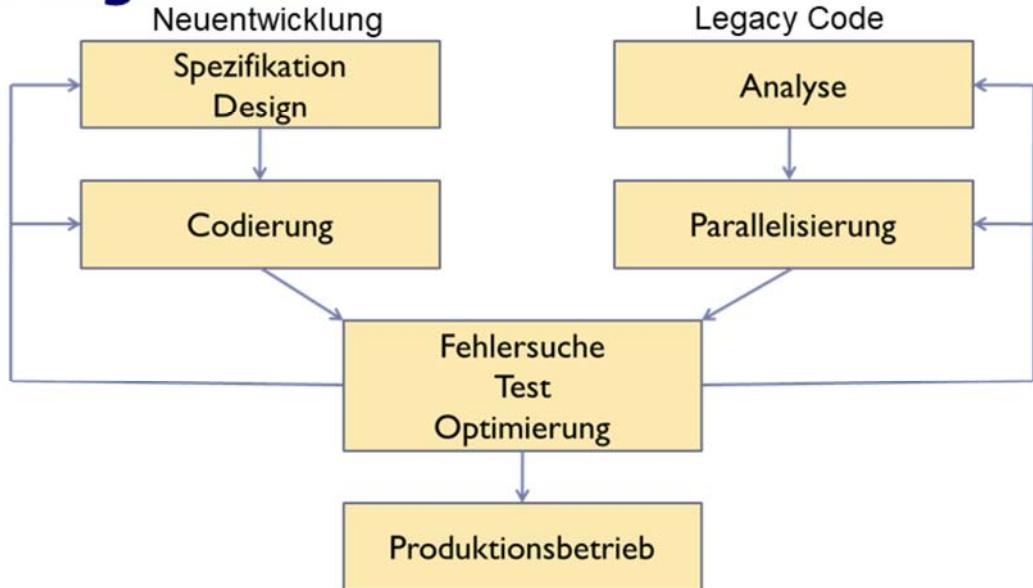
1. Entwicklungszyklus paralleler Programme
2. Fehlersuche
3. Häufige Fehlerquellen
4. Problemstellungen
5. Werkzeugunterstützung
6. Offline-Werkzeuge
7. Laufzeit-Debugger
8. Konzepte paralleler Debugger
9. Deterministische Ablaufkontrolle

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

657

# 1. Entwicklungszyklus paralleler Programme



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

658

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 viele 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

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

659

Siehe:

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

# Debugging?

9/9

0800 Aiken started  
1000 - stopped - aiken ✓  
13:00 (032) MP - MC { 1.2700 9.037847025  
033 PRO 2 1.3047645 9.037846985 convrt  
convrt 2.130676315  
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 Aiken started.  
1700 closed down.

Relay  
#70  
Bug

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

660

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

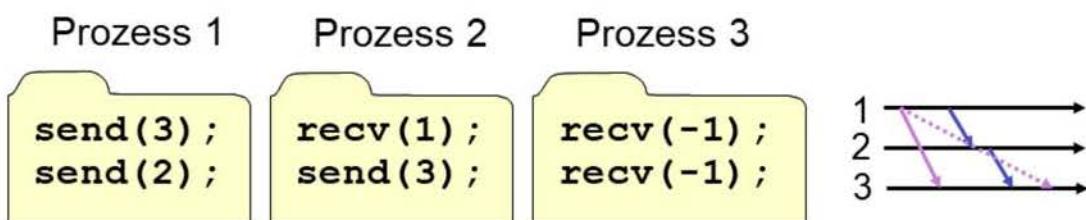
#### Parallele Programme

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

## Häufigste Fehlerquelle: Überholtvorgänge

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

Beispiel:

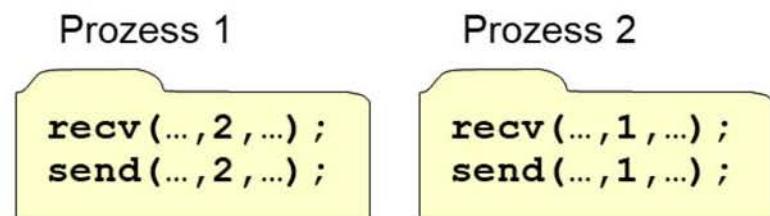


Konsequenz: Nichtdeterminismus,  
Nichtreproduzierbarkeit (schwer feststellbar)

## 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 (sog. Testorakel)

## Probleme bei parallelen Programmen

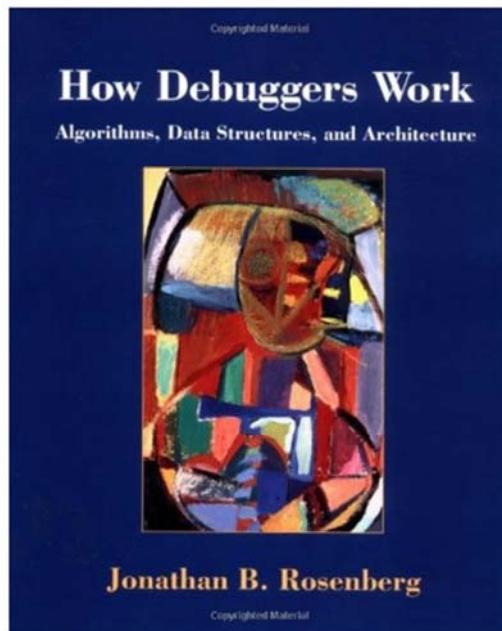
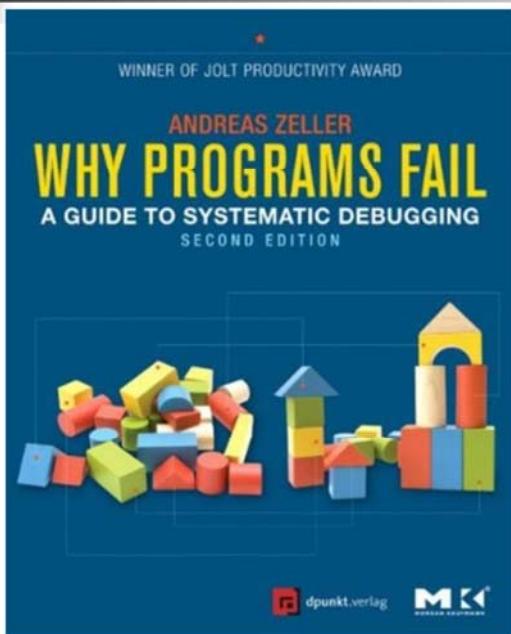
- Ergebnisse nicht nachrechenbar
- Ablauf abhängig von der Prozessorzahl
- Ablauf abhängig von zeitlichen Verhältnissen
- Häufig nicht bitidentisch nachrechenbar

→ 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



12.10.2021

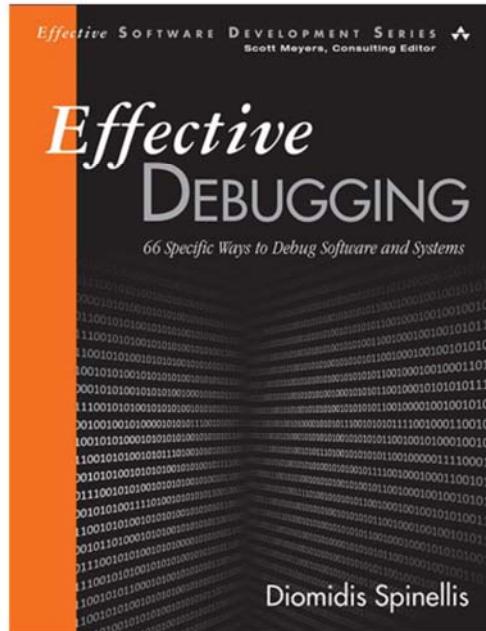
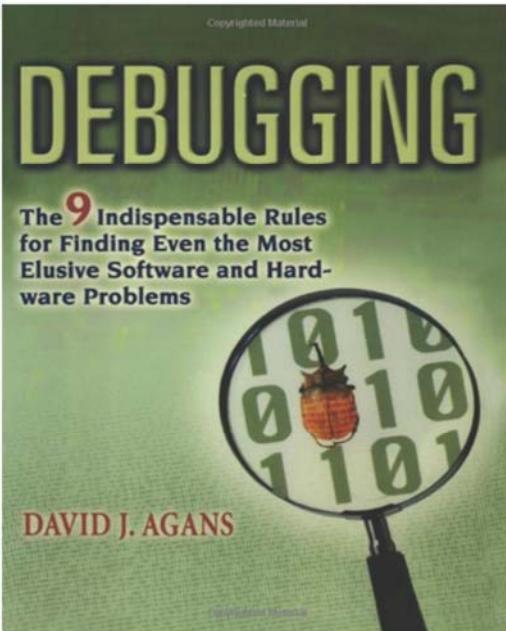
Hochleistungsrechnen - © Thomas Ludwig

668

Why Programs fail (2009) – zur Zeit in D vergriffen

How Debuggers Work (1996)

## Literatur



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

669

Debugging (2013)

Effective Debugging (2016)

# Literatur

OREILLY® technisch geprägtes material



## Weniger schlecht programmieren

Kathrin Passig &  
Johannes Jander

Urheberrechtlich geschütztes Material

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

670

Weniger schlecht programmieren (2013)

## 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)

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

672

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 temporale Ordnung der Ausgaben nicht gewährleistet, dadurch Kausalitäten unklar

## 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

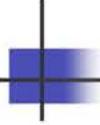
## 6. Spurbasierte Werkzeuge (offline)

### Merkmale

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

### Aufgezeichnete Ereignisse

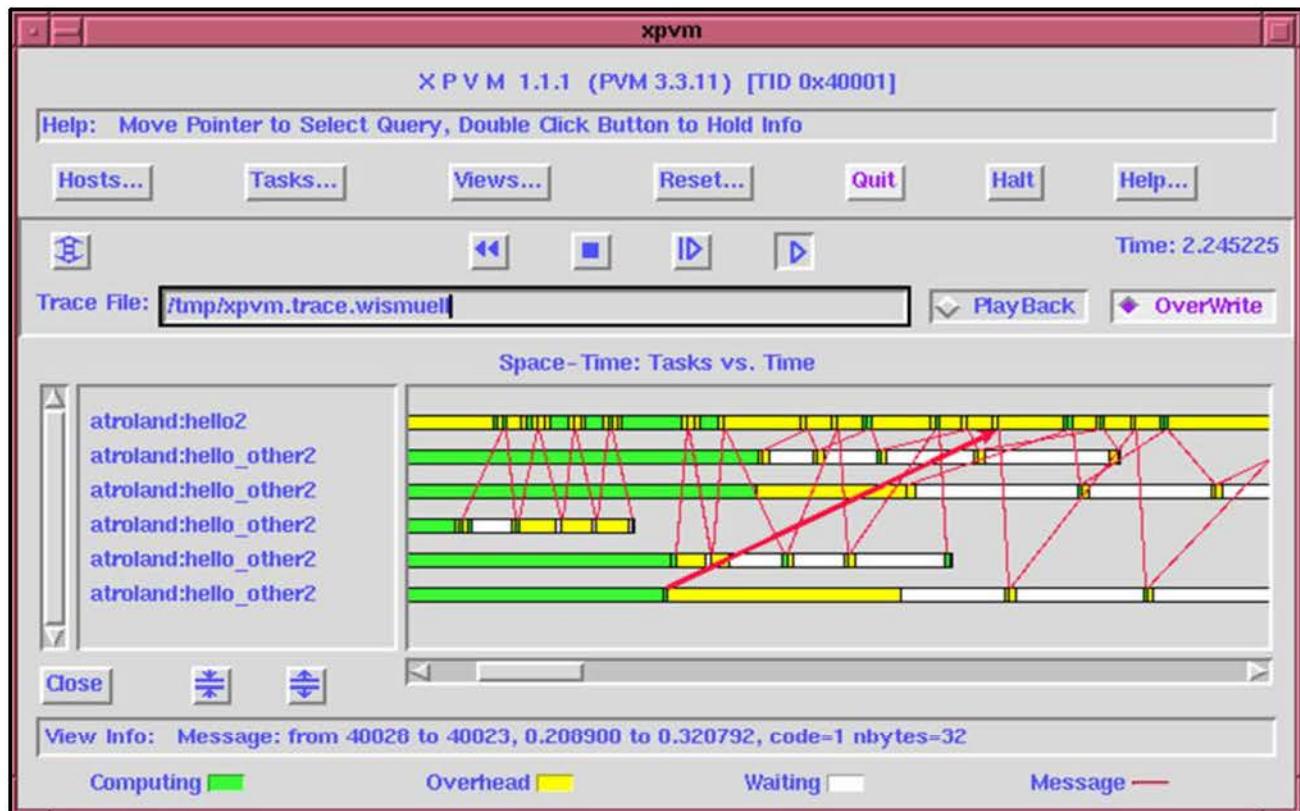
- Aufruf und Rückkehr der Funktionen der Programmierbibliothek  
Lokale Zeit, Dauer, Parameter
- Zum Teil auch benutzerdefinierte Ereignisse möglich



## 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, Variable
- 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, 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.

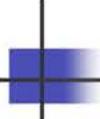
# Umgang mit mehreren Prozessen/Threads

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)

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

## 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

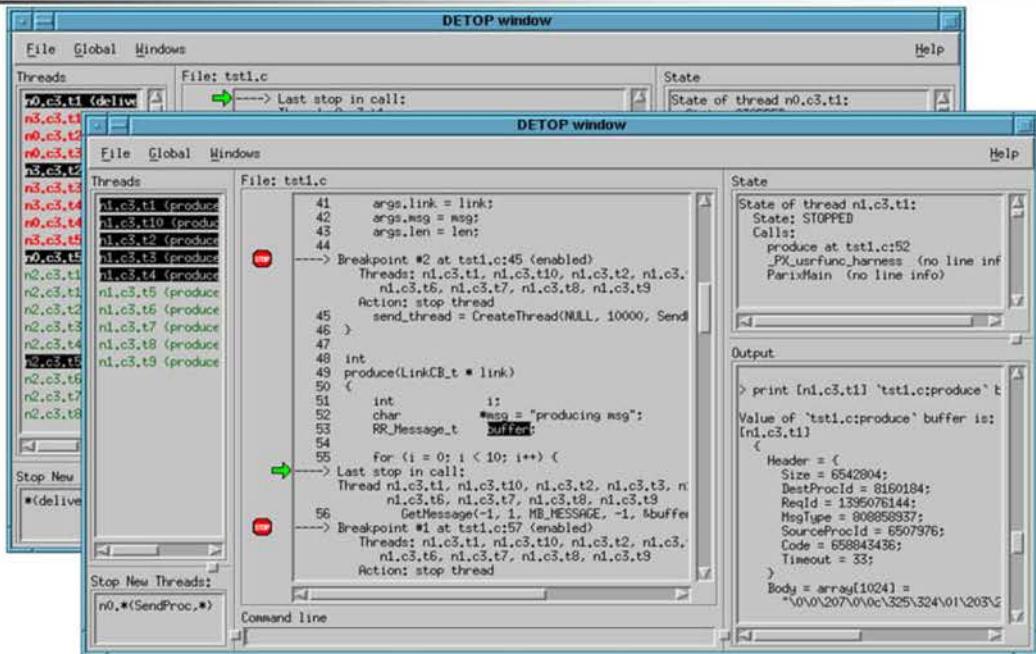
## Gemeinsame Daten verteilter Prozesse

- 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

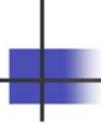
# Beispiel DETOP (1993)



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

686



## 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*

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

687

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.

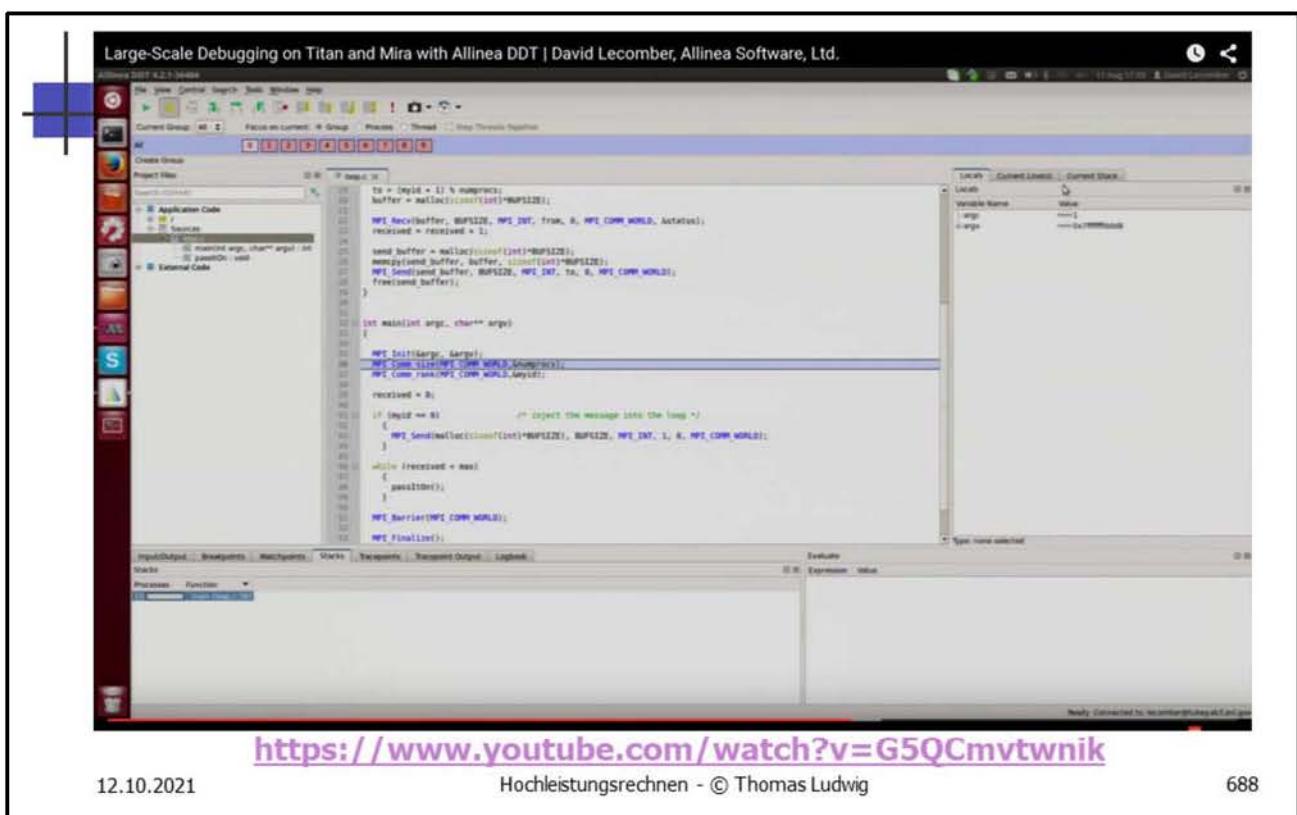
DDT's interface scales amazingly - providing the same clarity of information at thousands of processes as at a handful - highlighting commonality and differences with summary views and data comparison to focus your attention.

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

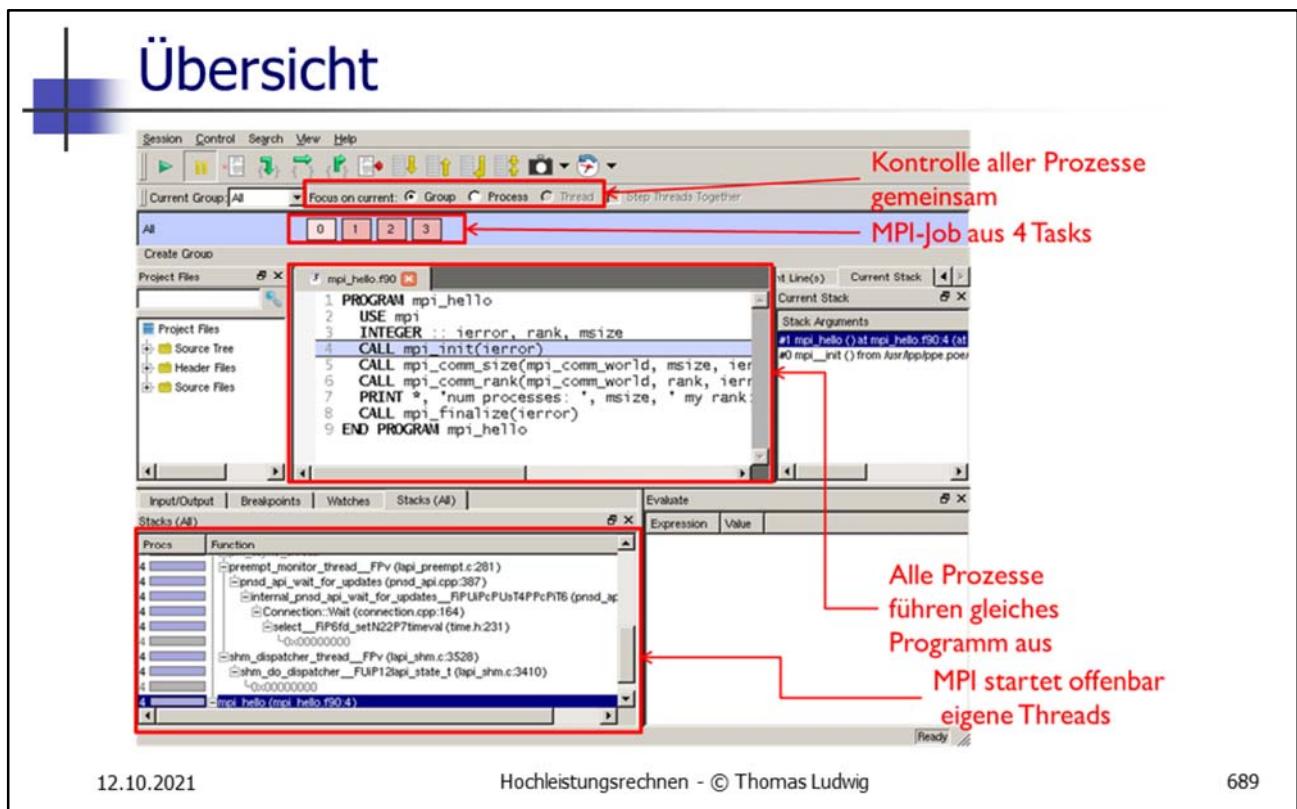
DDT's advanced memory debugging capability brings tremendous benefits to developers of scalar and parallel applications. DDT can find memory

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."



