Rechnerstrukturen

Vorlesung im Sommersemester 2008

Prof. Dr. Wolfgang Karl

Universität Karlsruhe (TH)

Fakultät für Informatik

Institut für Technische Informatik

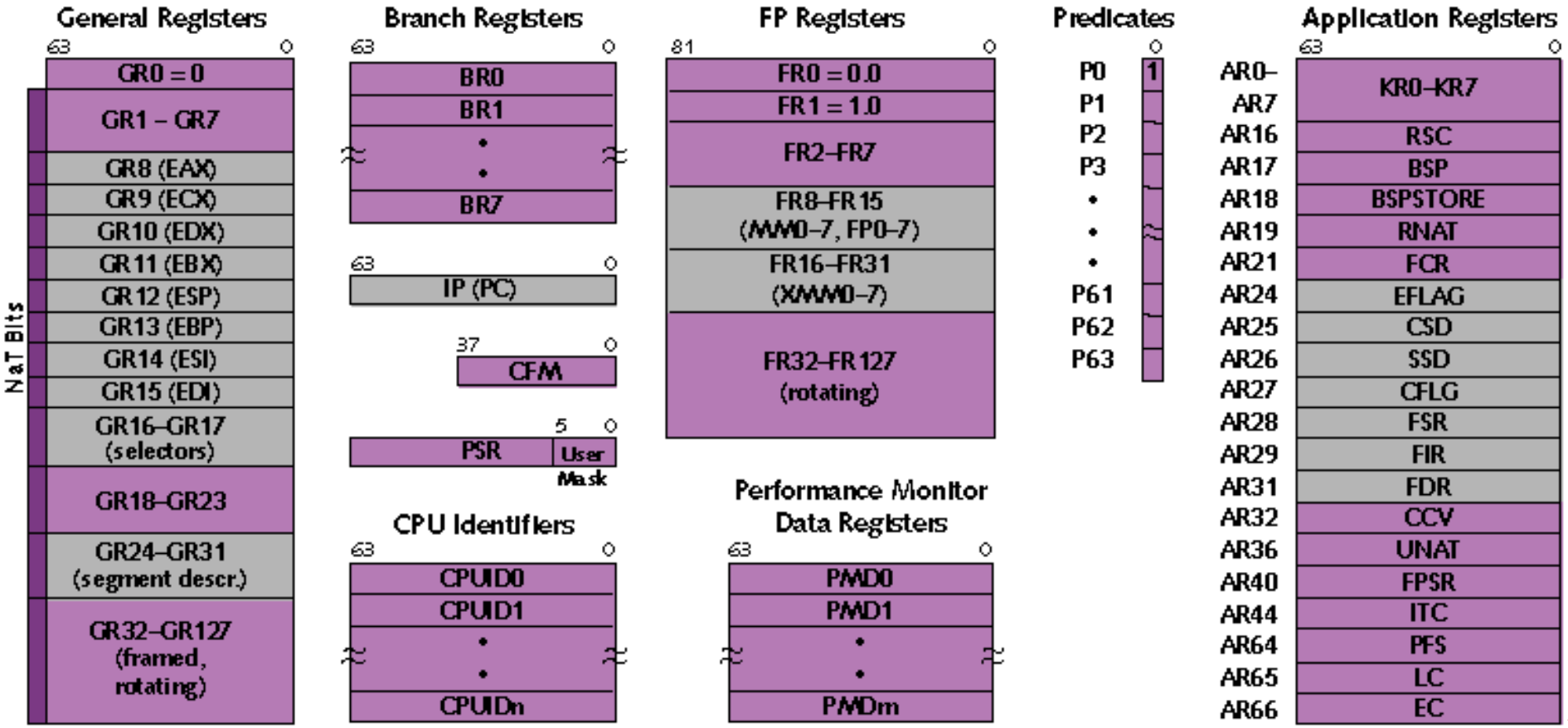
- **Kapitel 2: Parallelismus auf Befehlsebene**

2.3: Nebenläufigkeit, VLIW, EPIC

- EPIC: Explicitly Parallel Instruction Computing
  - Gemeinsames Projekt von Hewlett-Packard und Intel (1994 angekündigt)
  - Ziele:
    - 64 Bit Architektur: IA-64
    - Explizite Spezifikation des Parallelismus im Maschinencode: EPIC-Format, (entspricht VLIW-Prinzip)
    - Bedingte Ausführung von Befehlen (Predication)
    - Spekulative Ausführung von Ladeoperationen (Data Speculation)
    - Großer Registersatz
    - Skalierbarer Befehlssatz
    - Sinnvolles Zusammenwirken zwischen Compiler und Hardware
  - Itanium: erster Prozessor der P7-Generation (Code-Name Merced)

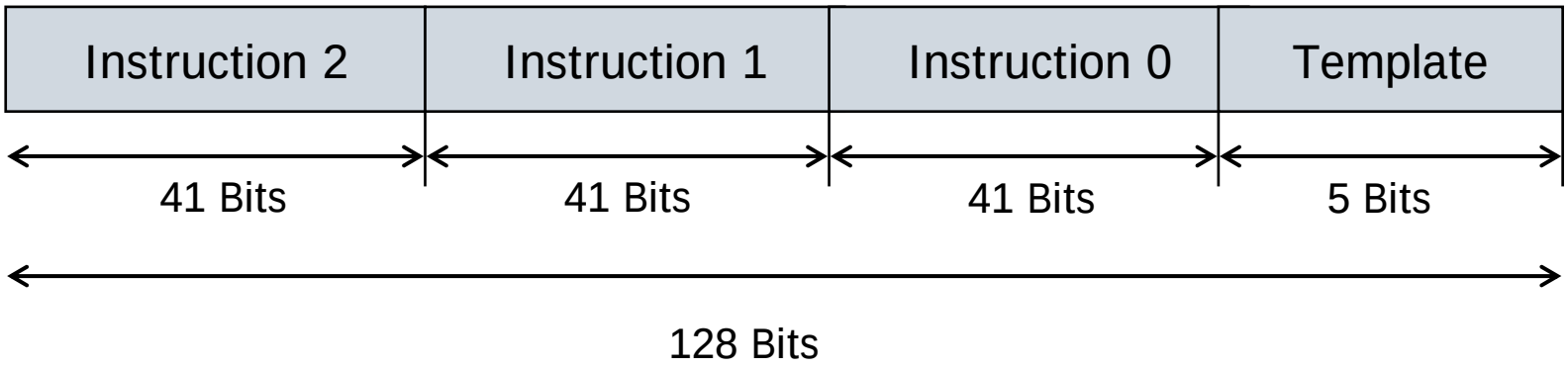
- Intel IA-64
  - Registersatz

Quelle: Microprocessor Report, Vol.13, Nr.7, 1999



- IA-64 ISA: Befehlsformat
  - Opcode
  - Predicate
  - 2 Felder für Quelloperandenregister
  - 1 Feld für Zielregister

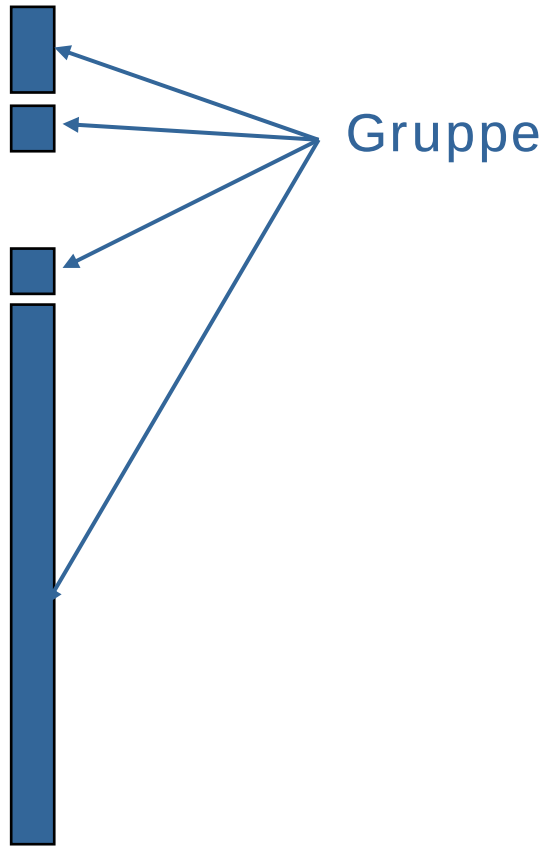
- IA-64 ISA
  - IA-64 Instruktionen werden vom Compiler in so genannte Bundles gepackt



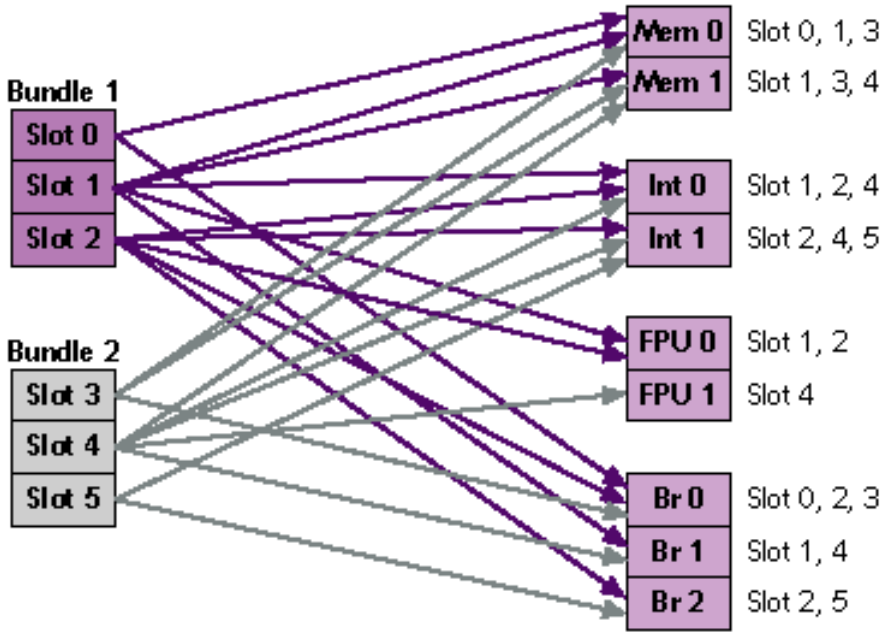
- IA-64 ISA
  - Bundles:
    - Template: zeigt explizit an, ob
      - die Instruktionen im Bundle gleichzeitig ausgeführt werden dürfen, oder
      - eine oder mehrere Instruktionen sequentiell auszuführen sind, oder
      - benachbarte Bundles parallel ausgeführt werden können.

- Beispiele von Befehlsgruppen:

```
{ .mii
        add r1 = r2,r3
        sub r4 = r4, r5;;
        shr r7 = r4, r12;;
}
{ .mmi
        ld8 r2 = [r1];;
        st8 [r1] = r23
        tbit p1,p2 = r4,5
}
{ .mbb
        ld8 r45 = [r55]
(p3)    br.call b1  =func1
(p4)    br.cond Label1
}
{ .mfi
        st4 [r45] = r6
        fmac f1 = f2,f3
        add r3 = r3,8;;
```



- Anstoßen der Befehle zur Ausführung
  - Beispiel: Itanium



  - Es können bis zu sechs Instruktionen pro Takt zur Ausführung angestoßen werden. Die Instruktionen kommen von zwei Bundles. Jeder Bundle hat 3 Slots.

- IA-64: Skalierbarkeit
  - Jedes Bundle enthält drei Instruktionen für eine Menge von drei Funktionseinheiten.
  - Ein IA-64 Prozessor kann *n* Mengen von jeweils drei Funktionseinheiten haben, welche die Informationen im Template nutzen, dann können mehrere Bundles in ein Instruktionswort mit der Länge *n* Bundles gepackt werden.

→ Skalierbarkeit bezüglich der Anzahl der Funktionseinheiten

- Predication, bedingte Befehlsausführung
  - Beispiel:

```
if (r1 == r2)
        r9 = r10 - r11;
else
        r5 = r6 + r7;
```

Bedingte Befehlsfolge

```
        cmp.eq p1,p2 = r1, r2;;
(p1)    sub r9 = r10, r11
(p2)    add r5 = r6, r7
```

- Evaluierung der bedingten Ausdrücke mit Hilfe von Compare-Operationen.
- Jeder Befehl hat ein 6 Bit breites Predicate Feld zur Angabe des Predicate-Registers
- Elimination von Sprüngen

- Predication (Bedingte Befehlsausführung)
  - Zur Laufzeit werden die voneinander unabhängigen Befehle zur Ausführung angestoßen.
  - Der Prozessor führt die Befehle auf den möglichen Programmverzweigungen aus, aber speichert die Ergebnisse nicht endgültig.
  - Überprüfen der Predicate Register
    - Falls das Register eine 1 enthält, dann ist wird die Ausführung der Instruktion abgeschlossen.
    - Falls das Register eine 0 enthält, dann wird das Ergebnis verworfen.

- IA-64 Control Speculation
  - Problem: Verzweigungen schränken Code-Verschiebungen ein

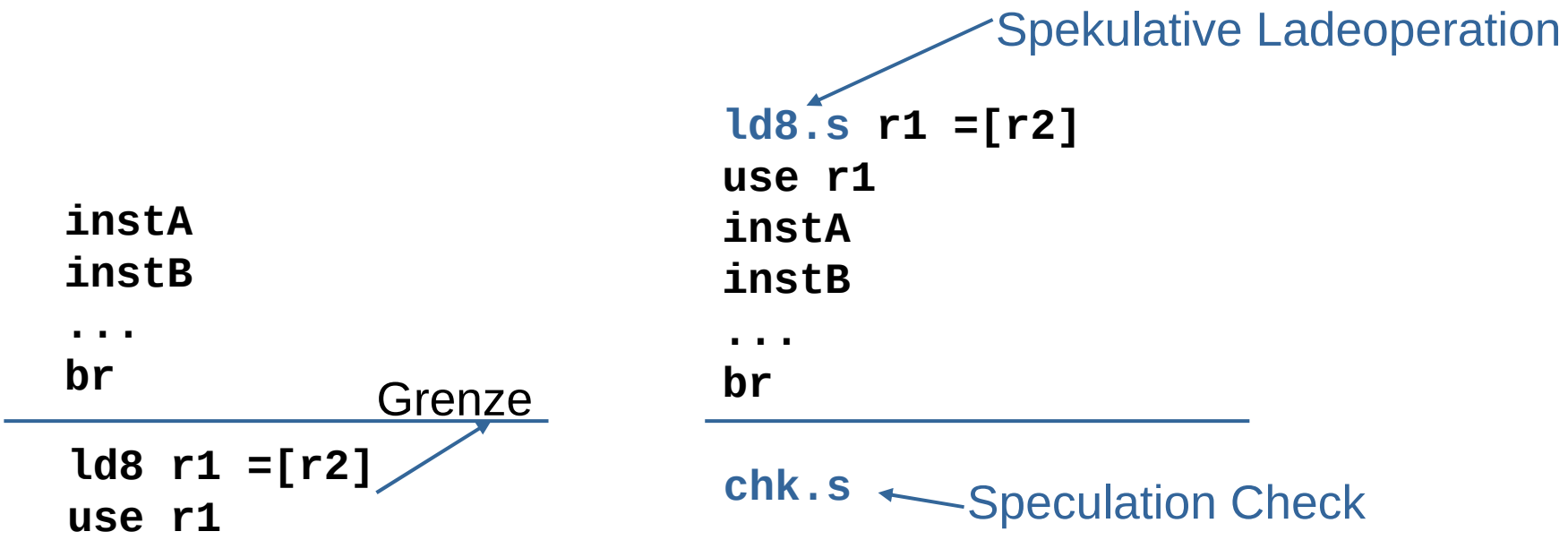
```
instA
instB
...
br
```

Grenze

```
ld8 r1 =[r2]
use r1
```

Ladeoperation kann nicht über den Sprung verschoben werden, denn es könnten Alarme ausgelöst werden.

- IA-64 Control Speculation
  - Lösung: Spekulative Operationen, die keine Alarme auslösen



- IA-64 Control Speculation
  - Einführung von spekulativen Ladeoperationen
    - Verursachen keinen Alarm:
      - Falls zur Laufzeit die Operation einen Alarm auslöst, dann wird dieser Alarm verzögert
      - Setzen von NaT-Bit (Deferred Exception Token, Not-a-Thing Bit) in Zielregister
    - Spekulative Ladeoperation kann über Verzweigungen hinweg verschoben werden
    - Einfügen von Speculation Check Instruktion (chk.s) anstelle der Ladeoperation
      - Zur Laufzeit überprüft die Check-Operation das Zielregister, ob NaT gesetzt ist.Wenn ja, dann erfolgt eine Verzweigung zu einem speziellen Fix-up-Code.