# Übersicht



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

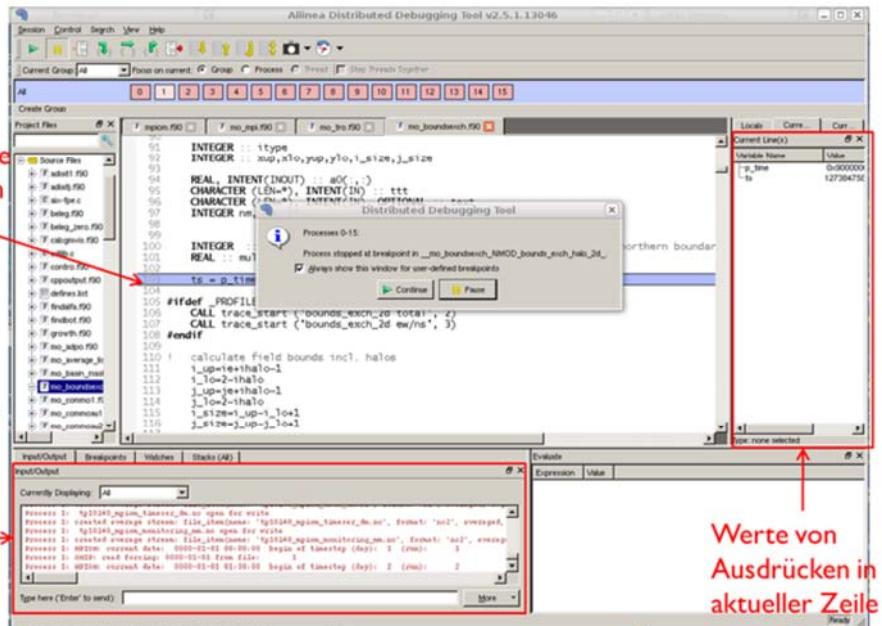
Standardausgabe und -fehler aller Prozesse

Werte von Ausdrücken in aktueller Zeile

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

690



Gezeigt ist ein Lauf von MPIOM mit 16 Prozessen.

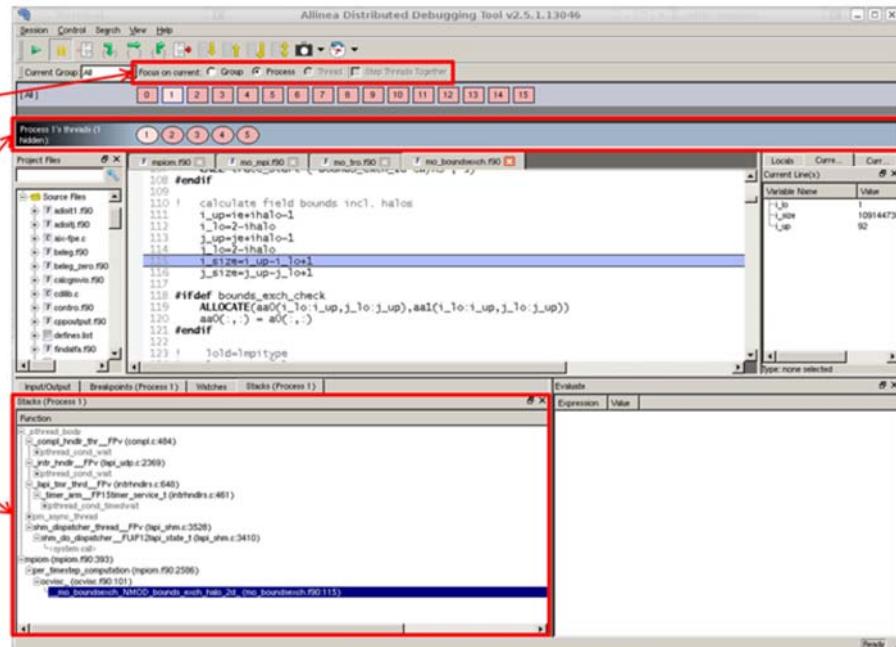
Gestoppt wurde durch einen Breakpoint auf die Kommunikationsfunktion boundsexch\_halo\_2d, es wird die erste ausführbare Zeile fokussiert, die erste Zeile der Funktionsdefinition ist Fortran-üblich bereits viele Zeilen vorher und deshalb nicht im Bild.

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



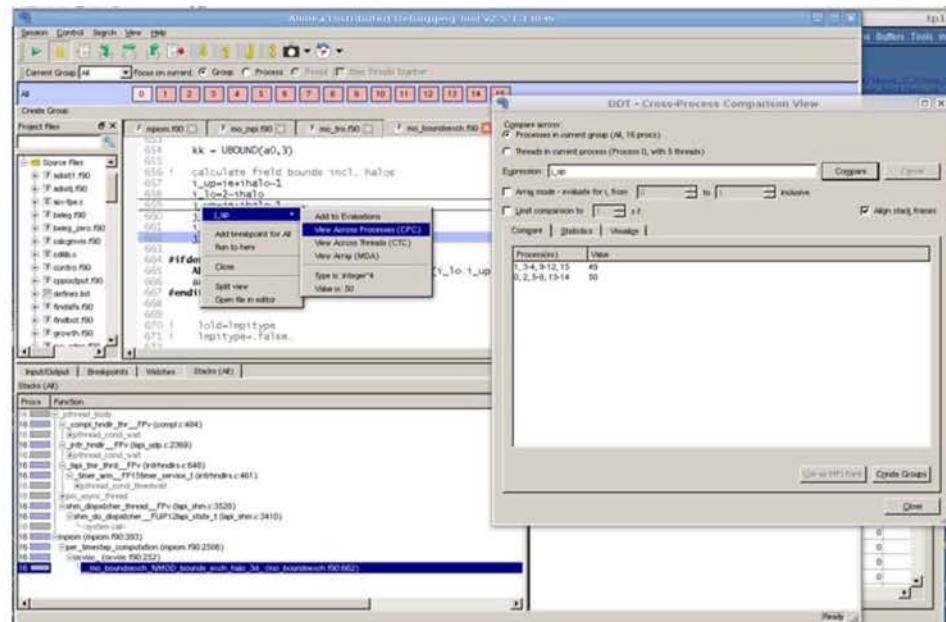
12.10.2021

691

## 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



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

692

## 8. Deterministische Ablaufkontrolle

### Problem des Nichtdeterminismus

- Erzwingen einer deterministischen Abarbeitungsreihenfolge der Kommunikation bei wiederholter Programmausführung  
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

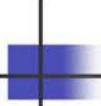
# Sicherungspunkte

## 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 Wiederauflauf des Programms von diesem Zeitpunkt aus.
- Evtl. gekoppelt mit Ablaufkontrolle



## Fehlersuche

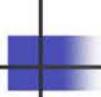
### Zusammenfassung

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

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

696



## 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 Überholvorgang?
- 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?



# Leistungsanalyse

1. Problemstellung
2. Ziele der Leistungsanalyse
3. Grundüberlegungen und Historie
4. Leistungsmaße
5. Amdahls Gesetz
6. Einfluss der Problemgröße
7. Superlinearer Speedup
8. Speedup-Bestimmung

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

698

## 1. Problemstellung

- Beurteilung der Leistungsfähigkeit des Hochleistungsrechners
  - Hardware, Betriebssystem, Compiler
- Beurteilung der Qualität der Parallelisierung  
Programmierbibliotheken, Programme
- Zur Kaufentscheidung  
Wieviel kostet es, wenn mein Programm in Zeit  $t$  berechnet sein muss?

## **2. Ziele der Leistungsanalyse**

- Optimierung der Ressourcenauslastung  
Prozessoren, Speicher, Netzwerk, Platten
- Leistungsoptimierung bei exklusiven Ressourcen  
Minimale Laufzeit des Programms
- Verweilzeitoptimierung bei nicht-exklusiver  
Ressourcennutzung  
Ebenso wichtig: Fairness zwischen den Programmen
- Visualisierung des dynamischen Ablaufs

### **3. Grundüberlegungen und Historie**

- Leistungsanalyse ist sehr komplexes Themengebiet
  - Vielfältige Literatur und Tutorials
- Hier nur einfacher Ansatz verfolgt:
  - Prozessoranzahl/Knotenanzahl (space-sharing-Betrieb)
  - Programmlaufzeit (wall clock time)

Annahme: Programme nutzen Ressourcen exklusiv
- Leistungsmodellierung/Leistungsvorhersage
  - Auch kompliziert und von Bedeutung

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

701

Space-sharing setzt sich hier vom Time-Sharing ab, das wir auf den einzelnen Rechnern haben: Bei Letzterem wird die Prozessorzeit auf die Prozesse zugeteilt. Die Zuteilung von Rechnerknoten zu einem parallelen Rechenjob eines Benutzers nennt man Space-Sharing, weil hier ein Teil der Maschine dem Benutzer zugewiesen wird.

Siehe: [http://en.wikipedia.org/wiki/Time\\_sharing](http://en.wikipedia.org/wiki/Time_sharing)

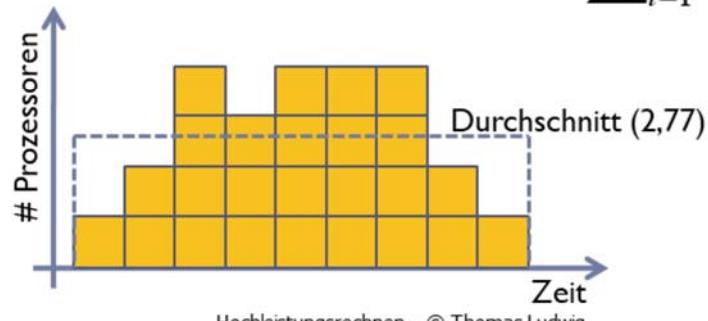
„Wall clock time“ ist die Zeit, die die Uhr an der Wand anzeigt. Also üblicherweise eine Genauigkeit lediglich im Sekundenbereich. Das genügt auch für die parallelen Programme, deren Laufzeiten ja mindestens im Minutenbereich liegen. Man misst hier die während des Programmlaufs verstrichene Zeit, d.h. wie lange wir auf das Ergebnis warten. Nicht zu verwechseln mit der Prozessorzeit, der Zeit also, die der Prozessor am Programm gearbeitet hat.

## Grundüberlegungen...

n Prozessoren, m ist maximaler Parallelismus ( $m \leq n$ )  
I ist Leistung

Gesamte geleistete Arbeit  $W = l \cdot \sum_{i=1}^n i \cdot t_i$

Durchschnittlicher Parallelismus  $A = \sum_{i=1}^m i \cdot t_i / \sum_{i=1}^m t_i$



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

702

Beachten Sie: diese merkwürdige Summenformel addiert über die y-Achse!

$$W = I * (1*2) + (2*2) + (3*1) + (4*4) = 25$$

$$A = 25 / 9$$

## Historische Überlegungen

Anfänglich gab es sehr pessimistische Überlegungen zum Leistungspotential von Parallelrechnern

Es wurde generell das Potential zur Leistungssteigerung kritisch gesehen bzw. verneint

Aus der Menge der Aussagen betrachten wir zwei:

- Minsky, 1971
- Amdahl, 1967

## Historische Überlegung: Minsky

- Ausgangspunkt  
Minsky betrachtet SIMD-Systeme (Vektorrechner)
- Überlegung: Viele Programme nutzen im ersten Arbeitsschritt alle Prozessoren, dann nur noch die Hälfte, ein Viertel usw.  
Beispiel: Addiere  $2p$  Zahlen mit  $p$  Prozessoren
- Voraussage: Durchschnittlicher Parallelismus gleich  $\text{Id}(p)$
- Bewertung: Es gibt praktisch keine Programme, die so aufgebaut sind

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

704

Das Beispiel entspricht dem Prinzip „mit Kanonen auf Spatzen“. So bekommt man aus keinem Rechner hohe Leistung. Die beschriebene Situation darf bestenfalls in der Endphase dieses Additionsvorganges auftreten.

Siehe:

- [https://en.wikipedia.org/wiki/Marvin\\_Minsky](https://en.wikipedia.org/wiki/Marvin_Minsky)
- [https://en.wikipedia.org/wiki/The\\_Society\\_of\\_Mind](https://en.wikipedia.org/wiki/The_Society_of_Mind)

“What magical trick makes us intelligent? The trick is that there is no trick. The power of intelligence stems from our vast diversity, not from any single, perfect principle”. – Marvin Minsky, *The Society of Mind*, p. 308

## 4. Leistungsmaße

- Speedup:  $S = T(1)/T(p)$   
 $T(1)$  ist die Rechenzeit auf einem Prozessor  
 $T(p)$  ist die Rechenzeit auf  $p$  Prozessoren
- Forderung  
Wähle jeweils den schnellstmöglichen Algorithmus  
(kritisch!!)
- Effizient:  $E = S/p$   
Normalisierung nach der Prozessorzahl.  $E$  gibt an,  
wieviel Prozent des maximalen Speedup  $S=p$  erreicht  
werden
- Vermutung:  $S < p$  und damit  $E < 1$

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

705

Siehe: <https://en.wikipedia.org/wiki/Speedup>

Der Speedup ist dann hoch, wenn im Quotient der Nenner sehr klein ist (das wollen wir ja), oder der Zähler sehr groß ist (das passiert irrtümlich manchmal).

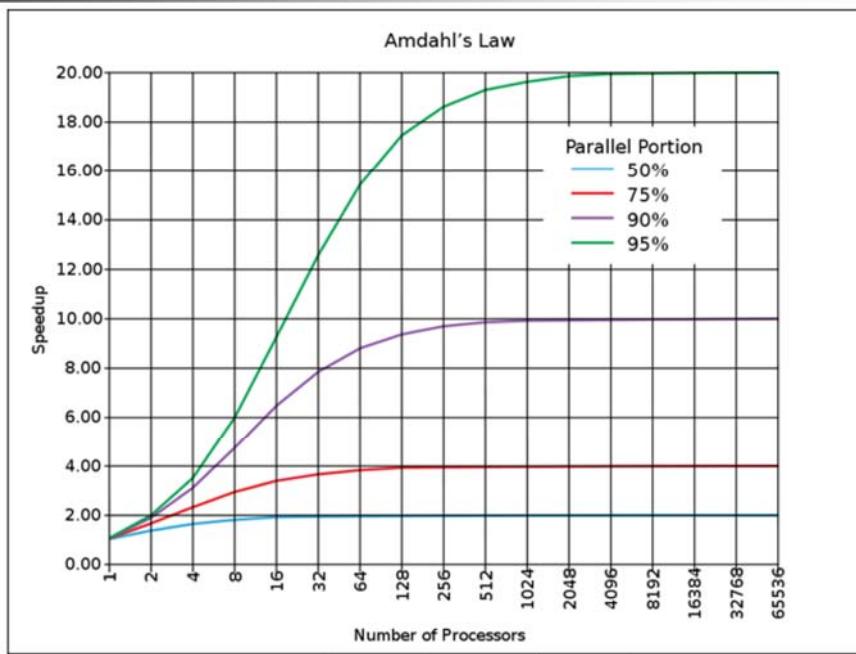
„Schnellstmöglicher Algorithmus“: Manche schnelle sequentielle (z.B. rekursive Verfahren) lassen sich so nicht parallelisieren. Die parallelisierbare Variante (z.B. eine iterative Umformung) ist zunächst deutlich langsamer.

## 5. Amdahls Gesetz

- Ausgangspunkt: Jedes Programm enthält einen Bruchteil  $f$  an Operationen, die nur sequentiell ausgeführt werden können
- Es gilt daher für den Speedup
$$S \leq 1/(f + (1-f)/p) \Rightarrow S_{\max} \leq 1/f$$
- Beispiel:  $f=0.01 \Rightarrow S_{\max}=100$
- Praktische Erfahrung  
Sequentieller Anteil meist sehr gering
- Bewertung: Amdahls Gesetz gilt! Man muss versuchen, den sequentiellen Anteil klein zu halten

Siehe: [https://en.wikipedia.org/wiki/Amdahl%27s\\_law](https://en.wikipedia.org/wiki/Amdahl%27s_law)

## Amdahls Gesetz...



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

707

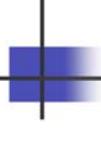
Quelle: Wikimedia Commons  
(<https://en.wikipedia.org/wiki/File:AmdahlsLaw.svg>)

# Ursachen von Leistungsverlusten

- Zugriffsverluste
  - Bei der Überwindung von Distanzen beim Datenaustausch zwischen Systemkomponenten
    - z.B. Nachrichtenaustausch oder entfernter Speicherzugriff bei (cc)NUMA
- Konfliktverluste
  - Bei der gemeinsamen Nutzung von Ressourcen durch mehrere Programmeinheiten
    - z.B. beim Bus- und Netzzugriff
- Auslastungsverluste
  - Bei zu geringem Parallelitätsgrad des Programms
    - z.B. permanente oder zeitweilige Lastungleichheit

## Ursachen von Leistungsverlusten...

- Bremsverluste
  - Beim Beenden gewisser Berechnungen, wenn bereits eine Lösung gefunden wurde
    - z.B. Suchbaumverfahren
- Komplexitätsverluste
  - Durch Zusatzaufwand im parallelen Programm gegenüber dem sequentiellen
    - z.B. Aufteilung der Daten
- Wegwerfverluste
  - Ergebnis mehrfach berechnet, aber nur eines von ihnen weiterbenutzt
    - z.B. Doppelberechnungen bei globalen Operationen



## Ursache von trügerischen Leistungsgewinnen

- Geänderte Ressourcennutzung
  - Bei fester Problemgröße und steigender Anzahl Prozessoren: relevante Daten- und Code-Anteile passen auf einmal wieder (besser) in den Cache  
=> deutlich schnellere Abarbeitung

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

710

Möglicherweise entsteht hier superlinearer Speedup. Das ist dann leicht zu erkennen. Aber meistens wird einfach nur die Kurve nach oben gedrückt, was sich eigentlich erstmal gar nicht erkennen lässt. Es sieht dann womöglich aus wie guter Speedup.

## 6. Einfluss der Problemgröße

Wir wollen nun betrachten, wie sich in verschiedenen Situationen die Aufwände für Kommunikation und Berechnung zueinander verhalten

### Schwache Skalierung (weak scaling)

- Bei schwacher Skalierung behalten wir die Problemgröße **pro Prozessor** bei und erhöhen die Anzahl der Prozessoren. Die Problemgröße nimmt dabei also zu!

### Starke Skalierung (strong scaling)

- Bei starker Skalierung behalten wir die gesamte Problemgröße bei und erhöhen die Anzahl der Prozessoren. Jeder hat dann zunehmend weniger zu tun – dies wird bei der Speedup-Bestimmung typischerweise zugrunde gelegt. (Amdahls Gesetz)

Schwache Skalierung ergibt eine bessere Rechnernutzung aber eignet sich nicht für jeden Problemtyp.

## Einfluss der Problemgröße...

### Beispiel: Berechnung einer Matrix

- Größe:  $1000 \times 1000$  Elemente
- Betrachtung eines Iterationsschrittes
- Berechnung dauert  $1000 \times 1000 \times 1$  Zeiteinheit

### Parallelisierung mit $p=5$

- Aufteilung in Blöcke von Zeilen
- Berechnung:  $200 \times 1000 \times 1$  Zeiteinheit
- Kommunikation: benachbarte Blöcke tauschen eine Zeile aus (alle gleichzeitig)  
Aufwand:  $2 \times 1000 \times 1$  Zeiteinheit
- Berechnung/Kommunikation-Verhältnis: 100:1  
( höher ist besser!)

## Einfluss der Problemgröße...

Wir verwenden mehr Prozessoren (starke Skalierung)

### Parallelisierung mit $p=10$

- Aufteilung in Blöcke von Zeilen
- Berechnung:  $100 \times 1000 \times 1$  Zeiteinheit
- Kommunikation: benachbarte Blöcke tauschen eine Zeile aus (alle gleichzeitig)
  - Aufwand:  $2 \times 1000 \times 1$  Zeiteinheit
  - Berechnung/Kommunikation-Verhältnis: 50:1

### Parallelisierung mit 100 Prozessoren

- Berechnung/Kommunikation-Verhältnis: 5:1

## Einfluss der Problemgröße...

Wir vergrößern das Problem (schwache Skalierung)

Parallelisierung mit  $p=10$  und doppelt so großer Matrix

- Berechnung:  $141 \times 1414 \times 1$  Zeiteinheit
- Kommunikation:  $2 \times 1414 \times 1$  Zeiteinheit
- Berechnung/Kommunikation-Verhältnis: 70:1  
( $p=5$  und Matrix  $1000 \times 1000$ : 100:1)

Trotz doppelter Prozessoranzahl und doppelter Problemgröße verschlechtert sich das B/K-Verhältnis

## Einfluss der Problemgröße...

Wir kaufen neue Prozessoren, das Netz bleibt gleich

Parallelisierung mit  $p=10$  und doppelt so großer Matrix

- Berechnung:  $141 \times 1414 \times \mathbf{0.3}$  Zeiteinheiten
- Kommunikation:  $2 \times 1414 \times 1$  Zeiteinheit
- Berechnung/Kommunikation-Verhältnis: 21:1

Zu dumm: die Effizienz der Parallelisierung wurde dadurch verschlechtert!

## 7. Superlinearer Speedup

### Aufgabenstellung

- $n$  Sortieralgorithmen mit Laufzeiten  $t_i$
- Finde den besten Sortieralgorithmus

### Sequentielles Programm

- Lasse den ersten Algorithmus laufen; liefert  $t_{\min} = t_1$
- Starte nächsten Algorithmus; wenn er  $t_{\min}$  überschreitet, dann sofort stoppen; andernfalls  $t_{\min} = t_j$
- Wiederhole mit allen Algorithmen
- Zeitbedarf  $t_{\text{seq}} \geq t_{\min} \times n$

# Superlinearer Speedup...

## Paralleles Programm

- Nimm n Prozessoren
- Starte n Algorithmen gleichzeitig auf den Prozessoren
- Stoppe die Berechnung, wenn der erste fertig ist
- Zeitbedarf:  $t_{\text{par}} = t_{\text{min}}$

Erzielter Speedup:  $S = t_{\text{seq}} / t_{\text{par}} \Rightarrow S >= n !?$

Perpetuum mobile des parallelen Rechnens?



## Superlinearer Speedup...

### Analyse des Beispiels:

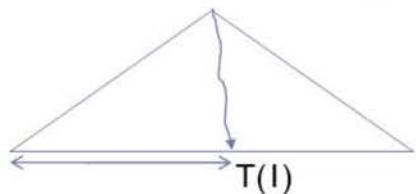
- Sequentielles Programm ist nicht „das schnellstmögliche“ sequentielle Programm
- Speedup-Definition fordert „schnellstmögliche“ Programm
- „Schnellstmöglich“ hier: quasiparallele Abarbeitung des Algorithmenvergleich auf einem Prozessor
- Damit verschwindet dann der superlineare Speedup

# Superlinearer Speedup...

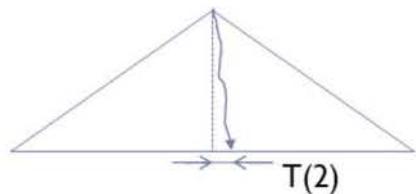
Suchbaum-Verfahren sequentiell/parallel (Tiefensuche)

- Wir suchen **eine** Lösung

Fall a)

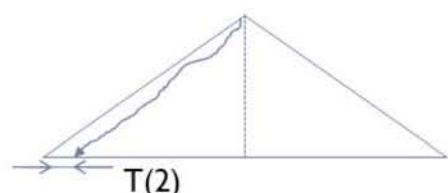
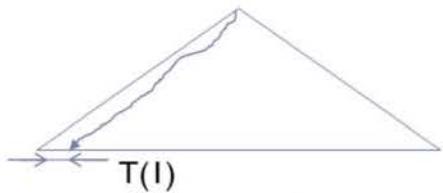


sequentiell



parallel

Fall b)



## Superlinearer Speedup...

Analyse des Beispiels:

- Für breite Bäume geht im Fall a) der Speedup gegen unendlich
- Für breite Bäume geht im Fall b) der Speedup gegen eins

Angabe des Speedup sinnlos!

Parallelisierung ebenso?

Bei Suche nach allen Lösungen problemlos!

- Hier verschwindet das Problem

12.10.2021

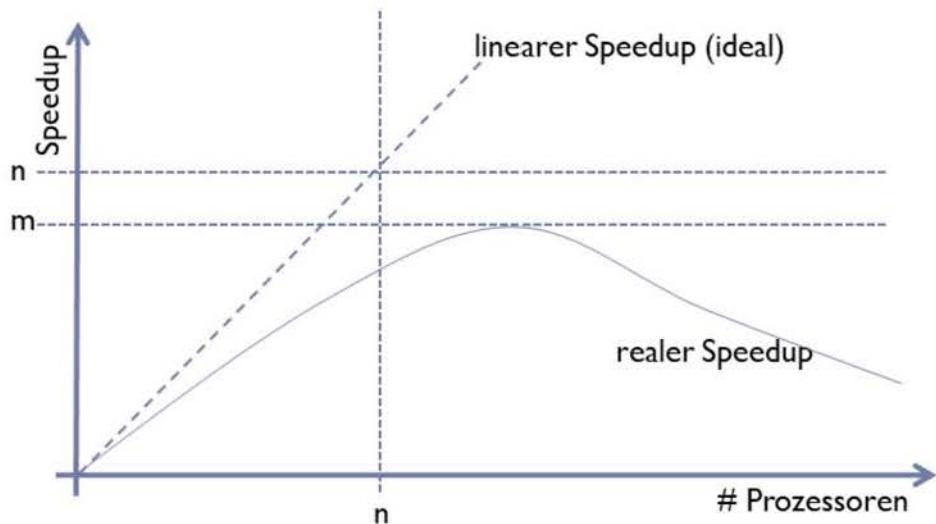
Hochleistungsrechnen - © Thomas Ludwig

720

Der Anwendungsfall findet sich leider häufig bei nichtnumerischen Verfahren, die Baumsuche verwenden: diskrete Optimierung, Formelbeweiser usw.

## 8. Speedup-Bestimmung

Verwendet in wissenschaftlichen Artikeln, in denen die Verfahren parallelisiert wurden



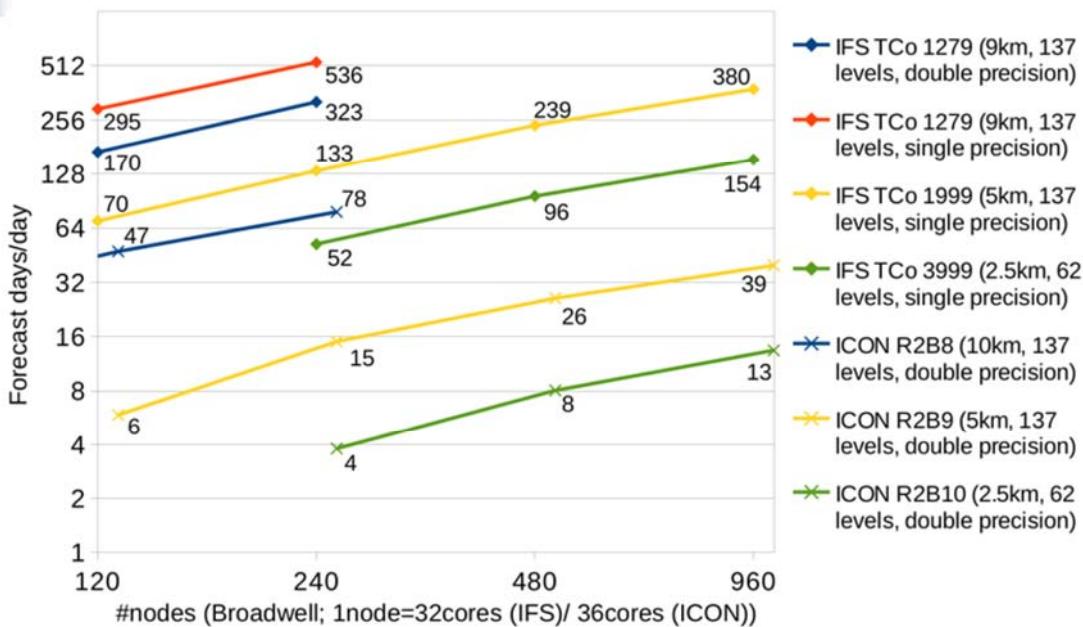
## Speedup-Bestimmung...

Speedup-Kurven angegeben für festen Datensatz

Kurvenverlauf

- Bei guten numerischen Verfahren
  - $S$  nahe bei  $p$  für alle  $p$ , die wir einsetzen
  - Effizienz zwischen 80% und 100%
- Bei schlechten Verfahren
  - Effizienz von 50% oder schlechter

## Speedup-Bestimmung... Klima-/Wettercodes



12.10.2021

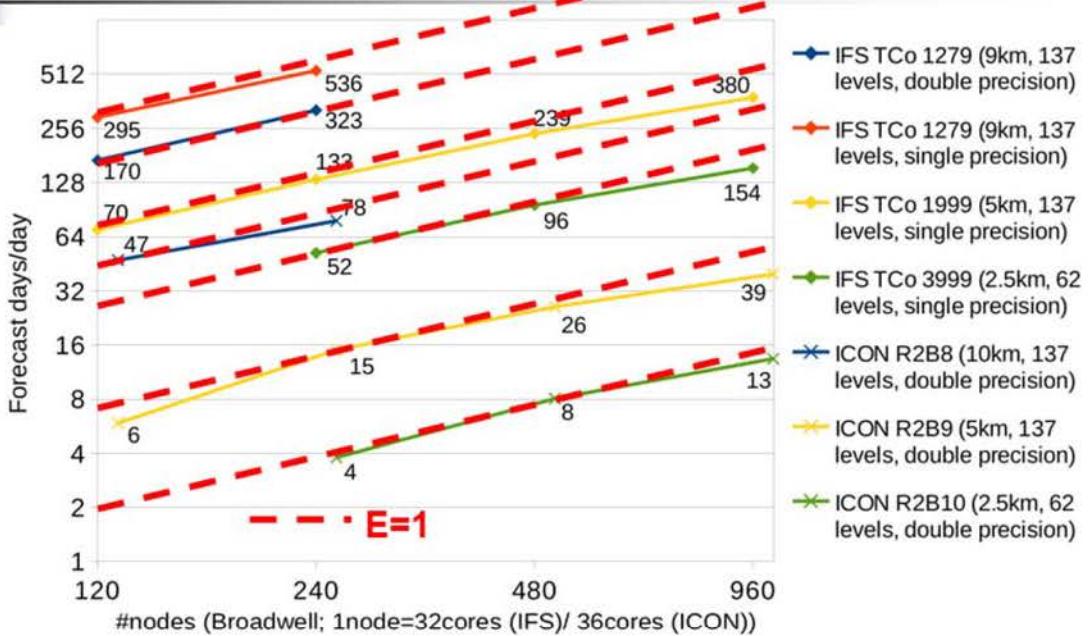
Hochleistungsrechnen - © Thomas Ludwig

723

Die Klimaforscher präferieren als y-Achse die Metrik "Simulated Years per Day (SYPD)", da die Klimasimulationen viele simulierte Jahre beinhalten. Bei den Ensembles z.B. handelt es sich um mehrere hunderte (oder Tausende) Modell-Jahre. Diese Metrik besagt, wie viele Modell-Jahre man innerhalb eines (Kalender-)Tages (also Wallclock=24 Stunden) bei einer gewählten Anzahl von Rechnerknoten rechnen könnte.

Eine weitere Metrik, die für mittelfristige Vorhersage (z.B. ECMWF) und auch für Wetterleute interessant ist, ist die Metrik "Forecast Days per Day". Die Grafik zeigt einen Vergleich zwischen dem IFS-Modell vom ECMWF und dem ICON-Modell vom DWD/MPI-M. Da sieht man, dass die y-Achse die Forecast days/day anzeigt. Wenn man mit ICON in einer sehr hohen Auflösung z.B. R2B9 (5km) oder R2B10 (2.5km) rechnet, dann kann man auch nur ein paar Modell-Tage pro Tag an Durchsatz erreichen. Das sind dann Anwendungen für einen Exascale Rechner.

## Speedup-Bestimmung... Klima-/Wettercodes



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

724



## Speedup-Bestimmung...

Was tun, wenn die Kurven nicht optimal aussehen?

David Bailey

*Twelve Ways to Fool the Masses When Giving  
Performance Results on Parallel Computers*

Supercomputer Review, August 1991

Liste sehr beliebter (Selbst-)Täuschungsmethoden zur  
Erzielung besserer Ergebnisse

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

725

Siehe: <http://crd.lbl.gov/~dhbailey/dhbpapers/twelve-ways.pdf>



## Speedup-Bestimmung...

- (2) Present performance figures for an inner kernel, and then represent these figures as the performance of the entire application
- (4) Scale up the problem size with the number of processors, but omit any mention of this fact
- (5) Quote performance results projected to a full system
- (7) When direct run time comparisons are required, compare with an old code on an obsolete system



## Speedup-Bestimmung...

- (12) If all else fails, show pretty pictures and animated videos, and don't talk about performance

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

727

„It sometimes happens that the audience starts to ask all sorts of embarrassing questions.

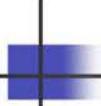
These people simply have no respect for the authorities of our field. If you are so

unfortunate as to be the object of such disrespect, there is always a way out --- simply

conclude your technical presentation and roll the videotape. Audiences love razzle-dazzle

color graphics, and this material often helps deflect attention from the substantive

technical issues.“ David Bailey



## Leistungsanalyse

### Zusammenfassung

- Leistungsanalyse dient der Rechner- und Programmbewertung
- Es gibt unterschiedliche Ursachen für Leistungsverluste
- Speedup und Effizienz charakterisieren die Qualität des parallelen Programms
- Amdahl: der Speedup ist nach oben begrenzt
- Problemgröße beeinflusst den erreichbaren Speedup
- Wir unterscheiden Strong und Weak Scaling
- Superlinearer Speedup ist nicht systematisch nutzbar
- Es gibt viele Möglichkeiten für irrtümlich oder absichtlich falsche Speedup- bzw. Leistungs-Angaben

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

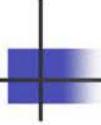
728



## Leistungsanalyse

### Die wichtigsten Fragen

- Was wollen wir mit der Leistungsanalyse bewerten?
- Welche Ursachen von Leistungsverlusten gibt es?
- Welche pessimistische Abschätzung machte Minsky?
- Welche zwei Maße charakterisieren typischerweise die Qualität einer Parallelisierung?
- Was beschreibt Amdahls Gesetz?
- Welchen Effekt hat ein sequentieller Anteil von n% auf die Leistungsausbeute?
- Wie verändern sich Aufwendungen von Berechnung und Kommunikation bei veränderten Datenstrukturen oder veränderter Hardware?
- Was ist der Unterschied zwischen Weak und Strong Scaling?
- Was ist superlinearer Speedup?
- Welche Formen der Parallelisierung widersetzen sich einer eindeutigen Speedup-Ermittlung?
- Wie sehen typische Speedup-Kurven aus für reale Anwendungen aus?



# Leistungsoptimierung

1. Ziele und Vorgehensweise
2. Messverfahren
3. Primitive Hilfsmittel zur Messung
4. Beispielwerkzeug Vampir
5. Optimierung des Programm-Codes
6. Messmethodik

12.10.2021

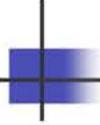
Hochleistungsrechnen - © Thomas Ludwig

730

# **1. Ziele und Vorgehensweise**

Ziele der Leistungsoptimierung

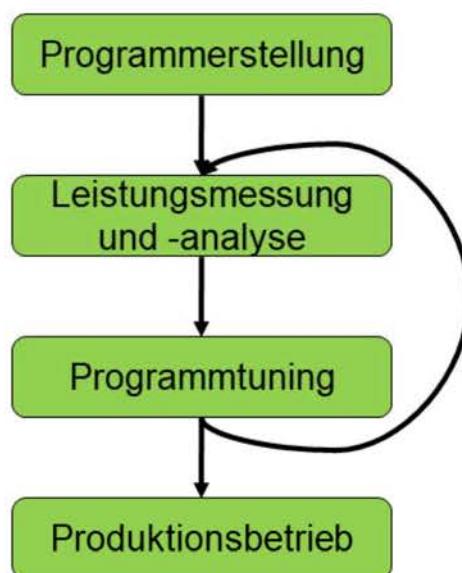
- Ausführungszeit minimieren
  - Programmergebnis so schnell wie möglich
- Durchsatz maximieren
  - So viele zufriedene Benutzer wie möglich
- Auslastung optimieren
  - Optimale Nutzung der investierten Gelder



## Ziele der Leistungsmessung

- Wie gut werden die Ziele erreicht?
- Zu diesem Zweck Leistungsmessung
  - Zunächst Messung der Leistungsdaten
  - Bewertung der Leistungsdaten
  - Code-Änderungen oder Änderung der Nutzungsweise des Rechners

# Vorgehensweise



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

733

## Vorgehensweise...

- Hypothese über Grund des Leistungsmangels aufstellen
- Messungen durchführen
- Wenn Hypothese richtig
  - Leistungsengpass korrigieren
  - Evtl. vorher Hypothese verfeinern
- Wenn Hypothese falsch
  - Andere Hypothese wählen
- Konzeptionell wie bei der Fehlersuche
  - „Debugging for performance“

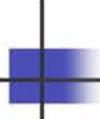
## **2. Messverfahren**

Wir kennen schon zwei Klassen

- Offline-Verfahren (Spurbasiert)
  - Erfasst Daten zur Laufzeit
  - Auswertung nach Programmende
- Online-Verfahren
  - Erfasst Daten zur Laufzeit
  - Auswertung zur Laufzeit
  - Änderung der Art der erfassten Daten

## Offline-Messverfahren

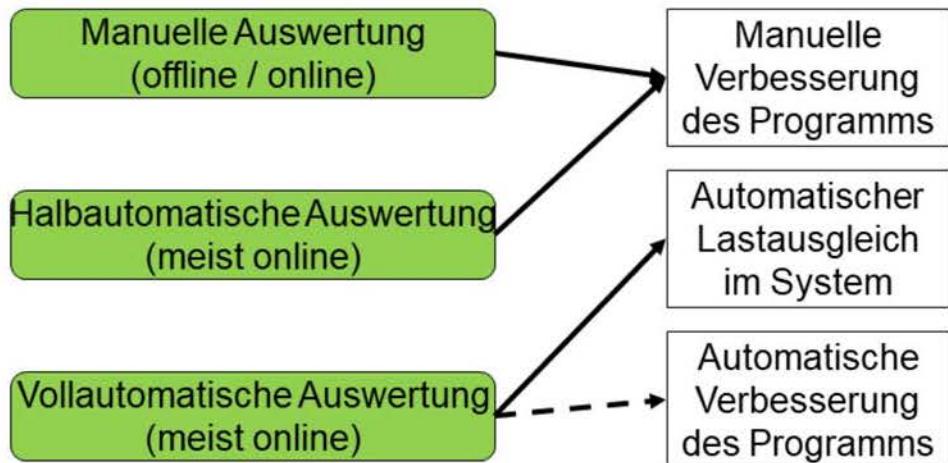
- Vorteile
  - Viele Informationen dauerhaft erfasst
  - Später beliebig auswählbar und analysierbar
  - Vergleich verschiedener Programmläufe möglich
- Nachteile
  - Konstante Zusatzlast durch Spurgenerierung
  - Nicht situationsbedingt verfeinerbar
- Probleme
  - Hohe Datenmenge (Mega-/Giga-/Terabyte-Bereich)
  - Zusammenfassen der verteilten einzelnen Spurdateien
  - Genau synchronisierte Uhren notwendig



## Online-Messverfahren

- Vorteile
  - Sofortige Reaktion möglich
  - Last der Instrumentierung regelbar
  - Bedingte Messung (einzelne Abschnitte)
- Nachteile
  - (Meist) keine nachträgliche Auswertung
- Probleme
  - Instrumentierung des laufenden Programms
  - Datentransport zum zentralen Programmteil
  - Online-Verfahren vs. Batch-Betrieb im Cluster

# Benutzungsmethoden





## Messwerkzeuge

- Primitive Hilfsmittel
  - C-Funktionalität
- Offline-Werkzeuge
  - Vampir
  - viele andere...
- Online-Werkzeuge
  - (Paradyn)

### **3. Primitive Hilfsmittel zur Messung**

- Manchmal sind Werkzeuge zu aufwendig und/oder zu teuer
- Rückgriff auf das, was die Programmierbibliotheken bereits anbieten
- (Erinnerung: Lieblingsdebugger: printf() ☺)

## Primitive Hilfsmittel...

- **clock\_gettime()**-Funktion
  - Liefert CPU-Zeit mit Nanosekundenauflösung

```
#include <time.h>

struct timespec starttime, endtime;
time_t elapsed;

clock_gettime(CLOCK_REALTIME, &starttime);
/* Code */
clock_gettime(CLOCK_REALTIME, &endtime);

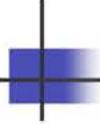
elapsed = endtime.tv_sec - starttime.tv_sec;
```

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

741

CLOCK\_REALTIME definiert einen von mehreren  
verfügbarer Zeitzählern.



## Offline-Werkzeuge: Allgemein

- Betrachtung aus Sicht eines Informatikers
  - Erstellung der Spuren schwierig
    - Zusätzlicher Code an verschiedenen Stellen im System
    - Zusammenführung
    - Zeitsynchronisation
  - Auswertung der Spuren macht Spaß
    - Verschiedenste bunte Displays, die die enthaltenen Informationen unterschiedlich darstellen
  - Folge: Es gibt viele Offline-Werkzeuge ☺



## Praktische Fragestellung

- Mein Programm hat 100 Funktionen, läuft 5 Tage lang auf 256 Knoten mit 512 Prozessen
- Frage: Wo ist der Leistungsengpass im Programm?
  