- IA-64 Data Speculation
  - Problem: Zeiger können Compiler zu konservativen Annahmen über die Referenzen zwingen, was eine Code-Verschiebung verhindert.

```
instA
instB
...
store
```

```
ld8 r1 =[r2]
use r1
```

Ladeoperation kann nicht über den die Speicheroperation verschoben werden, da beide dieselbe Adresse referenzieren können.

- IA-64 Data Speculation
  - Lösung: Vorgezogene Ladeoperationen

```
instA
instB
...
store
```

```
ld8 r1 =[r2]
use r1
```

Vorgezogene Ladeoperation

```
ld8.a r1 =[r2]
use r1
instA
instB
...
store
```

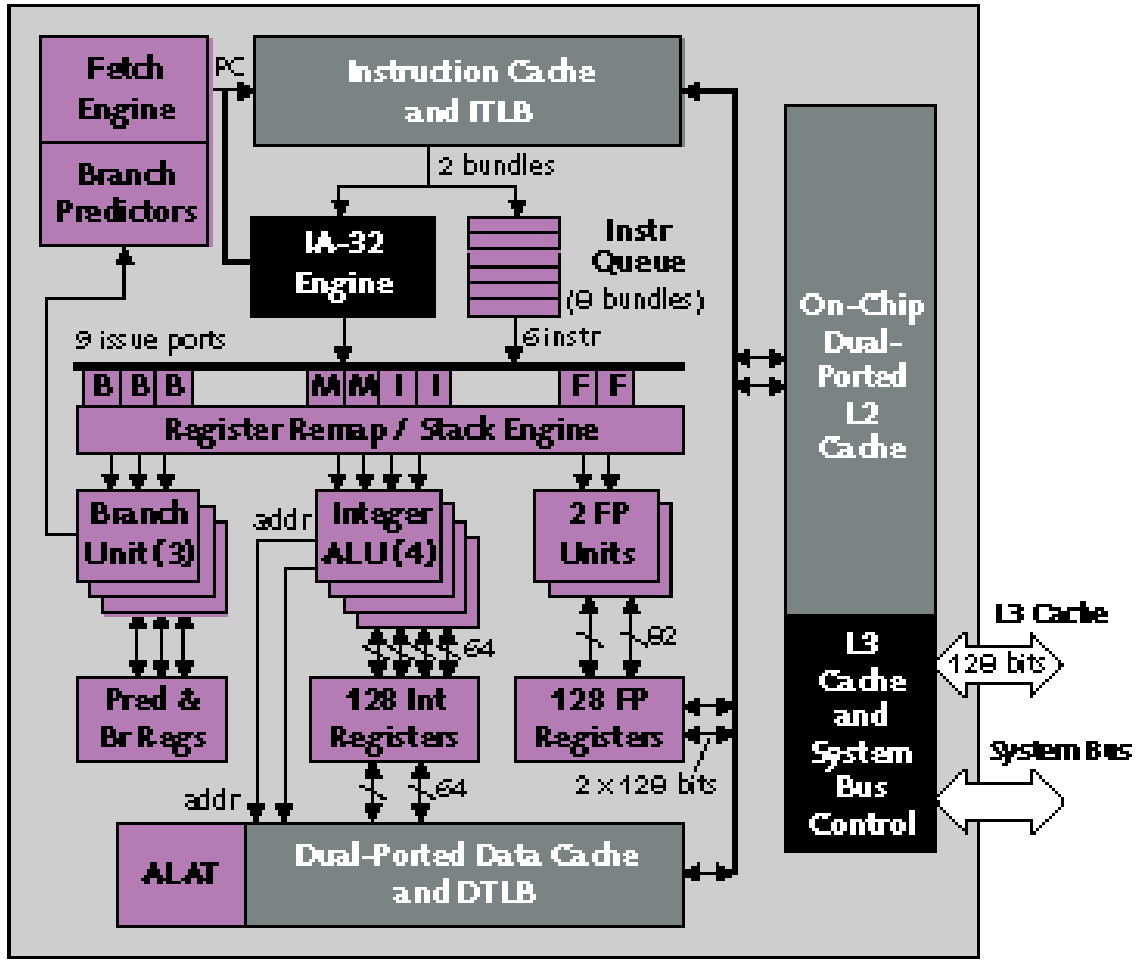
```
chk.a
```

Speculation Check

- IA-64 Data Speculation
  - Lösung: Vorgezogene Ladeoperationen
    - Verschieben von Lade-Operationen auch vor Speicher-Operationen.
      - Vorgezogene Lade-Operation (ld.a)
      - Zur Laufzeit werden Informationen (Zielregister, Speicheradresse, auf die zugegriffen wird, Zugriffsgröße) in Advanced Load Address Table (ALAT) festgehalten.
      - Zur Laufzeit prüft die Hardware, wenn eine Speicheroperation ausgeführt wird, ob eine Adresse in der ALAT mit der Zugriffsadresse übereinstimmt. Wenn ja, dann wird der Eintrag in ALAT gelöscht

- IA-64 Data Speculation
  - Lösung: Vorgezogene Ladeoperationen
    - Einfügen der Check-Operation (`chk.a`)
      - Prüft, ob Eintrag von der entsprechenden vorgezogenen Ladeoperation in ALAT steht.
      - Wenn nicht, dann hat es eine Kollission mit einer Speicheroperation gegeben und es erfolgt eine Verzweigung zu einem Fix-up-Code.

- Intel Itanium



Quelle: Microprocessor Report, Vol.13, Nr.13, October 5, 1999

EPIC

- Intel Itanium
  - 64-Bit Prozessor der P7 Generation
  - EPIC
  - 10-stufige Pipeline mit 6 Befehlen, die gleichzeitig zur Ausführung angestoßen werden können.
  - 128 GP- und 128 FP-Register - keine Registerumbenennung: rotierende Registerfenster
  - Bedingte Befehlsausführung: 64 1 Bit Predicate Register
  - 9 Funktionseinheiten
  - Misprediction Penalty: 9 Zyklen

- Intel Itanium
  - Spekulative Ladeopationen
  - 4-fach mengenassoziative L1 Befehls- und Daten-Cache-Speicher (Write-through)
  - 6-fach mengenassoziativer L2 Cache (Copy-back)
  - Informationen
    - Intel: http://www.intel.com/
      - Intel Technology Journal: http://developer.intel.com/technology/itj
    - IEEE Micro: Sonderheft im Spe./Okt. 2000
    - Henn./Patt.: Computer Architecture: Kap. 4.7

Superskalartechnik

- Literatur:
  - Brinkschulte/Ungerer: Mikrocontroller und Mikroprozessoren. Springer-Verlag, 2002: Kap. 6.1-6.4, Kap. 7
  - Hennessy/Patterson: Computer Architecture – A Quantative Approach. 3. Auflage: Kap. 3

- **Kapitel 2: Parallelismus auf Befehlsebene**

2.4: Thread-Level Parallelismus, Multithreading

- Grundsätzliche Aufgabe beim Prozessorentwurf:
  - Reduzierung der Untätigkeits- oder Latenzzeiten
    - Entstehen bei Speicherzugriffen, insbesondere bei Cache-Fehlzugriffen
    - Bei speichergekoppelten Multiprozessoren, wenn auf nicht-lokalen Speicher zugegriffen wird
    - Synchronisation von parallelen Kontrollfäden (Threads)
  - Lösung: parallele Ausführung mehrerer Kontrollfäden

Mehrfädigkeit

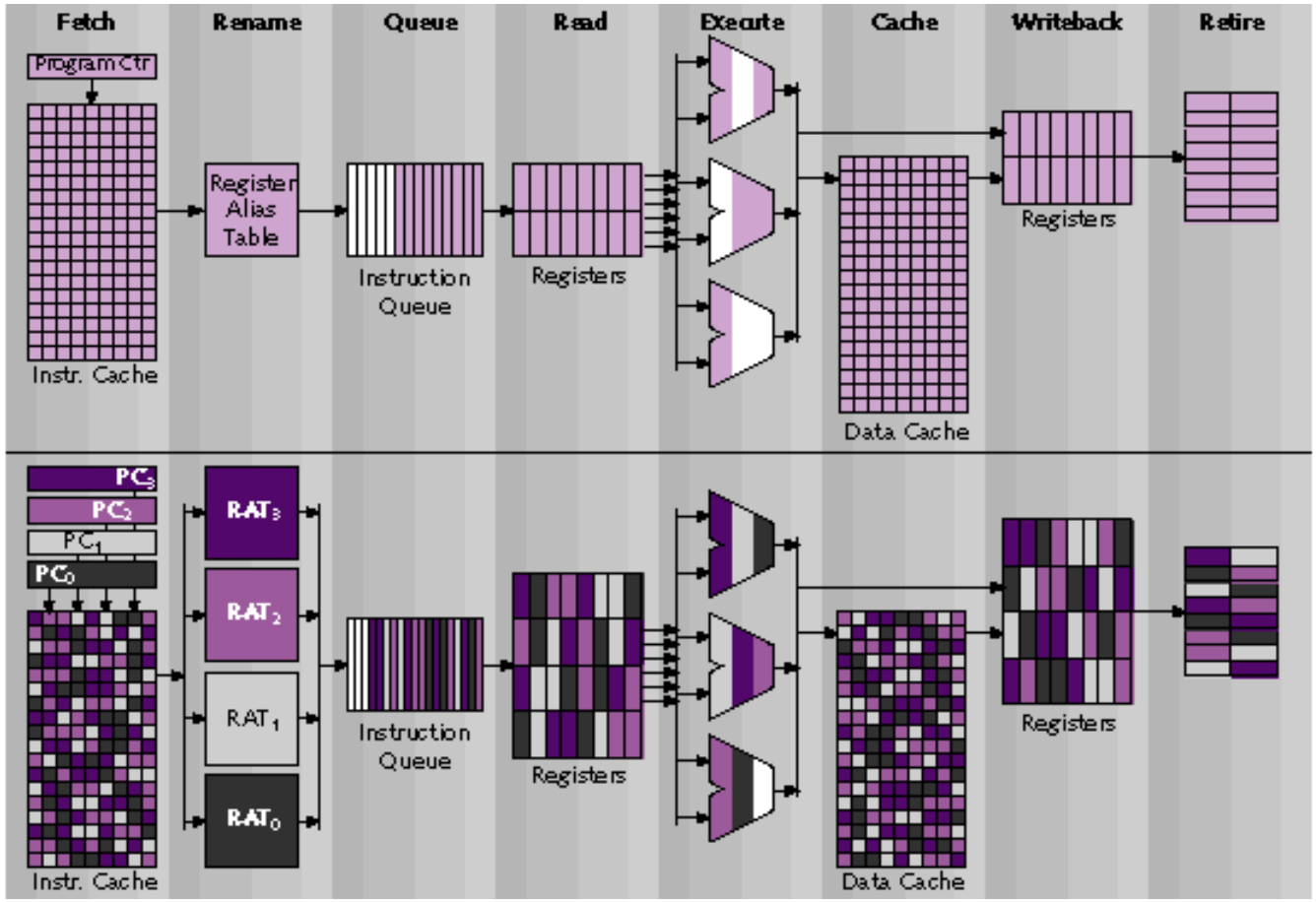
- Mehrfädige Prozessortechnik
  - Gegeben mehrere ausführbereite Kontrollfäden, Threads
  - Ziel: Parallele Ausführung mehrerer Kontrollfäden
  - Voraussetzung:
    - Mehrere Kontrollfäden sind geladen
    - Kontext muss für jeden Thread gesichert werden können
    - Mehrere getrennte Registersätze auf Prozessorchip
    - Mehrere Befehlszähler
    - Getrennte Seitentabellen
    - Threadwechsel, wenn gewartet werden muss

- Mehrfädige Prozessortechnik
  - Cycle-by-cycle Interleaving (feingranulares Multithreading)
    - Eine Anzahl von Kontrollfäden ist geladen.
    - Der Prozessor wählt in jedem Takt einen der ausführungsbereiten Kontrollfäden aus.
    - Der nächste Befehle in der Befehlsreihenfolge des ausgewählten Kontrollfadens wird zur Ausführung ausgewählt.
    - Beispiele
      - Multiprozessorsysteme HEP, Tera
    - Nachteil:
      - Die Verarbeitung eines Threads kann erheblich verlangsamt werden, wenn er ohne Wartezeiten ausgeführt werden kann

- Mehrfädige Prozessortechnik
  - Block Interleaving
    - Befehle eines Kontrollfadens werden so lange ausgeführt, bis eine Instruktion mit einer langen Latenzzeit ausgeführt wird. Dann wird zu einem anderen ausführbaren Kontrollfaden gewechselt.
    - Vorteil:
      - Die Bearbeitung eines Threads wird nicht verlangsamt, da beim Warten ausführungsbereiter Thread gestartet wird
    - Nachteil:
      - Bei Thread-Wechsel Leeren und Neustarten der Pipeline,
      - Nur bei langen Wartezeiten sinnvoll

- Mehrfädige Prozessortechnik
  - Simultaneous Multithreading
    - Mehrfach superskalarer Prozessor
    - Die Ausführungseinheiten werden über *eine* Zuordnungseinheit aus mehreren Befehlspuffern versorgt.
    - Jeder Befehlspuffer stellt einen anderen Befehlsstrom dar.
    - Jedem Befehlsstrom ist eigener Registersatz zugeordnet.

- Mehrfädige Prozessortechnik
  - Simultaneous Multithreading



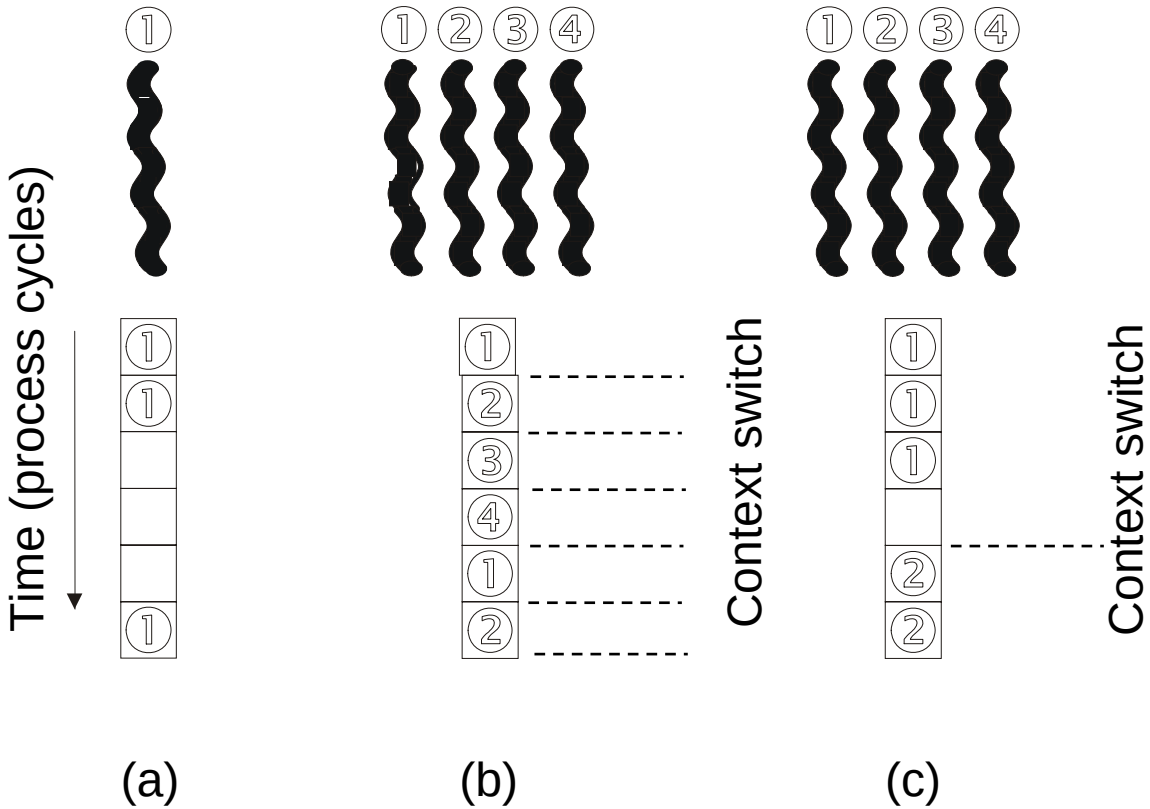
- Mehrfädige Prozessortechnik
  - Simultaneous Multithreading: Diskussion
    - Abwägen zwischen Geschwindigkeit eines Threads und dem Durchsatz vieler Threads
      - Ein bevorzugter Thread
        - » Allerdings kann dies auf Kosten des Durchsatzes gehen, da Befehle anderer Threads möglicherweise nicht bereit stehen
      - Mischen vieler Threads:
        - » Geht möglicherweise zu Lasten der Leistung der einzelnen Threads