- Mit Vampir: Langes Betrachten bunter Bilder(?)
- (Mit Paradyn: Halbautomatisches Auffinden des Engpasses)

## 4. Beispielwerkzeug Vampir

- Seit 1996 kommerziell verfügbar
  - Entwicklung seit den frühen 90er Jahren
- Entwickler
  - Jülich Supercomputer Center
  - Technische Universität Dresden
- Eigenschaften
  - Aufwendige Displays
  - Komplexe Spuraufzeichnung und -verwaltung

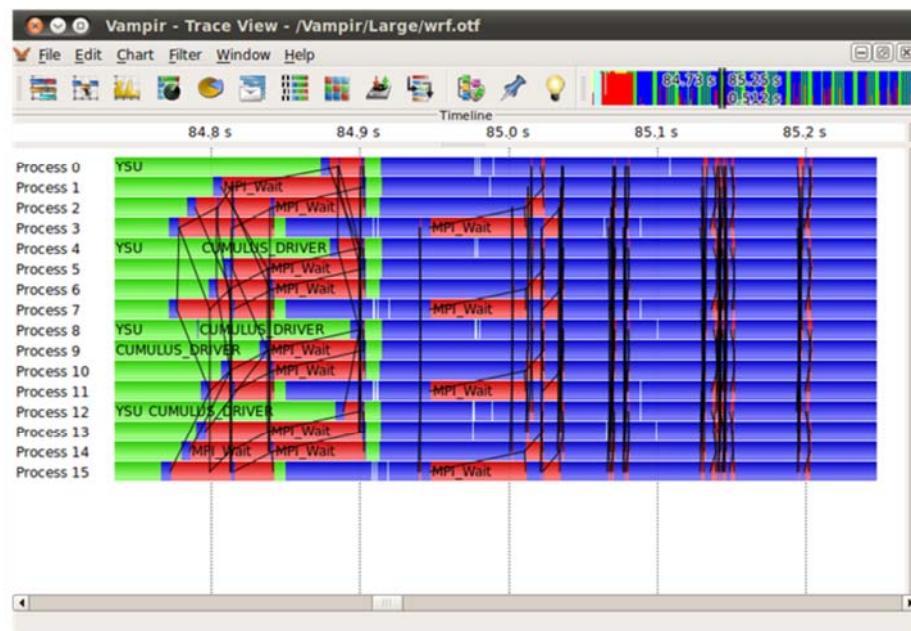
12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

744

Siehe: <https://vampir.eu/>

# Vampir: Master Timeline



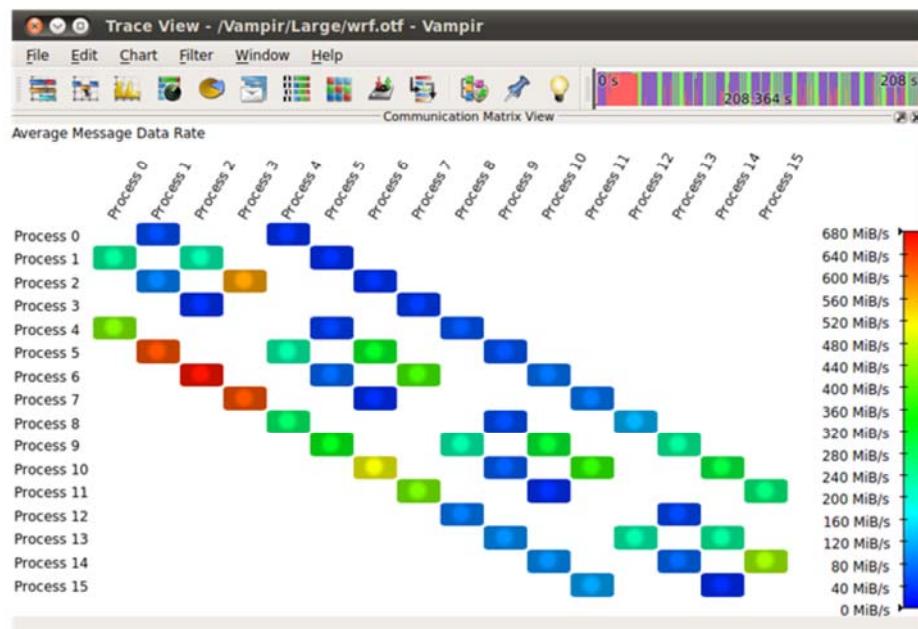
12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

745

Siehe: [https://vampir.eu/tutorial/manual/performance\\_data\\_visualization](https://vampir.eu/tutorial/manual/performance_data_visualization)

# Vampir: Communication Matrix



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

746

Siehe: [https://vampir.eu/tutorial/manual/performance\\_data\\_visualization](https://vampir.eu/tutorial/manual/performance_data_visualization)

## 5. Optimierung des Programmcodes

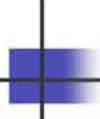
Was kann man denn nun konkret verändern?

- Code-Aufbau und Datenstrukturen
- Verteilung der Daten zu den Prozessen
- Kommunikation
- Synchronisation
- Abbildung der Prozesse auf die Prozessoren



## Optimierung des Programms...

- Code-Aufbau und Datenstrukturen
  - Bessere Nutzung des Caches durch geänderte Datenstrukturen und Code-Strukturen
    - Anordnung der Feld-Elemente
    - Aufbau der Schleifen
  - Datenstrukturen so aufbauen, dass man einfach senden und empfangen kann



## Optimierung des Programms...

- Verteilung der Daten zu den Prozessen
  - Gleichverteilung der Daten nur, wenn das auch Gleichverteilung der Lasten bedeutet
  - Bei Gitterstrukturen
    - Kanten, entlang derer kommuniziert wird, minimieren
  - Evtl. dynamische Umverteilung der Daten ermöglichen



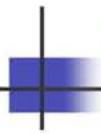
## Optimierung des Programms...

- Kommunikation
  - So **selten** wie möglich wegen Zeitverlust
  - Möglichst große Pakete wegen Aufsetzzeit
  - Möglichst **frühzeitig senden**, damit Empfänger die Daten schnell bekommen
  - Möglichst **spät empfangen**, damit Sender Zeit gewinnen
  - Möglichst **asynchrones Empfangen** und Überlagerung mit anderen Berechnungen
    - Vorsicht: schwer auffindbare Programmierfehler!



## Optimierung des Programms...

- Synchronisation
  - Möglichst selten globale Synchronisation
  - Möglichst selten globale Kommunikation, da diese meist auch synchronisieren
  
  - Zu häufige Synchronisation: Zu langsam
  - Zu seltene Synchronisation: Komplizierte Fehler



## Optimierung des Programms...

- Abbildung der Prozesse und Threads auf die Prozessoren und Kerne
  - Gleichbelastung der Prozessoren/Kerne erzielen
  - Minimierung der Kommunikation zwischen den Prozessoren/Kernen erzielen
  - Scheduling des Betriebssystems auf dem einzelnen Knoten bedenken

## 6. Messmethodik

- Ziel: Physikalische Größen quantitativ erfassen
  - Z. B. Durchsatz in MB/s, Latenz, Laufzeiten
- Veröffentlichte Messwerte sollten wissenschaftlich fundiert erhoben werden
  - Messwerte sind fehlerbehaftet
  - Es ist üblicherweise nicht ausreichend nur eine einzelne Messung durchzuführen

# Messfehler

- Zufällige Fehler
  - Heben sich bei unendlicher Messung im Mittel auf (?)
  - Ergebnisse variieren immer etwas
    - Natürliche Varianz (Netzwerkübertragungen, Speicherzugriffe)
    - Betriebssystemaktivitäten im Hintergrund
  - Größere Abweichungen sind auch möglich
    - Benutzung der Ressourcen durch andere Benutzer
    - Hardwarefehler (RAID-Rebuild, Paketverluste, ...)
    - Lastbalancierung
- Systematische Fehler
  - Heben sich bei unendlicher Messung im Mittel nicht auf
    - Cache-Effekte berücksichtigen
  - Falsche Messmethodik/-realisierung

## Messfehler

- Was kann man dagegen unternehmen?
  - Wohldefinierte Hard-/Softwareumgebung
  - Dokumentation der Versuchsanordnung
  - Äußere Einflüsse minimieren
    - Z. B. Einzelbenutzerbetrieb während der Messung
  - Messdauer erhöhen
  - Messungen wiederholen
    - Eliminiert zufällige Fehler
  - Vergleichen der Messwerte mit erwarteter Leistung
    - Z. B. durch Modellierung

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

755

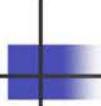
Die Dokumentation der Versuchsanordnung ist wichtig um eine Reproduzierbarkeit zu gewährleisten.

Eine Minimierung der äußeren Einflüsse ist nicht immer möglich, insbesondere in Mehrbenutzersystemen.

Wiederholungen sollten zu verschiedenen Zeiten gemacht werden um die Einflüsse von temporären Störungen zu reduzieren.

Kurzzeitige Einflüsse können durch länger andauernde Testläufe reduziert werden bzw. im Mittel bestimmt werden.

Aber eigentlich wollen wir so kurz wie möglich testen um das System für sinnvolle Aufgaben zu nutzen.



## **Leistungsoptimierung**

### Zusammenfassung

- Leistungsmessung, -analyse und Programmoptimierung ist ein zyklischer Prozess
- Verschiedene Werkzeugtypen stehen zur Auswahl: Offline- und Online-Werkzeuge
- Ziel ist die automatische Erkennung und Beseitigung von Leistungsengpässen
- Vampir ist eines der leistungsfähigsten Offline-Werkzeuge
- Insbesondere Optimierung der Kommunikation ist wichtig für die Programmeffizienz
- Die Messmethodik ist wichtig, um sinnvolle Ergebnisse zu erhalten



## **Leistungsoptimierung**

### Die wichtigsten Fragen

- Wie ist die Vorgehensweise bei der Programmoptimierung?
- Was sind die Vor- und Nachteile der Offline- und Online-Werkzeuge?
- Welches Verfahren eignet sich wozu?
- Welche primitiven Hilfsmittel gibt es?
- Nennen Sie Beispiele für Werkzeuge und deren Funktionalitäten
- Wie optimiert man ein Programm?
- Wieso ist die korrekte Messmethodik wichtig?
- Welche Fehlertypen gibt es?
- Was kann man gegen diese unternehmen?

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

757



# Fehlertoleranz

1. Einführung
2. Begriffsbildung
3. Realisierungstechniken
  - Statische
  - Dynamische
4. Quantitative Bewertung
5. Systeme in der Praxis

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

758

Siehe: [https://en.wikipedia.org/wiki/Fault\\_tolerance](https://en.wikipedia.org/wiki/Fault_tolerance)

# 1. Einführung

Warum „Fehlertoleranz“ als Thema?

Vorlesungsstunde hätte auch „Ausfallsicherheit“ oder „Hochverfügbarkeit“ heißen können

„Fehlertoleranz“ ist der allgemeine Mechanismus, mit dem Ausfallsicherheit und Hochverfügbarkeit erzielt wird

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

759

Siehe: [https://en.wikipedia.org/wiki/High-availability\\_cluster](https://en.wikipedia.org/wiki/High-availability_cluster)

# Motivation

## Wozu Fehlertoleranzmechanismen?

- Ausfälle im System werden verdeckt und das System läuft mit kurzer Unterbrechung oder mit verminderter Leistung weiter
- Schutz vor Verlust von Menschenleben und/oder Sachwerten

## Warum auf Parallelrechnern / in Clustern?

- Redundante Hardware-Strukturen eignen sich besonders gut zur Fehlererkennung und dynamischen Fehlerverdeckung

## **Warum** wirklich auf Parallelrechnern / in Clustern?

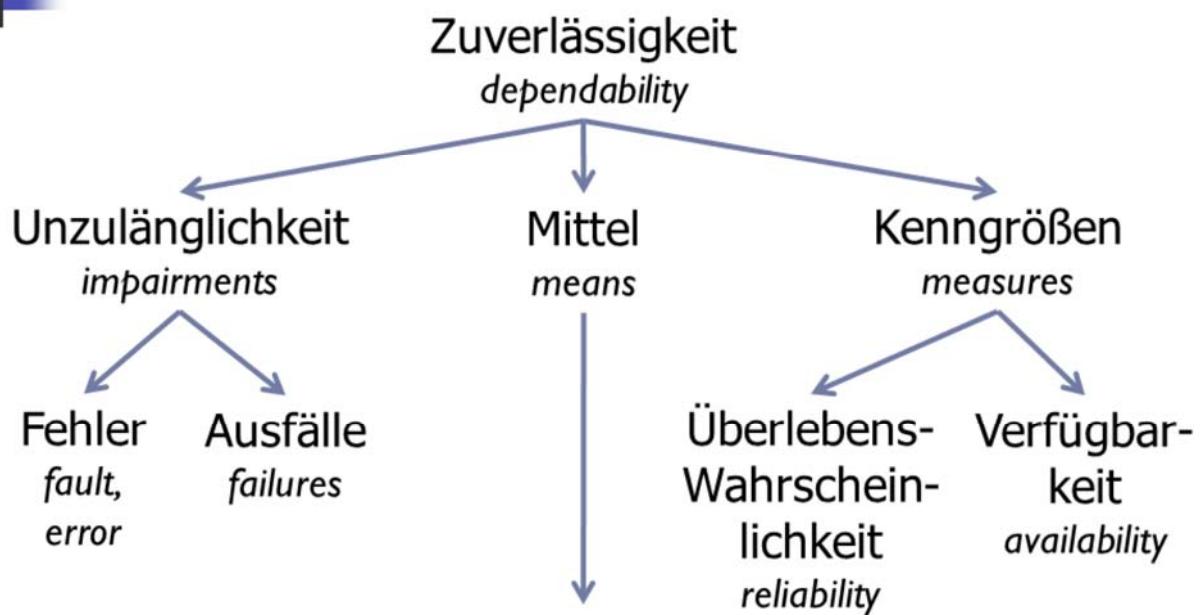
- Redundante Hardware-Strukturen fallen ständig aus ☹

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

760

## 2. Begriffsbildung



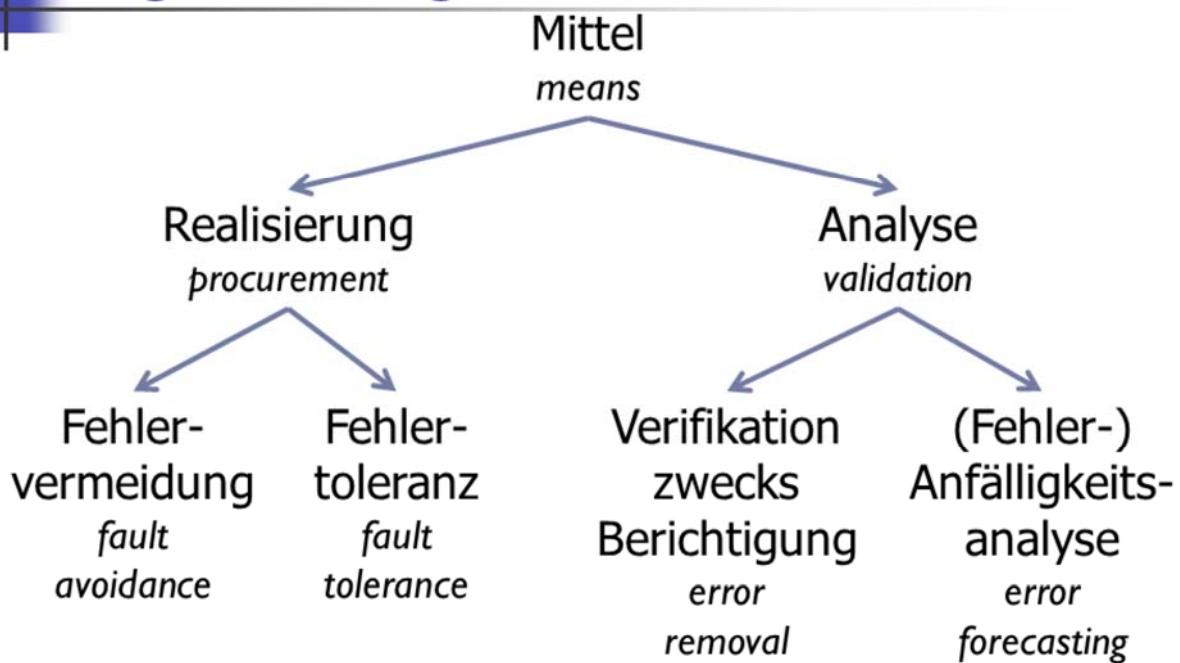
12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

761

Begriffsbildung nach Belli, Echtle, Görke.

## Begriffsbildung...



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

762

Begriffsbildung nach Belli, Echtle, Görke.

# Zuverlässigkeit

## Definition

Fähigkeit eines Systems, die spezifizierte Anwendungsfunktion während eines Einsatzzeitraumes zu erbringen

## Kenngrößen

- Überlebenswahrscheinlichkeit
- Verfügbarkeit

# Zuverlässigkeit...

## Überlebenswahrscheinlichkeit

- Wahrscheinlichkeit  $R(t)$  dafür, dass das System im Zeitintervall  $[0;t]$  fehlerfrei bleibt, wenn es zu  $t=0$  intakt war  
[System ohne Reparatur]

## Verfügbarkeit

- Wahrscheinlichkeit  $A(t)$  dafür, dass das System zum Zeitpunkt  $t$  intakt ist  
[System mit Reparatur]
- $A(t) = \text{MTBF} / (\text{MTBF} + \text{MTTR})$   
MTBF: mean time between failures  
MTTR: mean time to repair

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

764

Siehe:

- [https://en.wikipedia.org/wiki/Reliability\\_%28engineering%29](https://en.wikipedia.org/wiki/Reliability_%28engineering%29)
- <https://en.wikipedia.org/wiki/Availability>

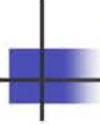
Beachten Sie, dass auch bei 99% Verfügbarkeit ein System untauglich sein kann, wenn die MTBF nicht viel höher als die durchschnittliche Joblänge ist. Die hohe Verfügbarkeit kann von einer sehr kleinen MTTR herrühren.

## Wichtige Kennzahlen der Verfügbarkeit

Man spricht z.B. von den fünf Neunen, gemeint ist  
0,99999 % Verfügbarkeit

Was bedeutet das als Ausfallzeit im Jahr?

| Verfügbarkeit | Ausfallzeit pro Jahr |
|---------------|----------------------|
| 0,9           | 876 Stunden          |
| 0,99          | 87 Stunden           |
| 0,999         | 9 Stunden            |
| 0,9999        | 52 Minuten           |
| 0,99999       | 5 Minuten            |



## Unzulänglichkeiten

### Beeinträchtigung der Zuverlässigkeit

- Fehler
  - Unerwünschter (nicht die Spezifikation erfüllender) Zustand des Systems  
(unterscheide noch Fehlerursache/-zustand)
- Ausfall
  - Ist das Ereignis, dass eine benötigte Funktion nicht erbracht wird

# Fehler

Einteilung nach ihrem Entstehen im Lebenszyklus des Systems

- Entwurfsfehler
- Herstellungsfehler
- Betriebsfehler

Einteilung nach der Zeitdauer

- Permanent
- Transient
- Intermittierend (pseudotransient)

# Fehler...

## Entwurfsfehler

Vor Inbetriebnahme des Systems

- Spezifikationsfehler
- Implementierungsfehler
- Dokumentierungsfehler

## Herstellungsfehler

System entspricht nicht der entworfenen Implementierung

- Z.B. fehlerhafte Materialien oder Werkzeuge
- Bei uns: Fehler in Compilern und Bibliotheken

# Fehler...

## Betriebsfehler

In der Nutzungsphase nach Inbetriebnahme

- Zufällige physikalische Fehler
- Verschleißfehler
- Störungsbedingte Fehler
- Bedienungsfehler
- Wartungsfehler
- Absichtliche Fehler (z.B. Sabotage)

Softwarefehler meist Entwurfs- oder Herstellungsfehler

## Fehlermodell

Aufgrund der Vielzahl möglicher Fehler:

- Einschränkung auf bestimmte Gruppen
- Fehlermodell nennt betroffene Subsysteme und beschreibt Fehlverhalten
- Damit dann analytische Studien möglich

Z.B. Fehlermodell für Rechner-Cluster

- Nur permanente Fehler in Knoten, Leitung, Switches. Zusätzlich evtl. Prozessoren, Speichermodule, Festplatten
- Fehlerzustände: intakt, defekt
- Systemzustand dargestellt durch Vektor

## 3. Realisierungstechniken

Methoden zur Erhöhung der Zuverlässigkeit

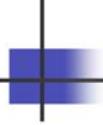
Ergänzen sich gegenseitig:

- Fehlervermeidung
- Fehlertoleranz

Fehlervermeidung (Fehlerintoleranz)

- Sorgfältige Konstruktion
- Ausführliche Tests

Nicht vermeidbare Fehler versuchen zu tolerieren



## Fehlertoleranz

Einsatz von Redundanz, um im Fehlerfall weiterarbeiten zu können

- Redundanz
  - HW-Redundanz (zusätzliche Bauteile)
  - SW-Redundanz (zusätzliche Programme)
  - Zeit-Redundanz (zusätzlicher Zeitaufwand)
- Aktivierung der Redundanz
  - Statische Redundanz (funktionsbeteiligte Redundanz)
  - Dynamische Redundanz (Reserveredundanz)

Siehe: [https://en.wikipedia.org/wiki/Redundancy\\_%28engineering%29](https://en.wikipedia.org/wiki/Redundancy_%28engineering%29)

# Fehlertoleranz...

## Statische Redundanz

- Ressourcen ständig in Betrieb
- Im Fehlerfall direkter Zugriff auf diese Einheiten
- Z.B. Hamming Codes / Triple Modular Redundancy