- Mehrfädige Prozessortechnik
  - Simultaneous Multithreading: Beispiele
    - Compaq Alpha 21464 (EV8), ursprünglich angekündigt für 2002/2003, Entwicklung aber eingestellt! Entwicklergruppe jetzt bei Intel
    - Intel P4: Hyperthreading
    - Sun Ultra SPARC IV: Chip Multithreading

- Literatur:
  - Brinkschulte, U.; Ungerer, T.: Microcontroller und Mikroprozessoren. Springer, Heidelberg, 2002: Kap.: 10.4.3

Zusammenfassung

- Vergleich von Prozessortechniken



(a): single-threaded scalar

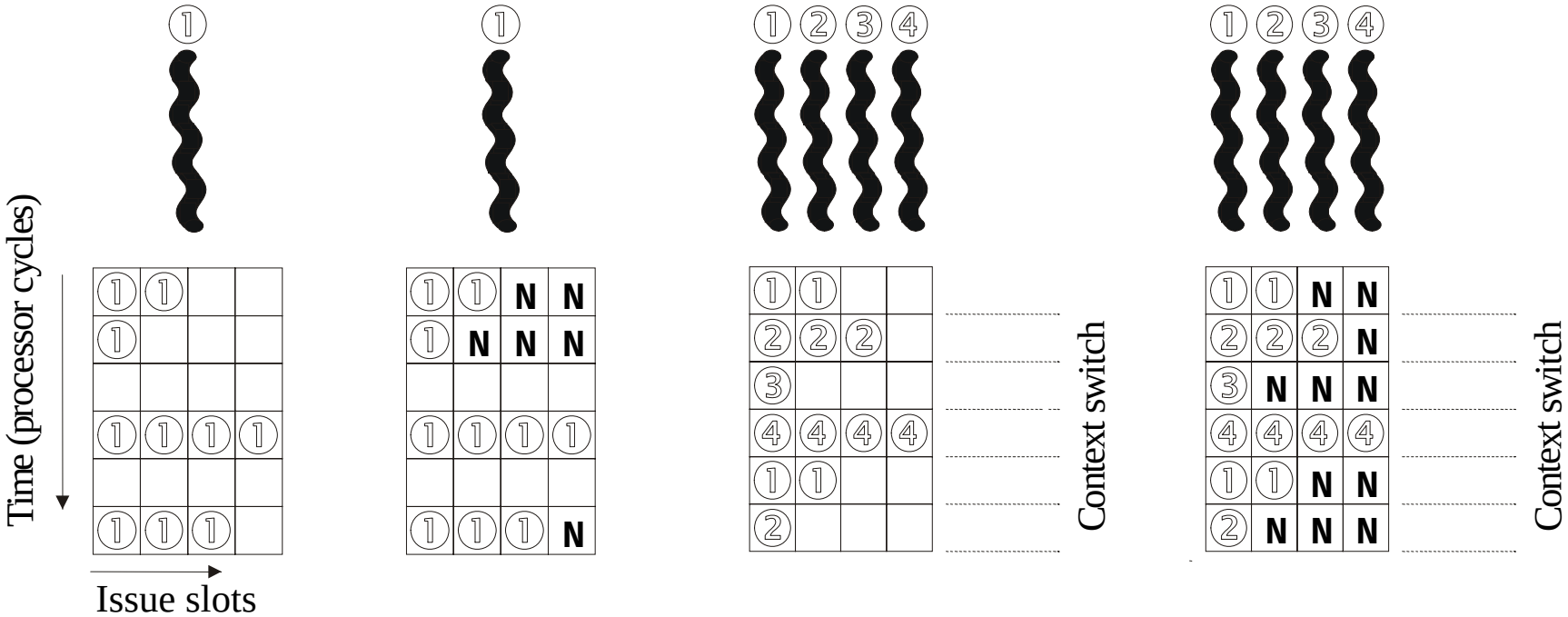
(b) cycle-by-cycle interleaving multithreaded scalar

(c) block interleaving multithreaded scalar

(siehe Brinkschulte, Ungerer: Mikrocontroller und Mikroprozessoren: Kap. 10.4.3)

Zusammenfassung

- Vergleich von Prozessortechniken



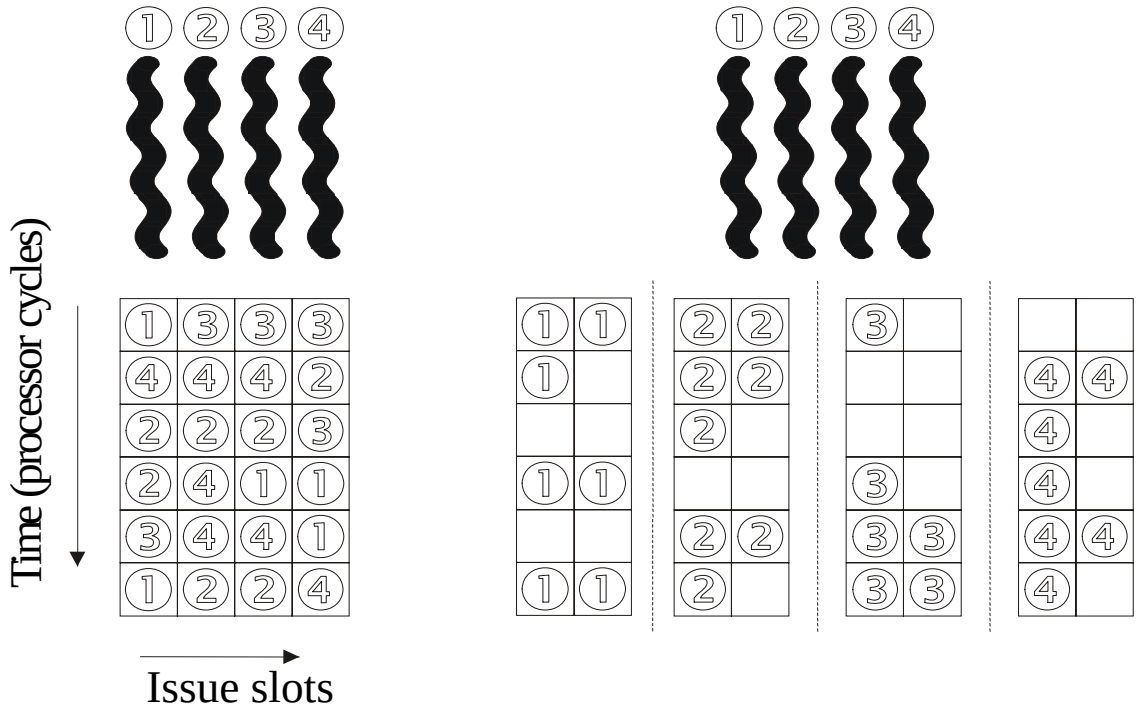
(a) Suparskalare Technik
(b) VLIW-Technik

(c) Cycle-by-cycle Interleaving
(d) Cycle-by-cycle VLIW

(siehe Brinkschulte, Ungerer: Mikrocontroller und Mikroprozessoren: Kap. 10.4.3)

Zusammenfassung

- Vergleich von Prozessortechniken



(a) Simultaneous Multithreading
(b) Single Chip Multiprocessing

(siehe Brinkschulte, Ungerer: Mikrocontroller und Mikroprozessoren: Kap. 10.4.3)

Zusammenfassung

- Architektur und Mikroarchitektur von Prozessoren
  - Architektur
  - RISC-Prinzip: Pipelining
  - Superskalare Prozessortechnik
  - VLIW/EPIC
  - Multithreading
  - Vergleich

Rechnerstrukturen

Vorlesung im Sommersemester 2008

Prof. Dr. Wolfgang Karl

Universität Karlsruhe (TH)

Fakultät für Informatik

Institut für Technische Informatik

- **Kapitel 3: Multiprozessoren – Parallelismus auf Prozess/Thread-Ebene**

3.1: Motivation

- Überblick
  - Allgemeine Grundlagen, parallele Programmierung, Verbindungsstrukturen, Leistungsfähigkeit
  - Speichergekoppelte Multiprozessoren: SMP und DSM, Cache-Kohärenz und Speicherkonsistenz, Rechnerbeispiele
  - Nachrichtengekoppelte Multiprozessoren, Beispielrechner

- Einordnung:
  - Klassifikation nach Flynn: MIMD-Rechner
- Warum Multiprozessorsysteme?
  - Hohe Anforderungen von Anwendungen an die Rechenleistung
  - Technisch-wissenschaftlicher Bereich
    - Rechnergestützte Simulation
  - Kommerzieller Bereich
    - Server, Datenbank-Anwendungen, WEB