## Dynamische Redundanz

- Aktivierung erst im Fehlerfall
- Bis zu diesem Zeitpunkt:
  - Ungenutzte Redundanz (Standby-Systeme)
  - Fremdgenutzte Redundanz (mit Verdrängung)
  - Gegenseitig nutzbare Redundanz (Fail-Soft-Systeme)

# Fehlertoleranztechniken

## Tolerierung verschiedener Fehlerarten

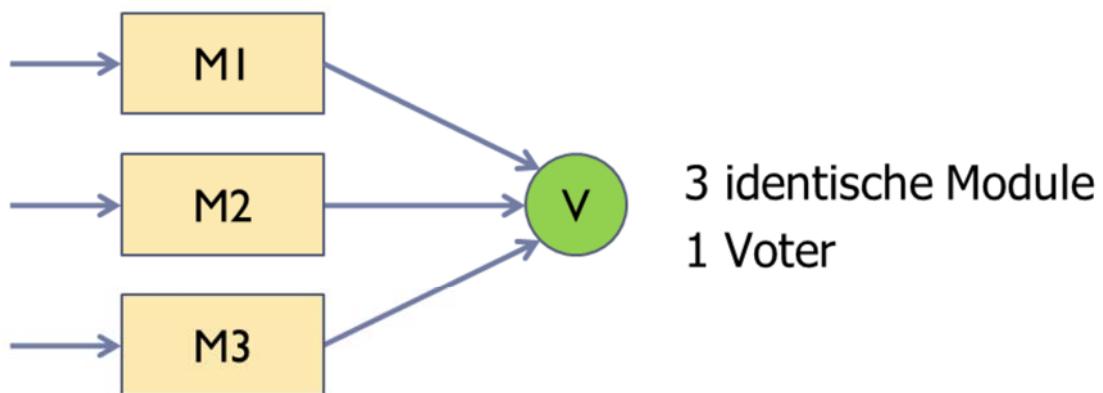
- HW-Fehler (siehe folgende Folien)
  - SW-Fehler
- N-Versionen-Programmierung (gut für Mehrprozessorsysteme)

## Betrachtungsebene

- Hier: Prozessoren, Speicher, Verbindungen
- Tiefere Ebenen auch möglich: z.B. fehlerkorrigierende Codes

## Statische Redundanz

Wichtigste Technik: Triple Modular Redundancy (TMR)

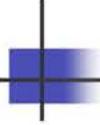


12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

775

Siehe: [https://en.wikipedia.org/wiki/Triple\\_modular\\_redundancy](https://en.wikipedia.org/wiki/Triple_modular_redundancy)



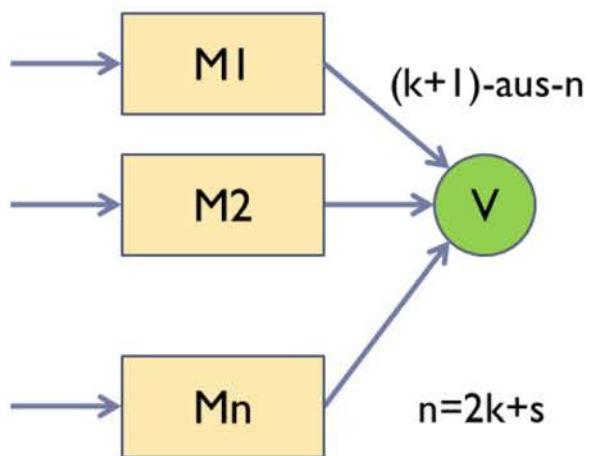
## Statische Redundanz...

### Konzept TMR

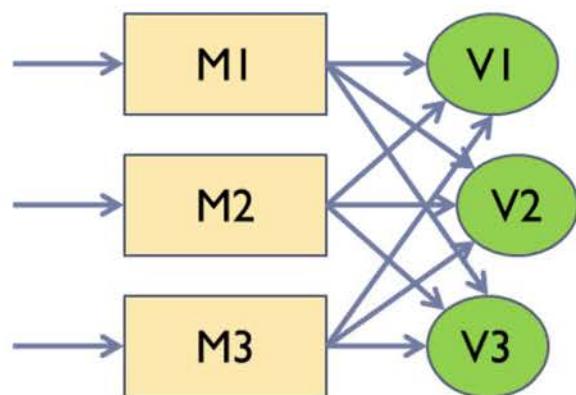
- Drei identische Module führen nebenläufig gleiche Funktion aus (kann HW oder SW sein)
- Voter fällt 2-aus-3-Mehrheitsentscheid
- Erster Ausfall kann toleriert werden
- Zweiter Ausfall kann noch erkannt werden
- Voter kann per Hardware oder Software realisiert werden

## Statische Redundanz...

### N-Modular-Redundancy (NMR)



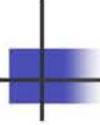
### Tolerierung von Voter-Ausfällen



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

777



## Statische Redundanz...

### Vorteile TMR/NMR

- Toleriert permanente und transiente Fehler
- Fehlerbehebung kostet keine Zeit

### Nachteile

- Hoher Aufwand in HW oder SW begrenzt Einsatzbereich

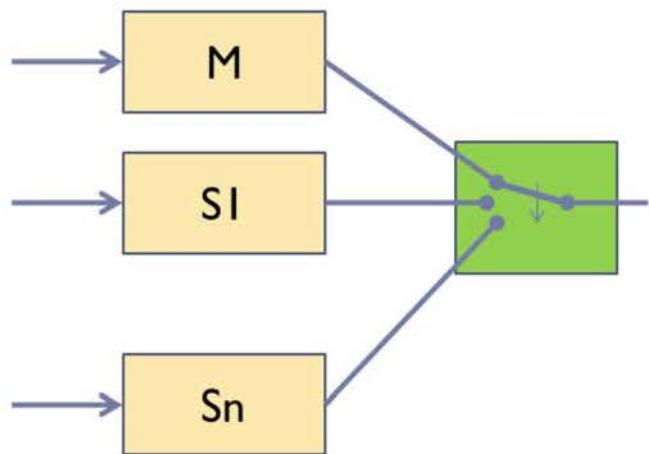
# Dynamische Redundanz

## Standby-Systeme

Zusätzlich zu den Systemkomponenten noch Reservekomponenten

### Nachteile:

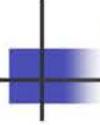
- Kein verzögerungsfreies Umschalten
- Ungenutzte Ressourcen (cold/hot standby)



## Dynamische Redundanz...

### Fail-Soft-Systeme (Graceful Degradation)

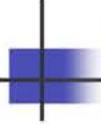
- Mehrfach vorhandene Subsysteme werden als Redundanzen genutzt (Eigenredundanz)
- Defekte Systemkomponenten werden deaktiviert
- Mit verminderter Leistung kann die Aufgabe weiterbearbeitet werden
  
- Ausgangsbedingungen und Verfahren durch ein Rechner-Cluster optimal abgedeckt



## Dynamische Redundanz...

### Vorgehensweise bei beiden Verfahren

- Fehlererkennung und –lokalisierung (Fehlerdiagnose)
- Rekonfiguration
- Fehlerbehandlung (Recovery)



## Dynamische Redundanz...

### (1) Fehlerdiagnose

- Aufgabe: Erkennen defekter Knoten
- Ebene: Komponenten- und Systemebene
- SRU (Smallest Replaceable Unit)
  - Ebene der Rekonfiguration
  - Keine Lokalisation auf Komponentenebene
- Fehlererkennung auf Rechnerknotenebene
  - Fehlererkennende Codes, Selbsttestprogramme
- Zur Rekonfiguration noch Diagnose auf Systemebene

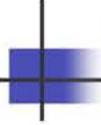
## Dynamische Redundanz...

- Systemdiagnose
  - Fremddiagnose
    - Separater Prozessor für Diagnose (und i.a. auch für Rekonfiguration und Recovery)
    - Nachteile
      - Diagnose- und Wartungsprozessor als perfekt zuverlässig angenommen
      - Zusätzliche Prozessortypen (Heterogenität)
  - Eigendiagnose
    - Zentral: Testrunde in bestimmten zeitlichen Abständen Koordinatoreinheit ermittelt defekte Komponenten  
Problem: Wahl des Koordinatorknotens
    - Dezentral: Alle intakten Einheiten haben Diagnosebild  
Problem: Konsistenz der Einzelbilder

# Dynamische Redundanz...

## Allgemeines Steuerungsproblem: Zentral vs. dezentral

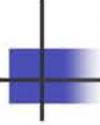
- Zentral
  - Vorteile  
Konsistente Systemsicht (wichtig!)
  - Nachteile  
Gefährlich! Sogenannter Single-Point-of-Failure  
Evtl. Überlastung der Kommunikationswege hin zur Zentrale
- Dezentral
  - Vorteile  
Kein Single-Point-of-Failure  
Gut im System verteilbare Last
  - Nachteil  
Konsistente Systemsicht beliebig schwierig  
Nutze „Verteilte Algorithmen“ (siehe z.B. Arbeiten von Leslie Lamport)



## Dynamische Redundanz...

### (2) Rekonfiguration

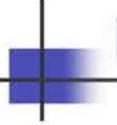
- Bildet aus den intakten Einheiten ein lauffähiges System
- Ausführung
  - Externer Wartungsprozessor
  - Zentraler Koordinator (hier unkritisch)
  - Dezentral
- Schwierigkeit
  - Suche einer neuen effizienten Abbildung der SW-Komponenten auf die HW-Komponenten



## Dynamische Redundanz...

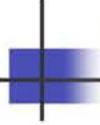
### (3) Fehlerbehebung

- Versetzt das System in einen korrekten Zustand
- Setzt die Anwendung mit korrekten Daten wieder auf (Wiederanlauf)
- Methoden
  - Rückwärtsfehlerbehebung
  - Vorwärtsfehlerbehebung



## Dynamische Redundanz...

- Rückwärtsfehlerbehebung
  - System zurücksetzen in früheren konsistenten Zustand (*rollback*)
  - Rücksetzpunkte (*recovery points*)
  - Abspeicherung in Haupt- oder Hintergrundspeicher
  - Zeitintervalle durch analytische Methoden
- Vorwärtsfehlerbehebung
  - Zusätzliche Operationen bringen das System in einen korrekten Zustand
  - Problem: Genaue Kenntnis der Anwendung notwendig



## Dynamische Redundanz...

### Bewertung

- Bei dynamischer Fehlertoleranz kein unterbrechungsfreier Betrieb
  - Für „normale“ Anwendungen akzeptabel, nicht jedoch bei Echtzeitanwendungen
- Je nach Variante können aber alle Komponenten produktiv genutzt werden

## 4. Quantitative Bewertung

Überlebenswahrscheinlichkeit  $R(t)$

Exponentiell verteilte Lebensdauer  $R(t) = e^{-\lambda t}$

MTTF (mean time to failure) =  $1/\lambda$

Zunächst das System ohne Fehlertoleranz:

$$R(t)p = e^{-p\lambda t}$$

Anzahl  $p$  PMU's (*processor memory unit*)

## Quantitative Bewertung...



[R]

1.0

0.5

0.0

p=512

p=256

p=16

p=32

p=64

p=128

[ $\lambda t$ ]

0.001

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

790

0,001 auf der x-Achse entspricht einem Promille der durchschnittlichen Lebensdauer.

## Quantitative Bewertung...

[R]

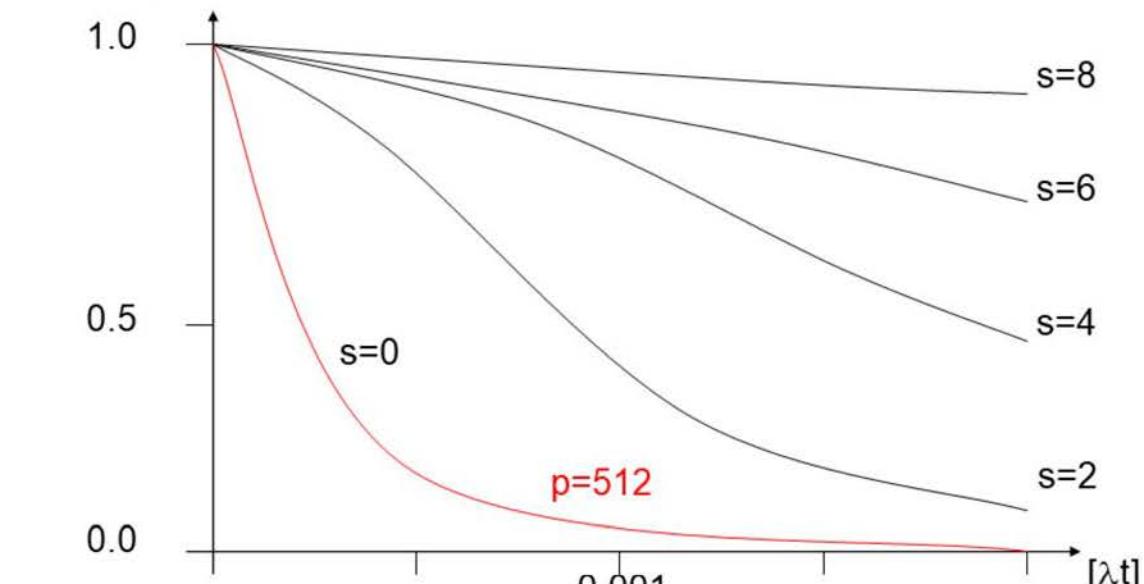
1.0

0.5

0.0

12.10.2021 Hochleistungsrechnen - © Thomas Ludwig

791



## Quantitative Bewertung...

System kann bis zu s ausgefallene PMU's tolerieren:

$$R_{(p-s) \text{ aus } p}(t) = \sum_{i=0}^s c^i (1-R(t))^i R(t)^{p-i}$$

$c = P(\text{Fehler wird behandelt} \mid \text{Fehler tritt auf})$   
*(coverage factor - Überdeckungsfaktor)*

## 5. Systeme in der Praxis

### Fehlertolerierende Speichersysteme

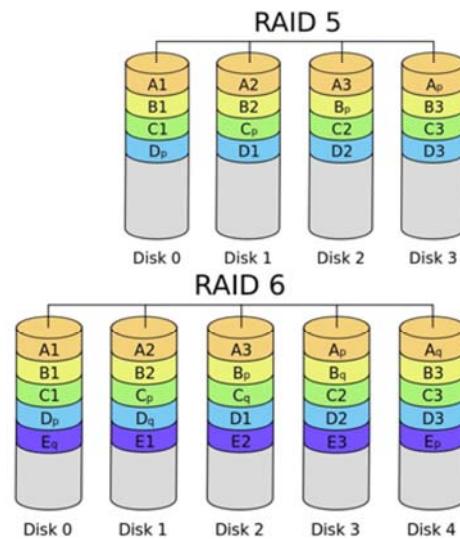
- Error correcting code memory (ECC memory)
  - Ursachen für Bitfehler: z.B. elektrische/magnetische Einstrahlungen
  - Folgen: Veränderung von Code oder Daten im Hauptspeicher
  - Schutzmechanismus
    - Verwende 9. Bit
    - Nutze Hamming-Codes (statische Redundanz)
  - Problem
    - Erhöht HW-Kosten und Stromverbrauch um mindestens 1/8



## Systeme in der Praxis...

### ■ RAID5/6 in Festplattensystemen

- Probleme: Veränderungen einzelner Bit und Ausfälle von Platten
- Folgen: Datenverlust droht
- Schutzmechanismen
  - Verwende zusätzliche Platten
  - Nutze Paritätsinformation (statische Redundanz)
  - Nach Plattendefekte ersetze defekte Platte und rekonstruiere Inhalt
- Probleme
  - Erhöht HW- und Betriebskosten
  - Geringere Leistung während Plattenrekonstruktion
  - Leistungseinbußen im Betrieb



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

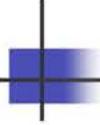
794

RAID 5: einfache Redundanz – eine Platte darf ausfallen, ohne dass man Daten verliert. RAID 6: doppelte Redundanz – zwei Platten dürfen ausfallen, ohne dass man Daten verliert.

# Systeme in der Praxis...

## Fehlertolerierende Systeme in der Praxis

- Parallelrechner und Hochleistungsrechnen
  - Nahezu keine Realisierungen
  - Gründe
    - Systeme zu aufwendig
    - Nutzen zu gering, da Schäden zu gering
  - Ausnahme: Deutscher Wetterdienst
- Parallelrechner und kommerzielle Anwendungen
  - High-Availability-Computing
    - Hochverfügbare Systeme
    - Normalerweise keine parallelen Programme
    - Bereich Datenbanken, Systemsteuerungen u.ä.



## Systeme in der Praxis...

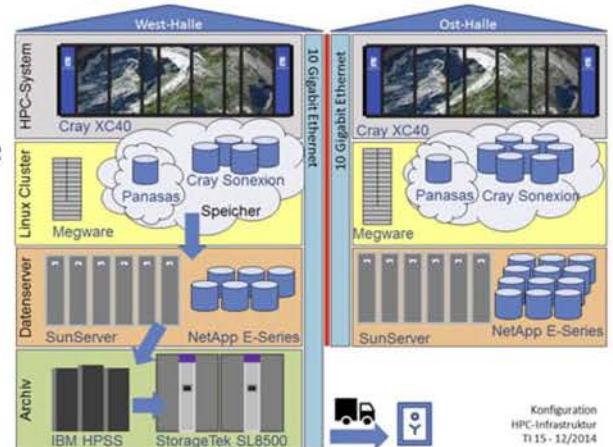
### DKRZ

- Plattensystem Mistral ca. 1% Ausfälle pro Jahr
  - Bei 10.000 Platten ca. 100 Ausfälle
- Rechnerknoten ca. 5 Ausfälle am Tag
  - Teilweise nur Reboot
  - Durch Skripte vor Zuteilung durch Scheduler erkannt

# Systeme in der Praxis...

## Systeme in der Praxis: Deutscher Wetterdienst

- Anbindung an Verkehrsministerium  
Verantwortet Wettervorhersage Flugverkehr
- Rechner- und Speicher-  
system verdoppelt
- Nutzung des Erstsysteams
  - Operationelle Wettervorhersage
- Nutzung des Zweitsystems
  - Meteorologische Forschung
- Dynamische Redundanz
  - Umschaltung (Failover) im Fehlerfall
- Problem: Kosten



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

797



## Umsetzungskonzepte

### Verfügbare Software-Unterstützung

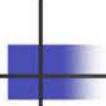
- High-Availability (HA) Linux Project
  - Definition von Verfahren, um Parallelrechner ausfallsicher zu machen
  - Cluster hier: Menge von HW-Komponenten, die als wechselseitige Redundanz dienen
- Linux High Availability HOWTO
- Viele kommerzielle HA-Cluster und HA-Lösungen

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

798

Siehe: <http://www.linux-ha.org/>



## Fehlertoleranz

### Zusammenfassung

- Fehlertoleranz ist ein wichtiges Thema bei allen Systemen, die aus sehr vielen Einzelkomponenten bestehen
- Wir unterscheiden im allgemeinen Systeme mit und ohne Reparatur
- Wir unterscheiden verschiedene Klassen von Fehlern
- Ein Fehlermodell beschreibt das System analytisch
- Redundanz ist notwendig, um Fehler tolerierbar zu machen
- Wir unterscheiden statische und dynamische Redundanzverfahren
- Statische Redundanz: Triple-Modular-Redundancy-Verfahren
- Dynamische Redundanz: Fail-Soft-Verfahren
- In der Praxis Anwendungen nur im Bereich des sogenannten High-Availability-Computing



## Fehlertoleranz

### Die wichtigsten Fragen

- Wozu Fehlertoleranz?
- Warum sind Parallelrechner in diesem Bereich wichtige Architekturtypen
- Welche Kenngrößen werden verwendet?
- Welche Fehlerkategorien gibt es?
- Was ist statische Redundanz?
- Welches sind hierbei die typischen Konzepte?
- Was ist dynamische Redundanz?
- Welche Durchführungsphasen finden wir hierbei?
- Wie sieht die qualitative Bewertung eines Systems ohne Reparatur aus?
- Welche Umsetzung gibt es für Linux-Systeme?