- Motivation
  - Hohe Anforderungen von Anwendungen an die Rechenleistung
    - Beispiel technisch-wissenschaftlicher Bereich

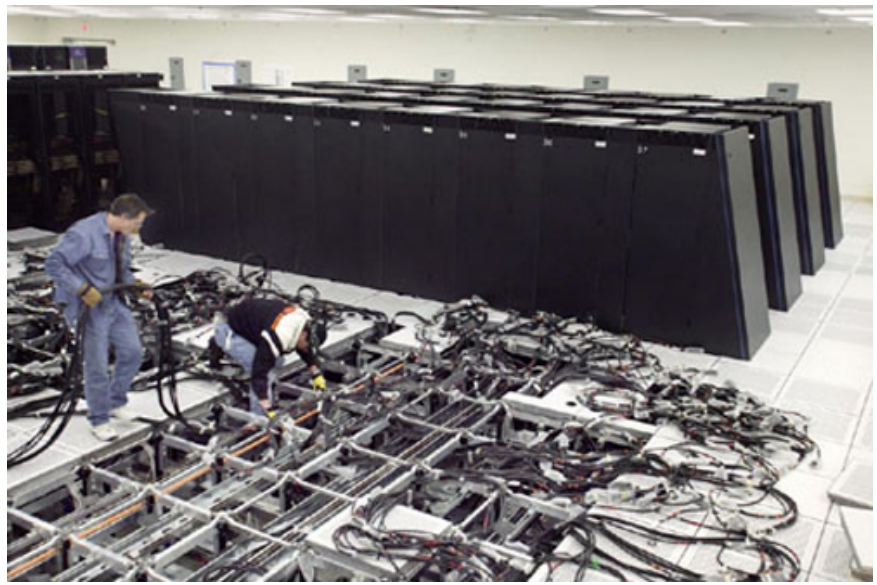
„Grand Challenges“
- Rechnergestützte Simulation
- Strömungsmechanik
- Modellierung der globalen klimatischen Veränderungen
- Evolution von Galaxien
- Struktur von Materialien
- .....

➔Anforderung an die Rechenleistung: Bereich Tera-, bzw. Petaflop

- Motivation
  - Hohe Anforderungen von Anwendungen an die Rechenleistung
  - Höchstleistungsrechner:
    - TOP500-Liste
      - Führt die schnellsten Rechner der Welt auf
      - Erscheint immer im Juni und im November eines Jahres
      - http://www.top500.org
        - » Beispiel: TOP500 Liste (November 2007)

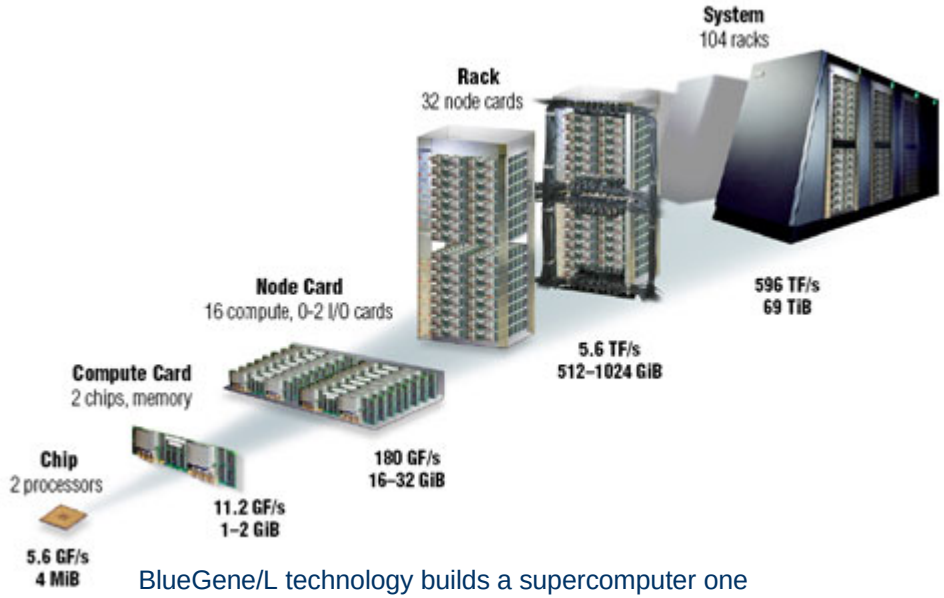
- Motivation
  - Höchstleistungsrechner:
    - TOP500-Liste
      - Nr.1: BlueGene/L, Modell: eServer Blue Gene Solution
      - Standort: DOE/NNSA/LLNL
      - Anzahl Prozessoren (PowerPC 440 700 MHz, 2.8 GFlops) : 212992
      - Speicher: 73728 GB
      - Leistung: 478200 GFLOPS (Linpack)
      - Installation: 2007

- Motivation
  - Höchstleistungsrechner:
    - Nr.1: BlueGene/L, Modell: eServer Blue Gene Solution

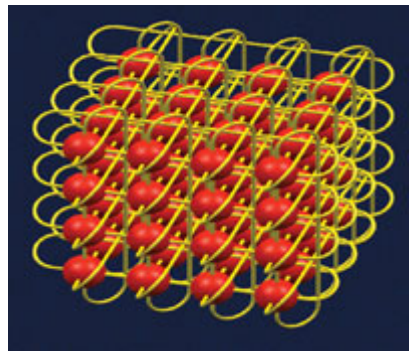


Quelle: https://asc.llnl.gov/computing_resources/bluegenel/photogallery.html

- Motivation
  - Höchstleistungsrechner:
    - Nr.1: BlueGene/L, Modell: eServer Blue Gene Solution



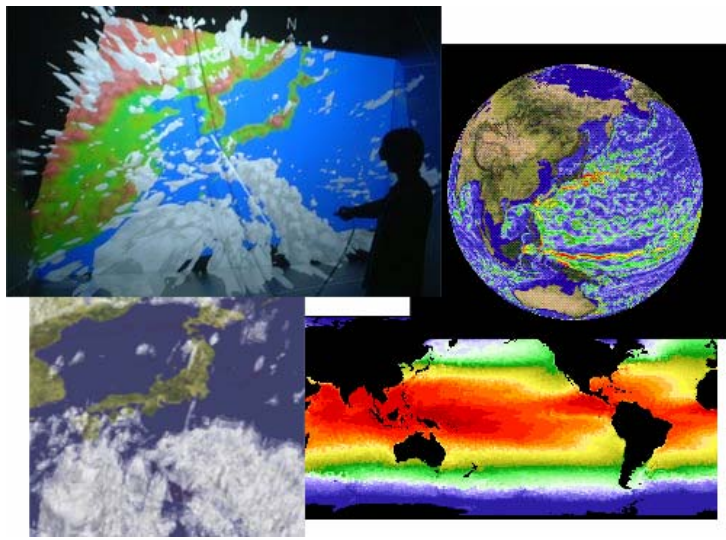
BlueGene/L technology builds a supercomputer one dual-processor chip at a time. Chips are aggregated into compute cards, which are then assembled into node cards. Each rack holds 2 node cards, and the full machine now comprises 104 racks.



BlueGene/L uses a three-dimensional (3D) torus network in which the nodes (red balls) are connected to their six nearest-neighbor nodes in a 3D mesh. In the torus configuration, the ends of the mesh loop back, thereby eliminating the problem of programming for a mesh with edges. Without these loops, the end nodes would not have six near neighbors.

https://asc.llnl.gov/computing_resources/bluegenel/configuration.html

- Motivation
  - Höchstleistungsrechner:
    - Earth Simulator (Japan, Platz 30 (TOP500, Nov. 07)
      - Anzahl Prozessoren: 5120
      - Leistung: 35,86 TFLOPS (Linpack),
      - Installation: 2002 (Nummer 1
      - Anwendung: Klimaforschung



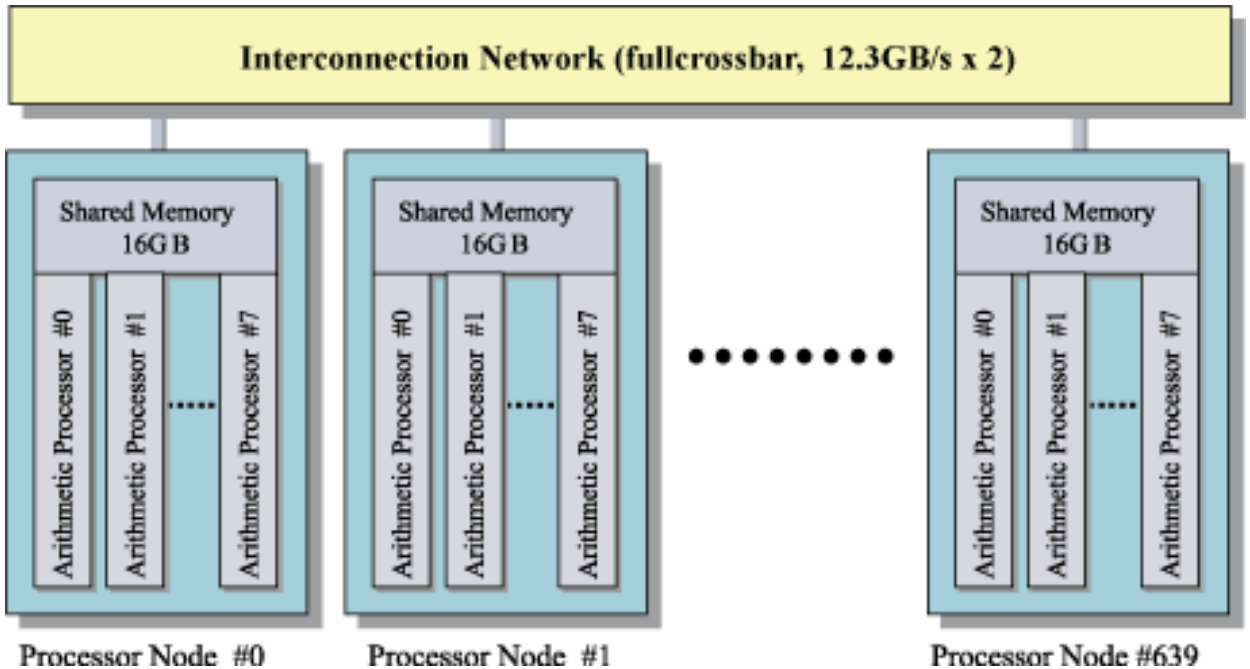
Quelle: The Earth Simulator Center;
http://www.es.jamstec.go.jp/esc/research/Perception/index.en.html

- Motivation
  - Höchstleistungsrechner:
    - Earth Simulator
      - Ziel des Earth Simulator Project:
        - » „The Earth Simulator Project will create a "virtual earth" on a supercomputer to show what the world will look like in the future by means of advanced numerical simulation technology.“
        - » „Achievement of high-speed numerical simulations with processing speed of 1000 times higher than that of the most frequently used supercomputers in 1996.“

- Motivation
  - Höchstleistungsrechner:
    - Earth Simulator
      - "Understanding and Prediction of Global Climate Change
        - » Occurrence prediction of meteorological disaster
        - » Occurrence prediction of El Niño
        - » Understanding of effect of global warming
        - » Establishment of simulation technology with 1km resolution"

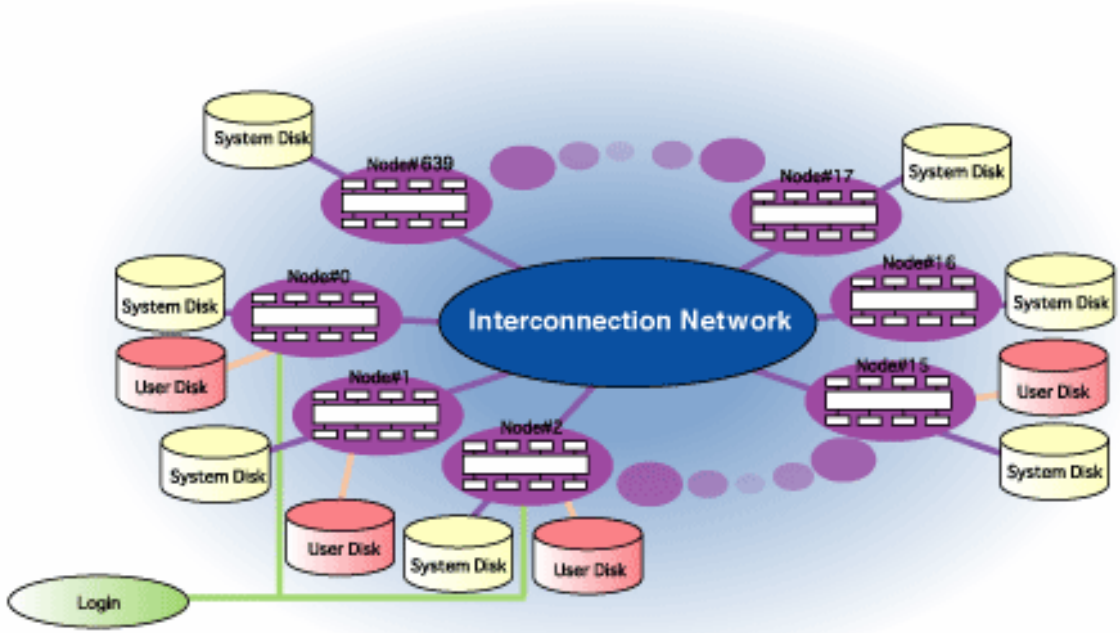
- Motivation
  - Höchstleistungsrechner:
    - Earth Simulator
      - "Understanding of Plate Techtonics
        - » Understanding of long-range crustal movements
        - » Understanding of mechanism of seismicity
        - » Understanding of migration of underground water and materials transfer in strata"

- Motivation
  - Höchstleistungsrechner:
    - Earth Simulator (Japan)
      - Systemkonfiguration



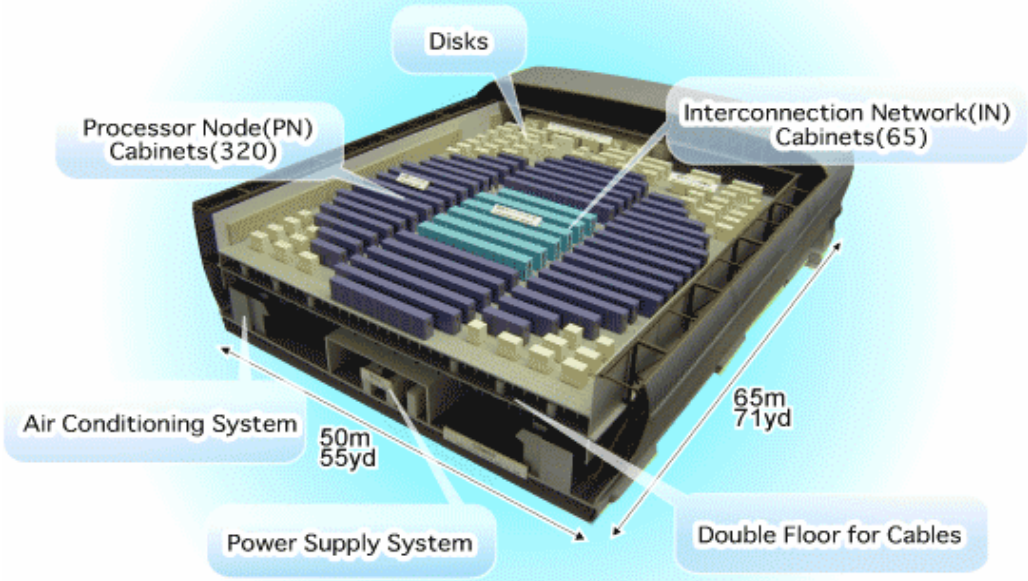
Quelle: The Earth Simulator Center;
http://www.es.jamstec.go.jp/esc/eng/Hardware/system.html

- Motivation
  - Höchstleistungsrechner:
    - Earth Simulator (Japan)