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

800



# **Grid- und Cloud-Computing**

1. Grid-Computing
  - a) Motivation
  - b) Grid-Computing
  - c) Anwendungsbereiche
  - d) Benutzung / Dienste
  - e) Projekte
2. Cloud-Computing
  - a) Varianten
  - b) Cloud-Dienstanbieter

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

801

# 1. Grid-Computing

## Ausgangssituation

- Viele Cluster und Parallelrechner in verteilten Rechenzentren in der ganzen Welt
- Viel Knowhow in unterschiedlichen Gruppen verteilt

## Aber:

- Ressourcen sind oft nicht da, wo die Benutzer sind
- Das Wissen ist nicht da, wo der Benutzer ist

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

## a) Motivation

Situation bei Clustern und Parallelrechnern

- Verschiedene Hardware-Architekturen
- Verschiedene Hersteller
- Verschiedene Betriebssysteme und Basis-SW
- Verschiedene Benutzungsmodelle (Stapelbetrieb, interaktiv)
- Verschiedene Formen der Systemverwaltung
- Verschiedene Formen der Datenverwaltung
- Verschiedene Sicherheitsmechanismen

## Motivation...

### Konsequenzen für den Benutzer

- Arbeitet nur mit vertrautem Rechner
- Benutzt nur Rechner mit passenden Ressourcen
- Vermeidet neue Rechner

### Konsequenzen für den Systembetreiber

- Teure Ressourcen nicht optimal genutzt
- Probleme nicht optimal gelöst
- Lösbarer Probleme zum Teil gar nicht gelöst

## Motivation...

### Neue Benutzungsmethodik

- Wir betrachten das Cluster, den Parallelrechner oder das Rechenzentrum als „einzelnen“ Rechner
- Zusammenfassen mehrerer solcher „Rechner“ zu einem neuen „Rechnersystem“
- Programme werden auf diesem „Rechnersystem“ ausgeführt

### „Rechnersystem“

- Früher vernetzte Einzelrechner
- Heute vernetzte Hochleistungsrechner

## Motivation...

Nutzungsmethode der neuen „Rechnersysteme“ in Analogie zu früher

- Zuweisung eines Jobs zu einer freien Ressource  
Einzeljob nicht parallel bzgl. neuen Rechnersystems  
Bis ca. 1999 genannt: Internetcomputing  
Frühes Beispiel: seti@home
- Parallelisierung eines Jobs über mehrere Ressourcen  
Paralleler Job bzgl. neuen Rechnersystems  
Bis ca. 1999 genannt: Metacomputing



## Motivation...

### Nutzungsmethode...

- Hardware- und Software-Konzepte  
Vernetzung, Sicherheit, Ein-/Ausgabe usw.
- Programmkonstruktion  
Individuelle Jobs für Einzelsysteme oder Verbundsysteme  
(aber intern alles parallelisiert)
- Nutzungsmethodik  
Stapelbetrieb und/oder interaktiver Betrieb

## Motivation...

Alles schon bekannt, aber jetzt eine Abstraktionsstufe höher

- Ressourcenverwaltung, Internetcomputing  
Die einzelne Anwendung ist parallelisiert und läuft an einem auswählbaren Ort  
Anbindung über Web-Frontend
- Metacomputing  
Die Einzelanwendung ist parallelisiert für verschiedene parallele Umgebungen und läuft parallel an verschiedenen Orten

Neuer Begriff seit 1998: GRID-Computing

# Eine wichtige Ressource

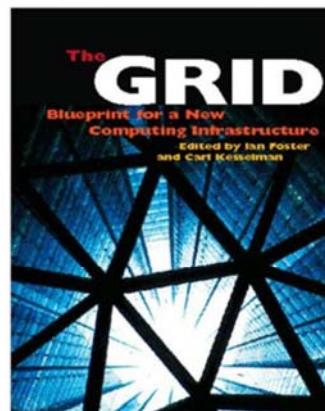


## The Grid: Blueprint for a New Computing Infrastructure

I. Foster, C. Kesselman (Eds), Morgan Kaufmann, 1999

- Available July 1998;
- ISBN 1-55860-475-8
- 22 chapters by expert authors including Andrew Chien, Jack Dongarra, Tom DeFanti, Andrew Grimshaw, Roch Guerin, Ken Kennedy, Paul Messina, Cliff Neuman, Jon Postel, Larry Smarr, Rick Stevens, and many others

*"A source book for the history of the future"* -- Vint Cerf



<https://slideplayer.com/slide/6979534/>

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

809

Siehe:

- <https://slideplayer.com/slide/6979534/>
- [https://en.wikipedia.org/wiki/Vint\\_Cerf](https://en.wikipedia.org/wiki/Vint_Cerf)



## b) Grid-Computing

### Idee

„Rechenleistung an jedem Ort“

Analogie zum „*power grid*“, dem Stromnetz

Einspeisung an verschiedene Stellen, Nutzung an beliebigen anderen Stellen

- Begriff des „Grid“ mittlerweile sehr umfassend
- Kein Konsens über Anwendung und Konzepte



## Grid-Computing...

Warum kam das gerade damals auf?

- Zugriff auf Ressourcen erwünscht, die lokal nicht vorhanden sind; Replikation zu teuer
- Vernetzung immer leistungsfähiger und gleichzeitig immer billiger
- Erfahrung mit dem parallelen Rechnen brachte Erkenntnis zum Thema Verteilung von Programmen



# Grid-Computing...

## Fragestellungen

- Auswahl der Rechnerressourcen
- Auswahl der Parallelisierung
- Auswahl der Sprachkonzepte
- Sicherheitsaspekte
- Lastausgleich
- Fehlertoleranz

Insgesamt nichts neues!

Jetzt alles in großen Dimensionen betrachtet



## Grid-Computing...

### Die Ziele

- Weltweite Nutzung freier Rechenleistung
- Aggregation hoher Rechenleistung
- Einfacher Zugriff auf Rechenleistung
- Bewältigung großer Aufgabenstellungen
- Kollaboration zwischen Orten

# Grid-Computing...

## Voraussetzungen

- Hochgeschwindigkeitsvernetzung  
Gigabit-Vernetzung  
Satellitenkopplung
- Geeignete Protokolle auf verschiedenen Ebenen

## Problem

- Kommunikationslatenz  
Beim Grid-Computing deutlich höher, d.h. nur grobgranulare Parallelisierung gewinnbringend

## c) Anwendungsbereiche

### Vier wichtige Gebiete

- *Desktop Supercomputing*  
Zugang zu hoher Rechenleistung von überall aus
- *Smart Instruments*  
Verbindet Radioteleskope, Kernbeschleuniger, Tomographen usw. mit geeigneten Hochleistungsrechnern
- *Collaborative Environments*  
Verbindet Benutzungsumgebungen miteinander und mit Supercomputern zu Simulationsläufen usw.
- *Distributed Supercomputing*  
Erzielt Summe der Leistungen der Einzelrechner

## Umgebungscharakteristika

- Wachsende Systemgrößen  
Jedoch meist nur noch Teilsysteme verwendet
- Heterogenität auf allen Ebenen  
HW, Betriebssysteme, Sprachen, Compiler usw.
- Wechselnde Strukturen  
Sowohl des Rechnersystems als auch des Programmsystems
- Dynamisches Verhalten  
Gemeinsam genutzte Ressourcen schwer kontrollierbar
- Verschiedene Administrationsprogramme  
Authentifizierung, Autorisierung usw.

## d) Benutzung / Dienste

Das Grid ist nicht nur eine Ansammlung von Ressourcen sondern auch von Diensten

- Scheduling
- Ressourcen-Verwaltung
- Sicherungspunkt-Verwaltung
- Sicherheit und Abrechnung
- ...

# Benutzung / Dienste...

## Scheduler

- Job-Scheduler
  - Optimierung auf Job-Durchsatz
- Ressourcen-Scheduler
  - Optimierung auf beste Ressourcen-Nutzung
- Anwendungs-Scheduler
  - Optimierung auf z.B. minimale Ausführungszeit

## Fragestellungen

- Die üblichen:  
Was soll wann wo ausgeführt werden?



## Benutzung / Dienste...

Im Grid heterogene Leistungs-Charakteristik

- Software, Hardware, Vernetzung
- Mitbenutzung durch andere Nutzer
- Keine zentrale Kontrolle über alle Ressourcen möglich
- Verfügbare Ressourcen sind dynamisch
- Leistung der Anwendung  $\neq$  Leistung des Systems

Scheduling hier maximal schwer

- Nur heuristische Verfahren sinnvoll

# Benutzung / Dienste...

## Sicherheit und Abrechnung

- Bekannte Mechanismen auf höherer Ebene
  - Kerberos
  - Public-Key-Systeme
  - Zertifikate
  - Firewalls
- Zusätzliche Probleme
  - Sicherheitsvereinbarungen zwischen Organisationen
  - Verteilter Zugang zum System
  - Verteilte Abrechnung

# Infrastrukturen

## Wichtigste Infrastruktur

- Globus Toolkit (Argonne National Laboratory)



Globus-Alliance: [www.globus.org](http://www.globus.org)

## Das Konzept

- Globus stellt ein Toolkit zur Verfügung, das Basismechanismen bereitstellt (Kommunikation, Autorisierung usw.)
- Hierauf lassen sich höhere Dienste des Grid-Computing abstützen (Programmierwerkzeuge, Scheduler usw.)
- Langfristziel: AWARE (*Adaptive Wide Area Resource Environment*)

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

821

Siehe: <https://www.globus.org/>

## e) Projekte: D-Grid

The screenshot shows the homepage of the D-Grid Initiative. At the top, there's a navigation bar with links for Startseite, News, Veranstaltungen, Kontakt, Sitemap, and language switches (German and English). Below the navigation is a banner featuring a map of Germany and the text "D-GRID Initiative". The main content area is divided into several sections:

- Startseite**: A sidebar with links to Startseite, Über D-Grid, Integrationsprojekt, Projektübersicht, D-Grid im eScience, Service, and Veranstaltungen.
- User-Portal** and **Provider-Portal**: Buttons for user and provider access.
- Suche/Search**: A search bar.
- GEFÖRDERT VON:** A logo for the Bundesministerium für Bildung und Forschung.
- Die Deutsche Grid-Initiative (D-Grid)**: A section describing the initiative as a joint effort by science and industry to build the D-Grids. It mentions the first projects started in September 2005 and the involvement of the Federal Ministry for Education and Research (BMBF).
- D-Grid 1, 2005-2008**: A section about IT services for scientists developed by experienced grid researchers and users. It highlights applications in high-energy physics, astrophysics, alternative energy, medicine, climate research, engineering, and philosophy.
- D-Grid 2, 2007-2010**: A section about IT services for science and industry built on the D-Grid integration layer, including examples from construction, finance, automotive, aerospace, and geospatial data processing.
- Veranstaltungen**: A section listing events like AHM III 2010 in Dresden, which brought together various grid projects for exchange and further development.
- Weitere Veranstaltungen**: Information on other events and workshops through the calendar.
- Aktuelles**: A news section dated 30.11.2009.

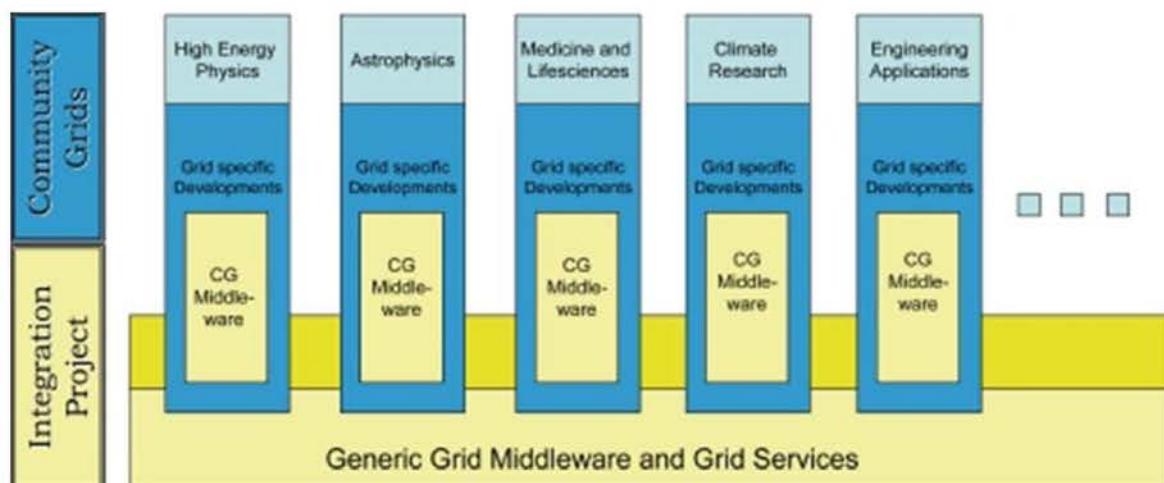
At the bottom left is the date 12.10.2021, and at the bottom right is the page number 822.

Hochleistungsrechnen - © Thomas Ludwig

822

# Deutsche Grid-Initiative (D-Grid)...

## Integration Project + Community Grids



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

823

# Gauss Centre for Supercomputing

The screenshot shows the homepage of the Gauss Centre for Supercomputing (GCS). The header features the GCS logo and a photograph of a supercomputer rack. A sidebar on the left contains a navigation menu with links to Homepage, About GCS, Resources, Science, Publications, Events, Links, News, and Imprint. The main content area has a blue header with the text "Supercomputing at the Leading Edge" and "A Key Technology for Science and Engineering". Below this, a paragraph states: "The Gauss Centre for Supercomputing (GCS) provides the most powerful high-performance computing infrastructure in Europe." Another paragraph notes: "The GCS is the alliance of the three German national supercomputing centres:". Three logos are displayed below: NIC (John von Neumann-Institut für Computing Jülich), LRZ (Leibniz-Rechenzentrum der Bayerischen Akademie der Wissenschaften Garching), and HLRS (Höchstleistungsrechenzentrum Stuttgart).

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

824

Siehe: [http://www.gauss-centre.eu/gauss-centre/EN/Home/home\\_node.html](http://www.gauss-centre.eu/gauss-centre/EN/Home/home_node.html)

Siehe: <http://www.prace-project.eu/>

Nicht notwendigerweise Grid-Computing, aber vermutlich eng damit verbunden.

Siehe auch: [https://en.wikipedia.org/wiki/European\\_Grid\\_Infrastructure](https://en.wikipedia.org/wiki/European_Grid_Infrastructure)

## 2. Cloud-Computing

Weltweit existieren sehr viele Rechner- und Speichersysteme und auf ihnen viele Softwaresysteme

Nutzer an beliebigen Orten könnten vielleicht über eine Vernetzung auf diese Ressourcen zugreifen

Hätte für Anbieter und Nutzer einige Vorteile

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

826

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

## Cloud-Computing...

„Cloud-Computing“ = Rechnen in der Wolke

Wolke? Abgeleitet vom Wolkensymbol für das Internet



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

827

Quelle: [http://commons.wikimedia.org/wiki/File:Cloud\\_computing.svg](http://commons.wikimedia.org/wiki/File:Cloud_computing.svg)

## b) Varianten

Konzepte und Systeme werden über das Netz zur Verfügung gestellt

Bezahlung je nach Nutzung

- Infrastructure as a Service (IaaS)
  - Rechenkapazität, Speicherkapazität bereitstellen
- Software as a Service (SaaS)
  - Nutzung fertiger Programmpakete
- Platform as a Service (PaaS)
  - Programmierumgebungen bereitstellen
- Archive as a Service (AaaS)
  - Archivierungsdienste bereitstellen

# Vorteile / Nachteile

## Vorteile

- Aus Nutzersicht
  - Keine Investitionskosten, Wartungskosten usw.
  - Flexibler Einkauf von Diensten
  - Billige Abwicklung von Lastspitzen
- Aus Betreibersicht
  - Effizientes Anbieten von Diensten
  - Effizientes Verwalten der angebotenen Dienste

## Nachteile

- Aus Nutzersicht
  - Datensicherheit
  - Anbieterbindung

## Abgrenzung

### Grid-Computing

- Gemeinschaftliche Nutzung der gemeinsamen Ressourcen, meist im Bereich des Hochleistungsrechnens
  - Cloud hat wenige einzelne Anbieter und die sind zentral gesteuert

### Peer-to-Peer-Computing

- P2P: Verteilung von Rechenlast auf viele Rechner
- Cloud: Verlagerung von Rechenlast auf andere Rechner

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

830

Siehe: <https://en.wikipedia.org/wiki/Peer-to-peer>

## b) Cloud-Dienstanbieter



[Bei AWS Management Console anmelden](#) [Legen S](#)

AWS

Produkte

Entwickler

Community

Sup

Produkte & Dienstleistungen ▾

### Amazon Elastic Compute Cloud (Amazon EC2)

#### Amazon EC2 Details

- [EC2 Überblick](#)
- [Häufig gestellte Fragen](#)
- [EC2 Preisgestaltung](#)
- [Amazon EC2 SLA \(Englisch\)](#)
- [Amazon EC2-Instanztypen](#)
- [Kaufoptionen für EC2-Instanzen](#)
- [Reserved Instances](#)
- [Spot Instances](#)

Amazon Elastic Compute Cloud (Amazon EC2) ist ein Webservice, der die Anpassung der Rechenkapazität in der Cloud ermöglicht. Mit diesem Service wird die Web-Skalierung der Rechenleistung für Entwickler einfacher.

Mit der einfachen Web-Service-Oberfläche von Amazon EC2 können Sie mühelos Kapazität erhalten und konfigurieren. Sie ermöglicht Ihnen die vollständige Kontrolle über Ihre Rechenressourcen sowie die Ausführung auf der bewährten Rechenumgebung von Amazon. Amazon EC2 verringert die zum Erwerben und Booten neuer Server-Instanzen benötigte Zeit auf wenige Minuten. So können Sie die Kapazität entsprechend den Änderungen Ihrer Rechenanforderungen schnell in beide Richtungen skalieren. Indem Sie nur für die Kapazität zahlen, die Sie auch tatsächlich nutzen, verändert Amazon EC2 die wirtschaftlichen Rahmenbedingungen von Rechenoperationen. Amazon EC2 bietet Entwicklern die Tools, um ausfallsichere Anwendungen zu erstellen und diese von üblichen Fehlerszenarien zu isolieren.

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

831

# Cloud und HPC? Cloud, HPC und Cray

Oktober 2017

- Cray XC, CS
- Speichersystem ClusterStor  
bis zu 80 PByte  
bis zu 1,7 TB/s

## Cray supercomputers coming to Azure cloud

New offering is aimed at simulation, modeling, and other HPC tasks.

PETER BRIGHT - 10/23/2017, 6:57 PM



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

832

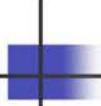
## Cloud und Storage am DKRZ

- OpenStack ist eine Softwarelösung zum Erstellen von privaten und öffentlichen Cloud-Systemen
- Lizenzkostenfreier Open Source Code
- Unterstützung durch die meisten großen IT-Firmen wie Redhat, SUSE, IBM, HP etc.
- Bereits im Einsatz bei vielen Unternehmen und Institutionen wie z.B. CERN, Wikimedia etc.



## Cloud und Storage am DKRZ...

- Software: OpenStack Swift ist ein hochskalierender, ausfallsicherer Objektspeicher mit automatischer Replikation (auch über verteilte Standorte)
- Einsatzgebiete:
  - (globale) Datenverteilung
  - Back-End Storage für Anwendungen wie bspw. iRODS (integrated rule-oriented data system)



## **Grid- und Cloud-Computing**

### Zusammenfassung

- Die heute weltweit verfügbare Rechenleistung soll beim Grid-Computing an verschiedenen Orten nutzbar sein
- Cluster, Parallelrechner, Rechenzentren werden zu einem Grid zusammengefasst
- Wir kennen verschiedene Nutzungsmethoden
- Die Fragestellungen bleiben dieselben, allerdings auf einem anderen Abstraktionsniveau
- Grid-Computing: Rechenleistung aus der Steckdose
- Globus ist eine Infrastruktur in Form einer Sammlung von Diensten
- Die großen Supercomputing-Zentren in Europa formieren sich zu einer Grid-Infrastruktur
- Cloud-Computing stellt verschiedene Dienste bereit
- Cloud-Computing zur Zeit (noch) nicht gut für HPC geeignet

12.10.2021

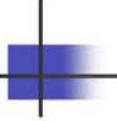
Hochleistungsrechnen - © Thomas Ludwig

835

## **Grid- und Cloud-Computing**

### **Die wichtigsten Fragen**

- Welche prinzipiellen Probleme gibt es bei der Nutzung von Clustern und Parallelrechnern?
- Wie faßt man solche Architekturen zu größeren Einheiten zusammen?
- Was bezeichnete man als Internetcomputing und Metacomputing?
- Was ist die Grundidee des Grid-Computing?
- Welche Fragestellungen finden wir hier?
- Welche Klassen von Anwendungen gibt es hier?
- Was ist die Grundidee des Cloud-Computing?
- Welche Dienste werden angeboten?
- Welche Vor- und Nachteile erkennen wir hier?
- Wie tauglich ist Cloud-Computing für HPC?



# Kosten-Nutzen-Analyse

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

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

837

# 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
- ...

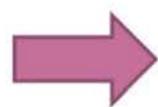
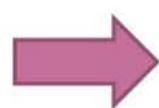
# Kosten in der Petascale-Ära

## 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€
  
- Beinhalten zum Teil das RZ-Gebäude
- Beinhalten zum Teil Stromkosten oder Kosten für ein Kraftwerk

# Kosten in der Petascale-Ära...

## Skalierbare Rechnersysteme



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

840



## 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

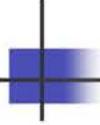
- Zahlreiche Exascale-Programme zum Bau eines Exaflops-Computers mit einem Exabyte-Speichersystem
- USA, Japan, Europa, China, Russland
  - Einige Milliarden Investitionen in F&E

### 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

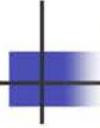
- 8 M€ für Hardware
- 2 M€ für elektrischen Strom
- 4 M€ für Humanressourcen (Brainware)
- 1 M€ für das Gebäude (bei 25 Jahren Abschreibung)
- 0,5 M€ für Magnetbänder
- ...

# 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-baselode 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](#). 847

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 [?](#)

12.10.2021

Hochleistungsrechner - © Thomas Ludwig

849

## 2. Nutzenbetrachtungen

Hochleistungsrechnen erweitert Experiment und Theorie

Numerische Simulation – die Dritte Säule

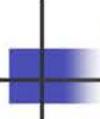
- Numerische Simulation als Mittel der Erkenntnisgewinnung
- Unabdingbar für moderne Wissenschaft und Technik

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 ...

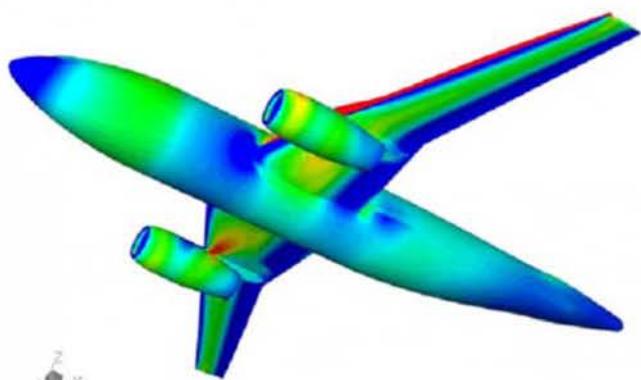


## HPC und Technik

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

## Zusammenarbeit von Boing und ORNL

(cf. <http://hpc4energy.org/hpc-road-map/success-stories/boeing/>)



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

853

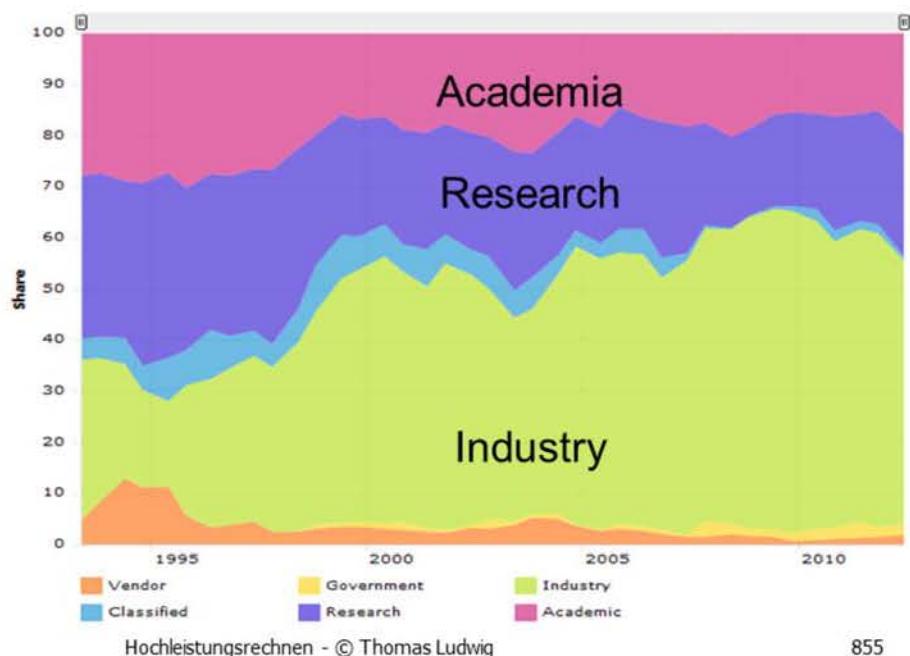
### Flugzeugdesign bei Boeing

- 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



12.10.2021

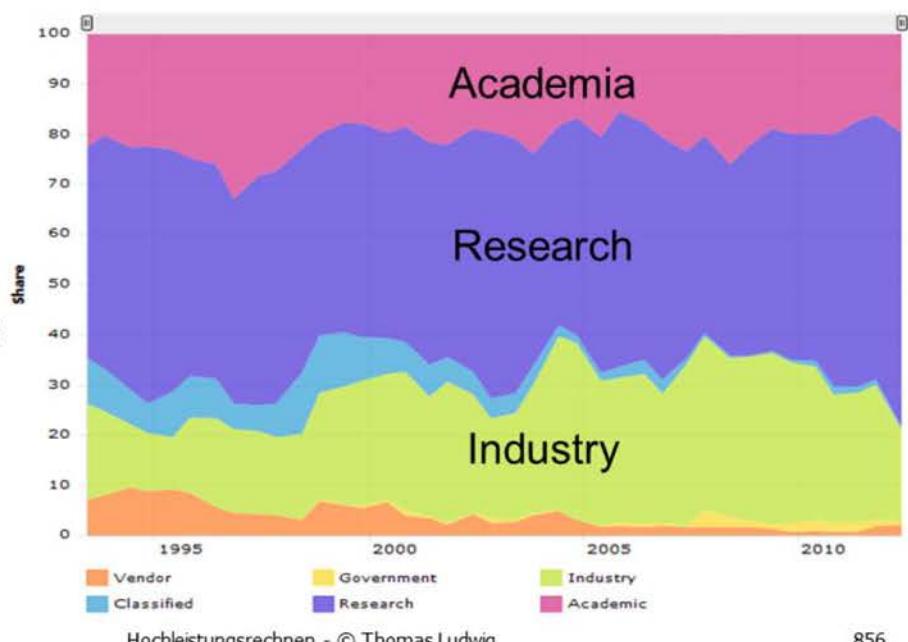
Hochleistungsrechnen - © Thomas Ludwig

855

# HPC in Wissenschaft und Technik...

TOP500  
June 2012

performance  
share



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

856

### **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



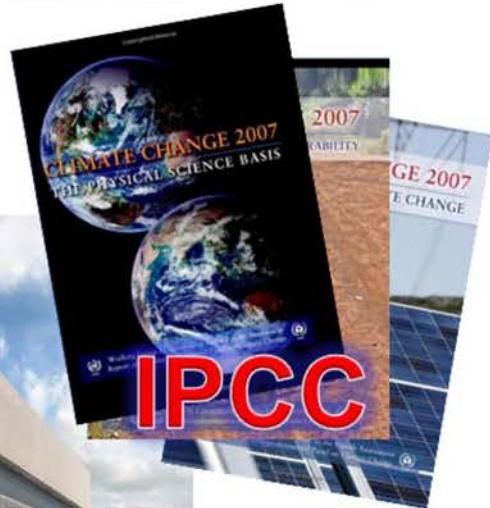
859

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

## Kosten-Nutzen-Modell des DKRZ



12.10.2021



IPCC

860

## 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

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

863

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

## Erster analytischer Ansatz...

For research oriented organizations

$$\text{productivity} = \frac{\text{(time saved by users on system)}}{\text{(BCR)} \cdot (\text{time to parallelize}) + (\text{time to train}) + (\cancel{\text{time to I-winch}}) + (\text{time to administrate}) + (\text{system cost})}$$

For industry environments

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

(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...

$$BCR = \frac{[Salary] \times 36000}{[Salary] \times (6200 + 800 + 228 + 2000 + 5000)} = \frac{36000}{14028} = 2.6,$$

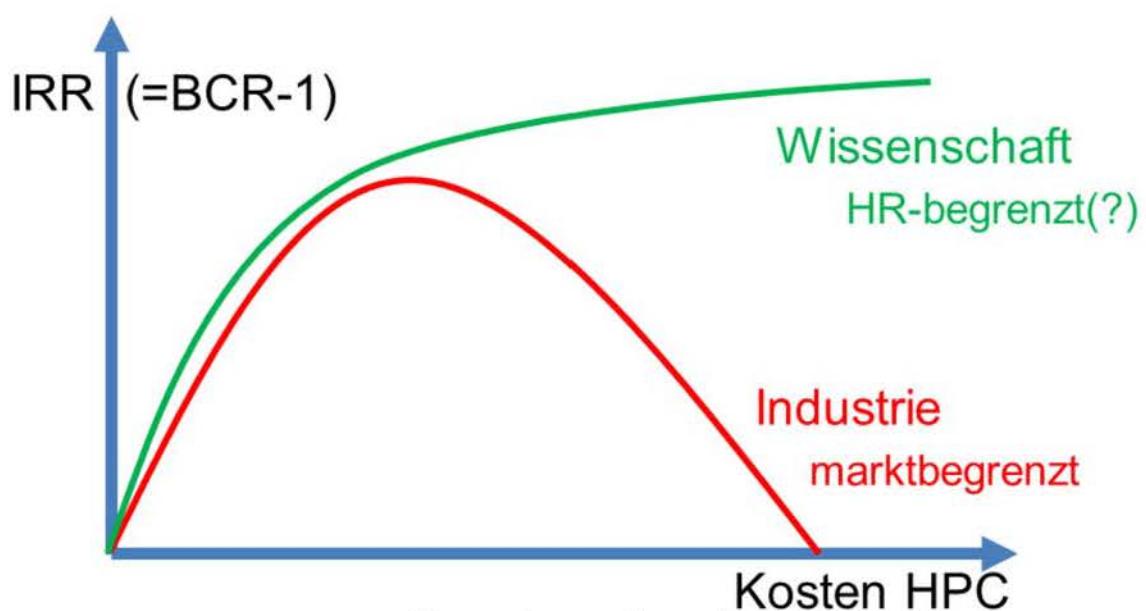
$$IRR_{\text{1year}} = BCR - 1 = 1.6 = 160\%.$$

Erspart Zeit für alle 200 Nutzer

Typischer Spruch eines Universitätskanzlers:

*"Warum Zeit sparen für Wissenschaftler – die sind ja eh da"*

## BCR-Überlegung [Ludwig]



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

867

## 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”

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

870

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

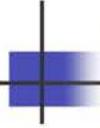
NSF-Förderung und Publikationen haben höhere Korrelationen mit Counts als mit dRankSum, d.h. eine konstante Investition bringt mehr als ein einzelner hoher Rang

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/Ahalt 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



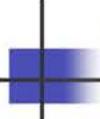
## Kosten-Nutzen-Analyse

Zwei weitere Beispiele...

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

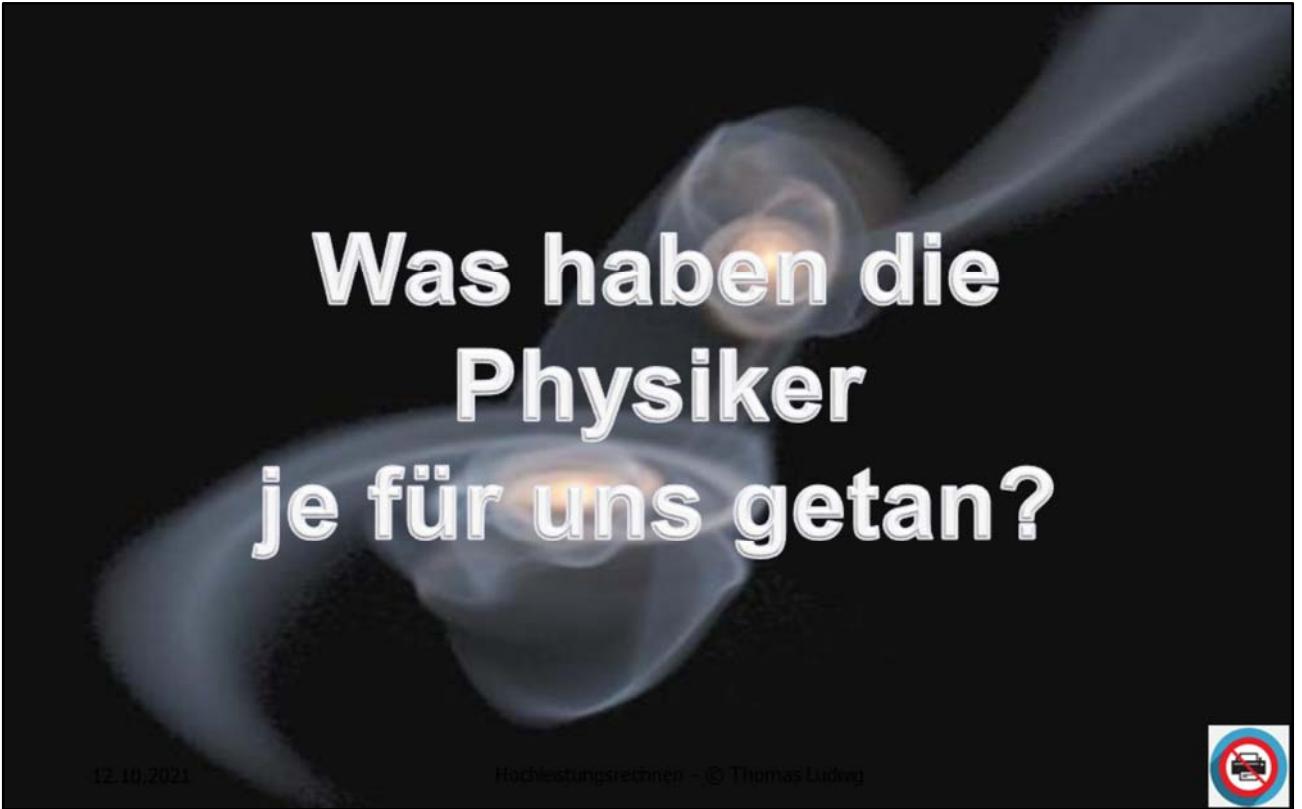
872



## 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 Physiker je für uns getan?

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig



AUSBLENDEN

## 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?

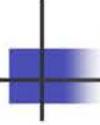
### Genereller Ansatz

- 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

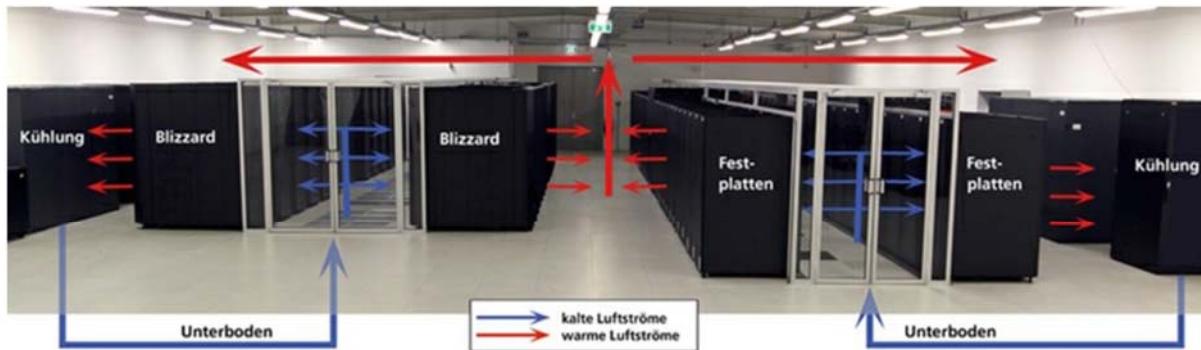
## Fiktives Beispiel Klimaforschung

### 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 450 t CO<sub>2</sub>

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

879

Siehe:

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

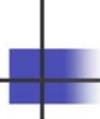
## Reales Beispiel HECToR

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...

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

881

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

12.10.2021

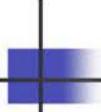
Hochleistungsrechnen - © Thomas Ludwig

882

## 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?



## **Rechnerbeschaffung**

1. HPC-Versorgung in Deutschland
2. Phasenmodell Beschaffung
3. Antragstellung
4. Markterkundung und Ausschreibung
5. Vertragsverhandlungen
6. Rechnerraumumbau und Installation
7. Produktionsbetrieb

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

884

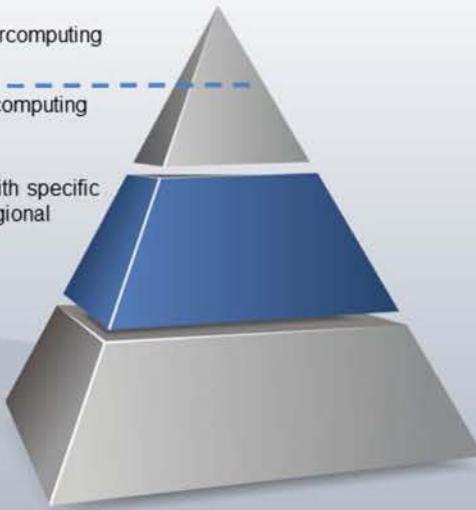
# HPC-Versorgung in Deutschland

**Tier 0:** European supercomputing Centres (PRACE)

**Tier 1:** National supercomputing centres (@GCS)

**Tier 2:** HPC centres with specific domains or specific regional responsibilities

**Tier 3:** Regional HPC centres and institutes with HPC resources



**53,242** PFlop/s RPeak

**4** HPC systems,  
**861.400** Cores

**25,31** PFlop/s RPeak

**14** HPC systems,  
**577.916** Cores

**2,366** PFlop/s Rpeak\*

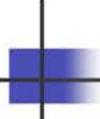
**22** HPC systems\*\*  
**124.852** Cores

**DKRZ: 3,6 PFLOPS**  
**100.000 Kerne**      **Januar 2020**

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

885



## Versorgungspyramide HPC

- „Höchstleistungs“rechner (Ebene 1)
  - Versorgung für Europa, Bund und Land
  - Gauss Center for Supercomputing
  - LRZ (Garching), HLRS (Stuttgart), JSC (Jülich)
- „Hochleistungs“rechner (Ebene 2)
  - Versorgung für Bund und Land
  - Z.B. Dresden, Aachen, Darmstadt, Hamburg (DKRZ)
- „Hochleistungs“rechner (Ebene 3)
  - Versorgung Land
  - Z.B. Erlangen, Kaiserslautern

# Geographische Verteilung

- Nicht alle Bundesländer vertreten
- Nordbundesländer im HLRN zusammengefasst
- Schwerpunkte in den industriereichen Bundesländern



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

887

## Finanzmittel

Ergebnis politischer Entscheidungen

- Ebene 1
  - Aktuell ca. 130 M€ pro Beschaffungszyklus und System
  - Beinhaltet auch Betriebskosten und Rechnergebäude
- Ebene 2
  - Typischerweise 15+ M€ pro Beschaffungszyklus und System
  - Ausnahme DKRZ mit 40+ M€ pro System
    - Andere Ausnahmen für Zentren mit anderen Finanziers
- Ebene 3
  - Typischerweise einige M€ pro Beschaffungszyklus und System

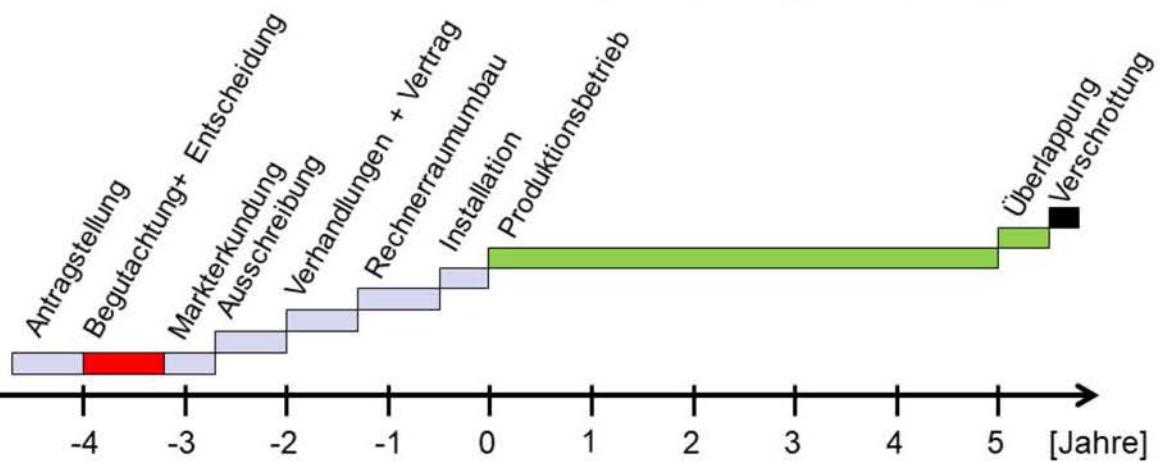
Seit 1.1.2021 neues Finanzierungskonzept im Rahmen des Nationalen Hoch- und Höchstleistungsrechnens (NHR)

## **Phasenmodell Beschaffung**

Regelmäßige Beschaffung z.B. alle 5 Jahre

- Antragstellung
- Markterkundung
- Ausschreibung
- Verhandlungen
- Kaufentscheidung
- Rechnerraumumbau
- Installation
- Produktionsbetrieb
- Dauerhaft:  
Politische Aktivitäten

## Zeitlicher Ablauf



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

890

## Antragstellung

- Absehbar zum Zeitpunkt des Betriebsendes
  - Alter Rechner überlastet (DKRZ: ca. 4fach überbucht)
  - Wissenschaft nicht mehr optimal unterstützt
- Am Anfang steht das Geld
  - Festlegung eines Rahmens liegt meistens vor
  - Üblicherweise: kontinuierliche Weiterentwicklung der Zentren, ihrer Systeme und Dienste
- Struktur eines Antrags (50-150 Seiten)
  - Darstellung neuer wissenschaftlicher Ziele (=Bedarf)
  - Darstellung neuer technischer Möglichkeiten im HPC (=Möglichkeiten der Bedarfsdeckung)
  - Abschätzung von verfügbarer Hardware, ihrer Beschaffungs- und Betriebskosten (=Umsetzungsplan)
  - Beschaffungs- und Betriebskonzept (=Details)

# Neue wissenschaftliche Ziele

## DRKZ und Klimaforschung: relativ homogenes Profil

- Neue Methoden der Wissenschaftler
  - Höhere räumliche und zeitliche Auflösung der Modelle
  - Mehr Prozesse (Wolken, Chemie und anderes)
  - Mehr Ensemble-Mitglieder  
Ensemble-Berechnungen: Verrechnungen statistischer Schwankungen der Ergebnisse bei modifizierten Eingaben

## Allgemeine Rechenzentren

- Verschiedene Wissenschaftsbereiche mit unterschiedlichen Programmcode-Strukturen und Abläufen in der computerbasierten Modellierung und Datenauswertung
  - Teilweise nur kleinere Datenmengen benötigt
  - Manchmal gut auf Beschleunigerhardware implementierbar/portierbar
  - ...

## Neue wissenschaftliche Ziele...

### Darstellung im Antrag

- Künftige Bedarfe an
  - Rechenzeit
  - Speicherplatz
  - Spezial-Hardware (Beschleunigung, Visualisierung, Nachverarbeitung...)
- Erwünschter Ausbau am DKRZ: Faktor 10...100

### Problem

- Wir befinden uns **4 Jahre** vor Inbetriebnahme des Systems
- Wissenschaftsentwicklung schwer vorhersagbar

# Neue technische Möglichkeiten

## Entwicklung in der Hardware

- Prozessoren
  - Prozessorfamilien von Intel und AMD; sonst noch Firmen?
- Speichersysteme
  - Insbesondere Dateisysteme: Lustre oder GPFS?
  - SSD-Burst-Buffer? Non volatile memory?
- Vernetzungen
  - Infiniband, OmniPath; wenige Überraschungen
- Spezialhardware
  - Beschleuniger: GPGPU, FPGA, Xeon Phi ...
  - Visualisierung
  - Bandspeicherung (HSM)

## Problem

- Wir befinden uns **4 Jahre** vor Inbetriebnahme des Systems
- Technische Entwicklung schwer vorhersagbar

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

894

## Abschätzung der verfügbaren HW und Kosten

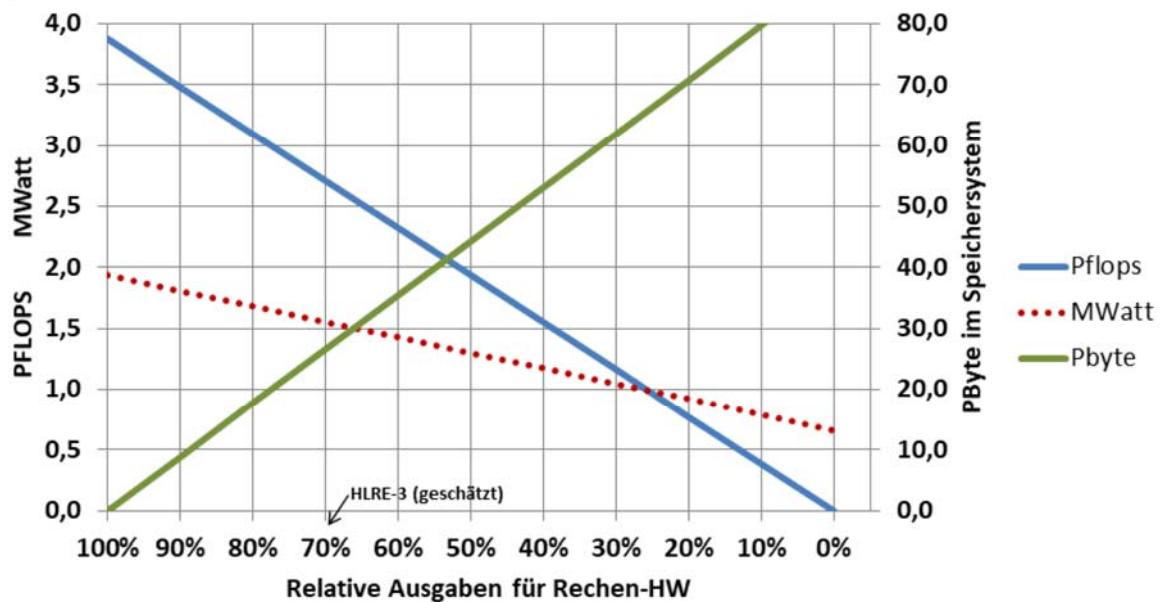
### Beschaffungs- und Betriebskosten

- Aus getrennten Budgets: Beschaffung durch Beantragung, Betrieb aus Jahreshaushalt
- Muss am Ende zusammenpassen
- Problem: wenn ich alles Geld für HW ausgebe, kann ich dann den Strom des Systems bezahlen?

### Speziell am DKRZ

- Aufteilung der Ausgaben für Rechnen und Speichern

## Abschätzung der verfügb. HW und Kosten...



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

896

Die Grafik zeigt die relativen Ausgaben für Rechen-HW vom Gesamtbudget. Bei 100% erhält man knapp 4 PFLOPS, bei 0% keine Rechen-HW, aber ca. 120 PByte im Speichersystem. Der für beide Teile in Summe entstehende Stromverbrauch ist rot gepunktet dargestellt.

# Abschätzung der verfügb. HW und Kosten...

## Abschätzung HLRE-3 im Antrag 2011

Rechnersystem: 31M€ - HSM: 5M€ - Umbauten: 5M€

| Characteristic               | HLRE-2                        | HLRE-3                          | Factor |
|------------------------------|-------------------------------|---------------------------------|--------|
| a) Peak performance          | 158 TFLOPS                    | ~ 3,000 TFLOPS                  | ~ 20   |
| b) No. of processor cores    | 8,300                         | ~ 120,000                       | ~ 14   |
| c) Mean memory per core      | 3 GB (cores with 4GB and 2GB) | ~ 3 GB (cores with 4GB and 2GB) | ~ 1    |
| d) Main memory               | 20 TB                         | ~ 360 TB                        | ~ 18   |
| e) Storage on disk           | 6 PB                          | ~ 120 PB                        | ~ 20   |
| f) Memory-to-disk            | 30 GB/s                       | ~ 600 GB/s                      | ~ 20   |
| g) Full memory dump to disk  | < 1 hour                      | < 1 hour                        | ~ 1    |
| h) Storage on tape           | 65 PB                         | ~ 650 PB                        | ~ 10   |
| i) Disk-to-tape              | 3 GB/s                        | ~ 30 GB/s                       | ~ 10   |
| j) Annual data production    | 10 PB/year                    | ~ 100 PB/year                   | ~ 10   |
| k) Overall power consumption | 2 MW                          | 2 MW                            | 1      |

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig



AUSBLENDEN

# Beschaffungs- und Betriebskonzept

## Mehrphasige Installation

- Installation in zwei Phasen mit einem Jahr Abstand
- Ziel
  - Im ersten Jahr ist der Rechner noch nicht ausgelastet
    - Kleineres System genügt; wir sparen Strom
  - Wir sparen Geld auf für bessere Technik
    - HLRE-3: Haswell- und Broadwell-Prozessoren
  - Nachteil: System ist in den Komponenten heterogen
    - Erschwert Verwaltung, Jobs eher nicht auf beiden Teilen zugleich
- Alternativen
  - Z.B. mehrere Systeme, die im Wechsel oder in Ergänzung hochgezogen werden (z.B. in Jülich)
  - Nachteil: wiederholte vollständige Beschaffungen notwendig

# Markterkundung

Zeitpunkt: 3 Jahre vor Inbetriebnahme

## Ziel

- Frühzeitige Kontaktaufnahme mit potentiellen Anbietern
- Übersicht über Entwicklungslinien bei
  - Prozessoren
  - Speichersystemen
  - Anderen HW- und SW-Systemen
- Grobe erste Ideen von PFLOPS/M€ und PByte/M€
- Erste Abschätzungen von Stromverbräuchen
- Kommunikation unserer Zielvorstellungen an Hersteller

Wichtig: Ausschreibung muss damit umsetzbar sein

## Ausschreibung

- Zeitpunkt: 2 Jahre vor Inbetriebnahme
- Europaweite Ausschreibung nach den Regularien aus dem öffentlichen Bereich
  - Strenge rechtliche Abwicklungsvorgaben zur Erzielung von Chancengleichheit, Korruptionsfreiheit usw.
  - Vermeide IT-Elbphilharmonie ☺
- Üblich bei anderen Produkten
  - Bedarf definieren – Ausschreibung gewinnt der Bieter mit dem wirtschaftlichsten Angebot (nicht notwendigerweise das billigste)
- Üblich bei HPC-Ausschreibungen
  - Geldsumme festlegen – Ausschreibung gewinnt der Bieter mit der am besten bewerteten Leistung (oft: meiste Hardware)
- Bewertungsschema mit Ausschreibung festgeschrieben

# Ausschreibungsdokument

Vergabeunterlagen (RFP – Request for Proposals)

- Festlegung der Wertung der Angebote
- Systempreis und Preise für Erweiterungen
- Leistungsanforderungen Rechnen (Phase 1 und 2)
- Leistungsanforderungen Speichern (Phase 1 und 2)
- Unterstützende HW und SW
- Elektrische Leistungsaufnahme
- Integration in bestehende Infrastruktur
- Benchmarks

Umfang: ca. 50 Seiten am DKRZ

## Ausschreibungsdokument...

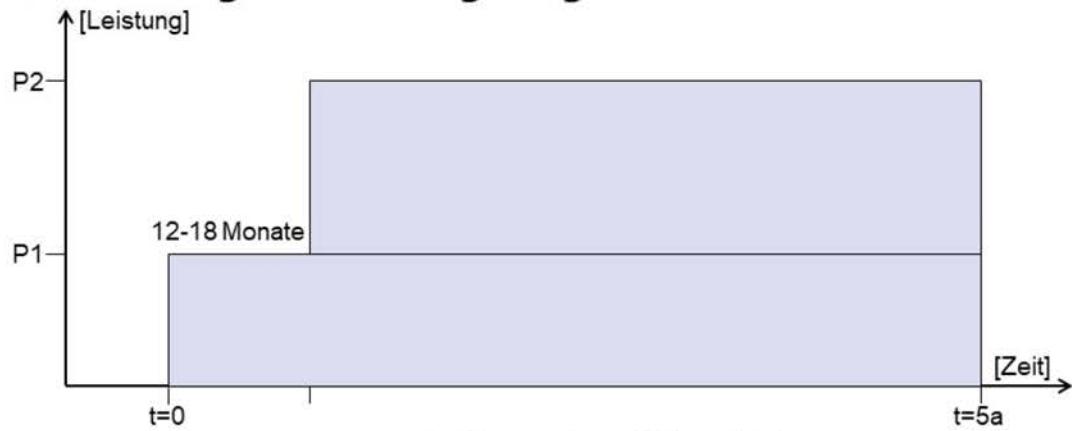
### Festlegung der Wertung der Angebote

- Als Excel-Tabelle mit etwa folgenden Gewichtungen
  - Preis (1/4 der Punkte)
  - Preise für Erweiterungen (TFLOPS, PByte)
  - Rechenleistung (ca. 1/3 der Punkte)
  - Speichersystem (ca. die Hälfte der Rechenleistung)
  - Punkte für Software und andere (weiche) Faktoren
- Die Ausschreibung gewinnt der Bieter mit der höchsten Punktzahl
- Muss alles rechtssicher dokumentiert werden
- Klagen bei Fehlern sind wahrscheinlich!

## Ausschreibungsdocument...

Leistungsanforderungen Rechnen (Phase 1 und 2)

- Integrale Leistung über beide Phasen angefordert
- Leistungsbewertung aufgrund von Benchmarks



## Ausschreibungsdocument...

Leistungsanforderungen Speichern (Phase 1 und 2)

- Speicherkapazitäten z.B. vorgegeben
  - Für beide Phasen getrennt
- Anzahl zu speichernder Dateien vorgegeben
- Forderung nach qualitativem Dateisystem
  - Z.B. Zeit zum Neustart nach Absturz
- Problem: es gibt nur Lustre und GPFS
  - IBM liefert GPFS, alle anderen Lustre

# Ausschreibungsdocument...

## Unterstützende Hardware und Software

- Zusätzliche Rechnerknoten mit Grafikkarten für Visualisierungen
  - Direkt angebunden an das Speichersystem
- Testsystem
  - Für Tests neuer Software und Firmware
- Software-Komponenten
  - Linux
  - Batch-Scheduler mit Vorgabe bzgl. Steuermöglichkeiten
  - Backup-Software
  - Compiler Fortran/C/C++, Bibliotheken, MPI, OpenMP
  - Werkzeuge zur Fehlersuche und Leistungsanalyse

## Ausschreibungsdokument...

### Elektrische Leistungsaufnahme

- Vorgabe eines maximalen akzeptierten Verbrauchs
  - Grund: Finanzierung von Strom und steigenden Energiesteuern ist kritisch
- Ermittelt durch Mix von realistischen Benchmark-Programmen
  - LINPACK nur geeignet, um Maximalverbrauch zu testen
  - Hier
    - Maschine mit Anwendungsbenchmarks vollpacken  
Dann Leistungsaufnahme messen

## Ausschreibungsdocument...

Integration in bestehende Infrastruktur

- Vorgabe der Stellflächen
- Vorgabe der Bodenbelastungen
- Vorgabe der Stromschienen und Stromverteiler
- Vorgabe der Kühlsysteme

Weitere Kleinigkeiten

- Deckenhöhen
- Säulenabstände
- Türweiten, Höhe und Belastbarkeit der Aufzüge

## Ausschreibungsdokument...

Leistungsanforderungen aus Kundensicht

- Anzahl geeigneter Bieter: ca. 3-6 Hersteller
- Was hindert Bieter an einer Teilnahme?
  - Z.B. zu strenge Vorgaben für Dateisystem
  - Z.B. zu enges Strombudget
- Folge: Ausschreibung bleibt ohne Angebote
  - Ist an anderer Stelle bereits geschehen
  - Neuauusschreibung erforderlich
    - Zeitverlust, Reputationsverlust
    - Probleme mit Finanzierung von altem und neuem Rechner

# Ausschreibungsdokument...

## Leistungsanforderungen aus Bietersicht

- Bieter bietet viel Leistung
    - Höhere Gewinnchancen im Wettbewerb
    - Muss dann aber auch die nötige Hardware liefern, wenn er gewinnt
  - Bieter ist vorsichtig mit Leistungsprognosen
    - Verringerte Gewinnchancen
    - Im Gewinnfall aber auch realistische Hardware-Lieferung
- IT-Branche allgemein: schwieriges Geschäft
- Beteiligung an Ausschreibung teuer für Bieter

# Ausschreibungsdocument...

## Benchmarks

- Benchmarking ist eine Kunst
- Es gibt unzählige Vorgehensvarianten
- Vielleicht einfachste: LINPACK-Benchmark verwenden
- Unser Ansatz
  - Rechenleistung  
Anwendungsbenchmarks  
Mix relevanter Modelle der Kunden mit MPI und OpenMP  
Methode: Erhöhung des Jobdurchsatzes
  - Speichersystemleistung  
Synthetische Benchmarks  
Methode: Vorgabe von Leistungsdaten

# Ausschreibungsdocument... Benchmarking

## Anwendungsbenchmarks – Vorgehensmodell (1)

- Wir bestimmen eine Referenzlaufzeit für einen Modellcode, z.B. 10 Minuten (dazu benötigen wir n Kerne)
- Wir ermitteln auf unserer alten Maschine, wieviele Jobs wir pro Sekunde auf der vollen Maschine durchbekommen
- Der Bieter ermittelt die kleinste Anzahl von Kernen, mit denen er die Referenzzeit unterschreitet
- Für die von ihm gebotene Anzahl Kerne bestimmt er den Durchsatz für seine volle Maschine
- Der Quotient der beiden Durchsätze ist die Durchsatzsteigerung für diesen Benchmark

# Ausschreibungsdocument... Benchmarking

## Anwendungsbenchmarks – Vorgehensmodell (2)

- Nicht ein Benchmarkcode sondern ein halbes Dutzend
- Jeweils evaluiert in zwei Varianten
  - Unoptimiert (nur Compilereinstellungen)  
Zeigt uns, was das System und der Compiler können
  - Optimiert  
Zeigt uns, was das Team des Bieters leisten kann
- Macht ein Dutzend Varianten
  - Unterschiedliche Gewichtung pro Benchmark (optimierte geringer)
- Getrennt angegeben für Phase 1 und Phase1+Phase2
- Gewinner ist der Bieter mit dem höchsten Wert

# Ausschreibungsdocument... Benchmarking

## E/A-Benchmarks

- Single stream Posix-Transferrate von/zu Rechnerknoten
- Aggregierte Posix-Streaming-Transferrate
- Leistungen für HDF5 und NetCDF
- Parallele E/A mit MPI auf eine Datei
- Metadaten-Leistung

Hier jeweils Vorgaben an den Bieter, die er einhalten muss

# Vertragsverhandlungen

Zeitpunkt: ca. 1,5-2 Jahre vor Inbetriebnahme

- Bieter liefern Angebot
  - Dokument mit 200+ Seiten
    - Technische Spezifikation
    - Ergebnisse des Benchmarking
    - Konditionen für Lieferung, Inbetriebnahme, Wartung usw.
  - Typischerweise stellen beide Seiten Problembereiche im Ausschreibungsdokument fest
    - Weitere Detaillierung der Ausschreibung
    - Kommunikation an alle
    - Neue Runde mit Angeboten
  - Nach mehreren Runden konvergiert das Verfahren

# Ergebnisse des Benchmarking ☺

## Probleme

- Anwendungsbenchmarks: Zielprozessor existiert nicht
  - Alle Angaben sind Hochrechnungen
  - Erstellt auf einem existierenden Prozessor (Vorgängermodell)
  - Hintergrundinformation des Prozessorherstellers über Leistungszuwachse der kommenden Generation am Bieter
  - Hochrechnungen und Simulationen beim Bieter
  - Datum der Markteinführung nicht genau bestimmbar
  - Varianten des Prozessor bei Markteinführung im Voraus nicht final bekannt
    - Stromverbräuche aller Prozessortypen nicht final bekannt
  - Preise auch nicht final bekannt
- Somit: Bieter spielt Benchmark-Poker
- Verfehlten der Leistungszusagen
  - Erfolgreicher Bieter muss soviel HW nachliefern, bis Leistungszusage erfüllt

## Ergebnisse des Benchmarking... ☺

### Probleme

- E/A-Benchmarks: System der ausgeschriebenen Größe existiert noch gar nicht
  - Leistungsangaben sind Hochrechnungen
  - Insbesondere bei Dateisystemen unklar, ob sie die geforderte Größe und Skalierbarkeit mit allen Qualitätsforderungen erfüllen können
- Somit: Bieter spielt Benchmark-Poker
- Verfehlten der Leistungszusagen
  - Erfolgreicher Bieter muss soviel HW nachliefern, bis Leistungszusage erfüllt

# Rechnerraumumbau und Installation



12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

920

# Rechnerraumumbau und Installation...

## Umbauten am Rechnerraum (HLRE-3)

- Stromversorgung
  - Batteriepufferung (höhere Verfügbarkeit)
  - Weiterer Mittelspannungstransformator (höhere Verfügbarkeit)
  - Umbau der Stromschienen (andere Aufstellung im Raum)
- Kühlung
  - Rechnersysteme mit Hochtemperaturflüssigkeitskühlung
    - Ermöglicht ganzjährige freie Kühlung über das Dach ohne weitere Kühlaggregate
- Bandarchiv mit Sauerstoffreduktionsanlage
  - Reduktion von 20,5% (normal) auf 17% und 15%
  - Entstehung von Bränden weitestgehend verhindert

# Batteriepufferung



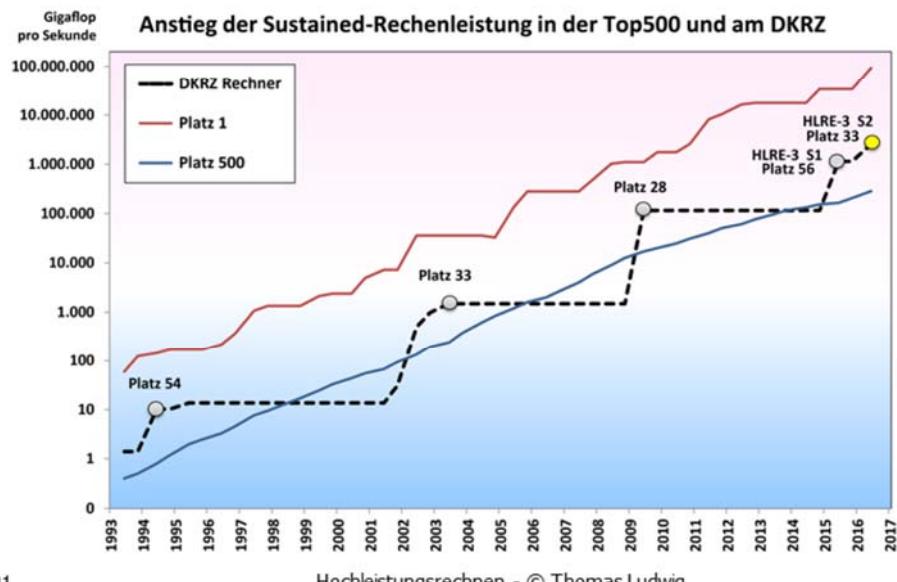
12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

923

# Produktionsbetrieb

DKRZ ist wieder im Rennen

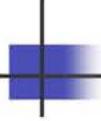


12.10.2021

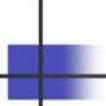
Hochleistungsrechnen - © Thomas Ludwig

924

FOLIE NICHT DRUCKEN



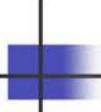
Danach geht alles von vorne los...



## Rechnerbeschaffung

### Zusammenfassung

- Zeitdauer der Beschaffung: >4 Jahre
- Antragstellung sehr früh mit erster Abschätzung zu Wissenschaft und HPC-Technik
- DKRZ: Aufteilung der Finanzmittel auf Rechnen und Speichern wichtig und schwierig
- Beschaffung mit zwei Installationsphasen
- Ausschreibung ist aufwendig
- Bieterverfahren muss rechtskonform ablaufen
- Problem für Kunde: Definition geeigneter Benchmarks
- Problem für Bieter: Benchmark-Hochrechnungen bei nicht-existentierender Ziel-Hardware



## **Rechnerbeschaffung**

Die wichtigsten Fragen

- In wievielen Phasen soll die Installation ablaufen?
- Wie könnte ein Benchmarking für eine zu beschaffende Maschine aussehen?
- Was muss der Kunde beachten?
- Welche Probleme stellen sich für den Bieter?

12.10.2021

Hochleistungsrechnen - © Thomas Ludwig

928