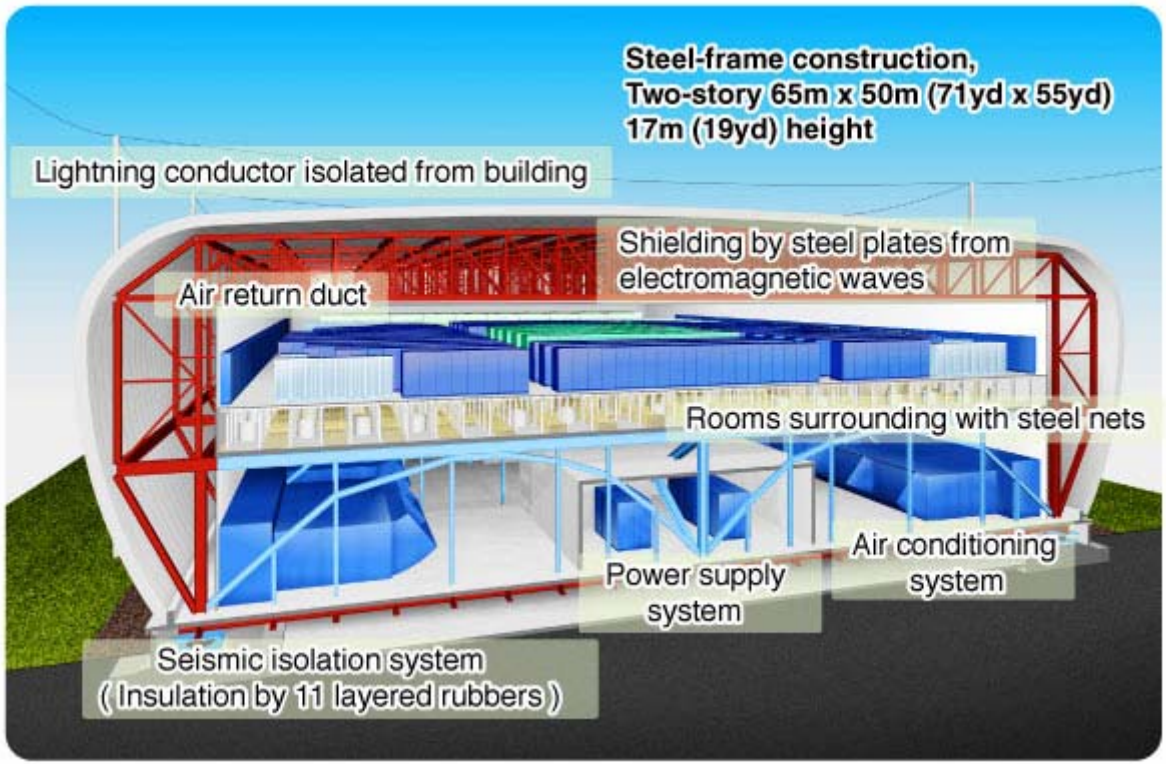
Quelle: The Earth Simulator Center;
http://www.es.jamstec.go.jp/esc/research/Perception/index.en.html

- Motivation
  - Höchstleistungsrechner:
    - Earth Simulator (Japan)



Quelle: The Earth Simulator Center;
http://www.es.jamstec.go.jp/esc/research/Perception/index.en.html

- Motivation
  - Höchstleistungsrechner:
    - Earth Simulator (Japan)

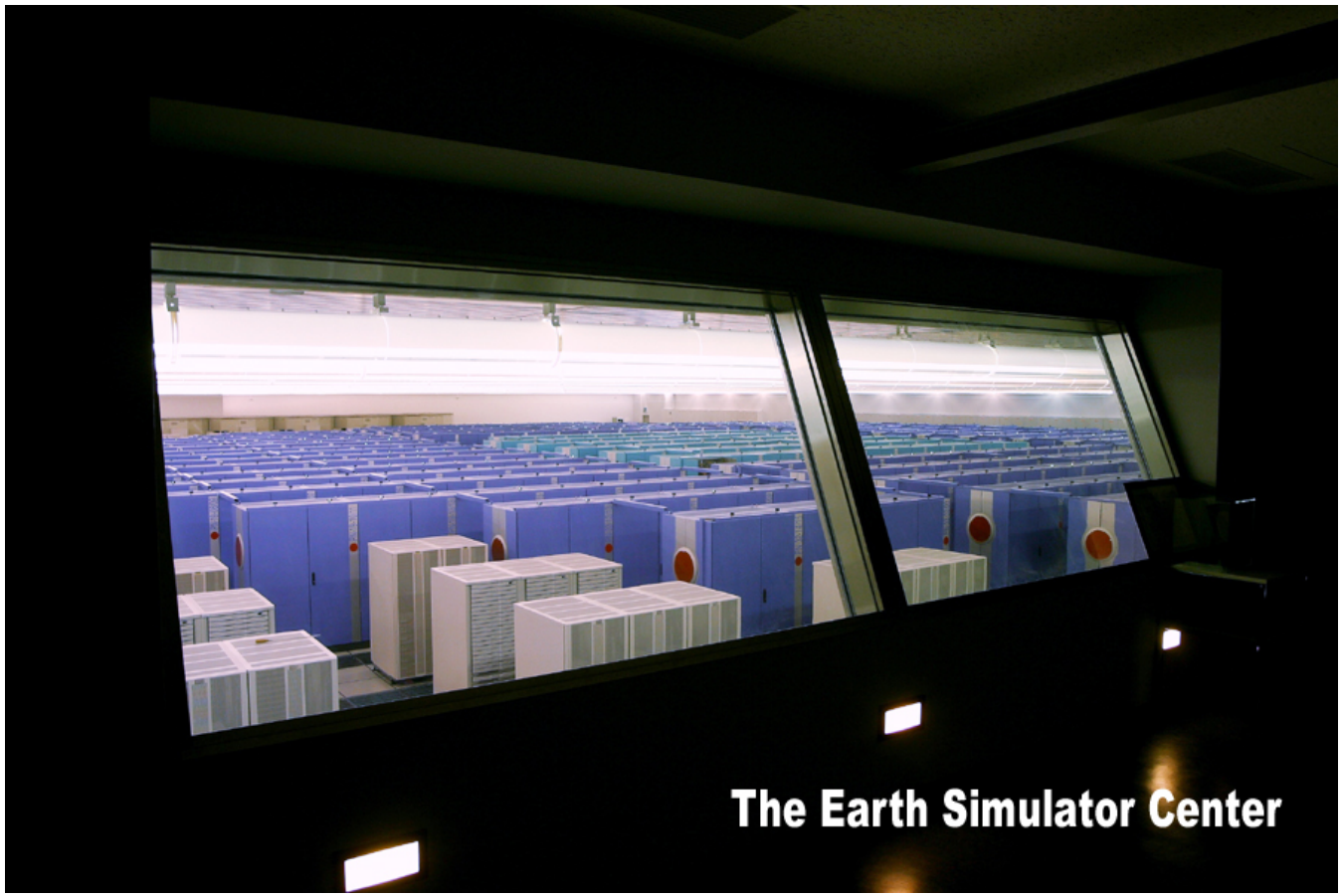


Quelle: The Earth Simulator Center;
http://www.es.jamstec.go.jp/esc/research/Perception/index.en.html

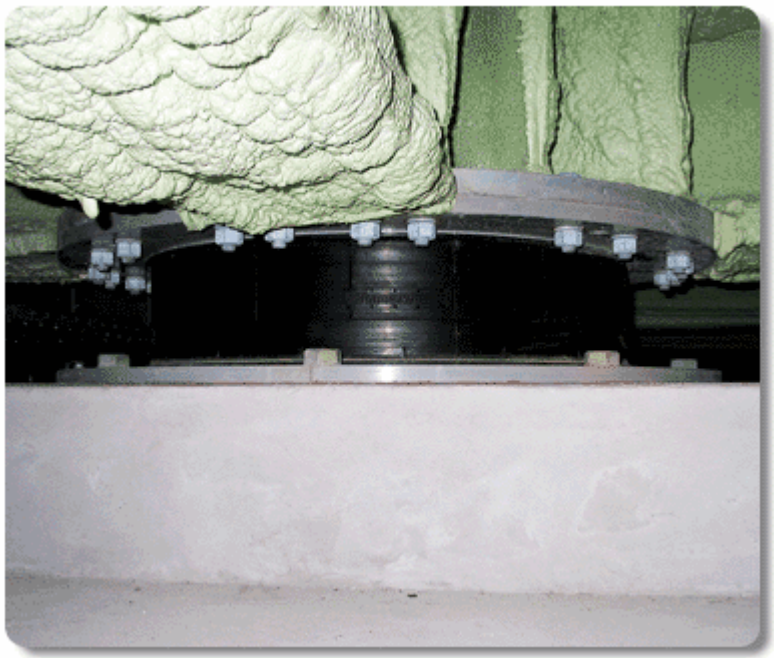
- Earth Simulator

Quelle: The Earth Simulator Center;
http://www.es.jamstec.go.jp/esc/eng/GC/index.html

- Earth Simulator



- Earth Simulator
  - Erdbebenschutz:



Quelle: The Earth Simulator Center;
http://www.es.jamstec.go.jp/esc/research/Perception/index.en.html

- **Kapitel 3: Multiprozessoren – Parallelismus auf Prozess/Thread-Ebene**

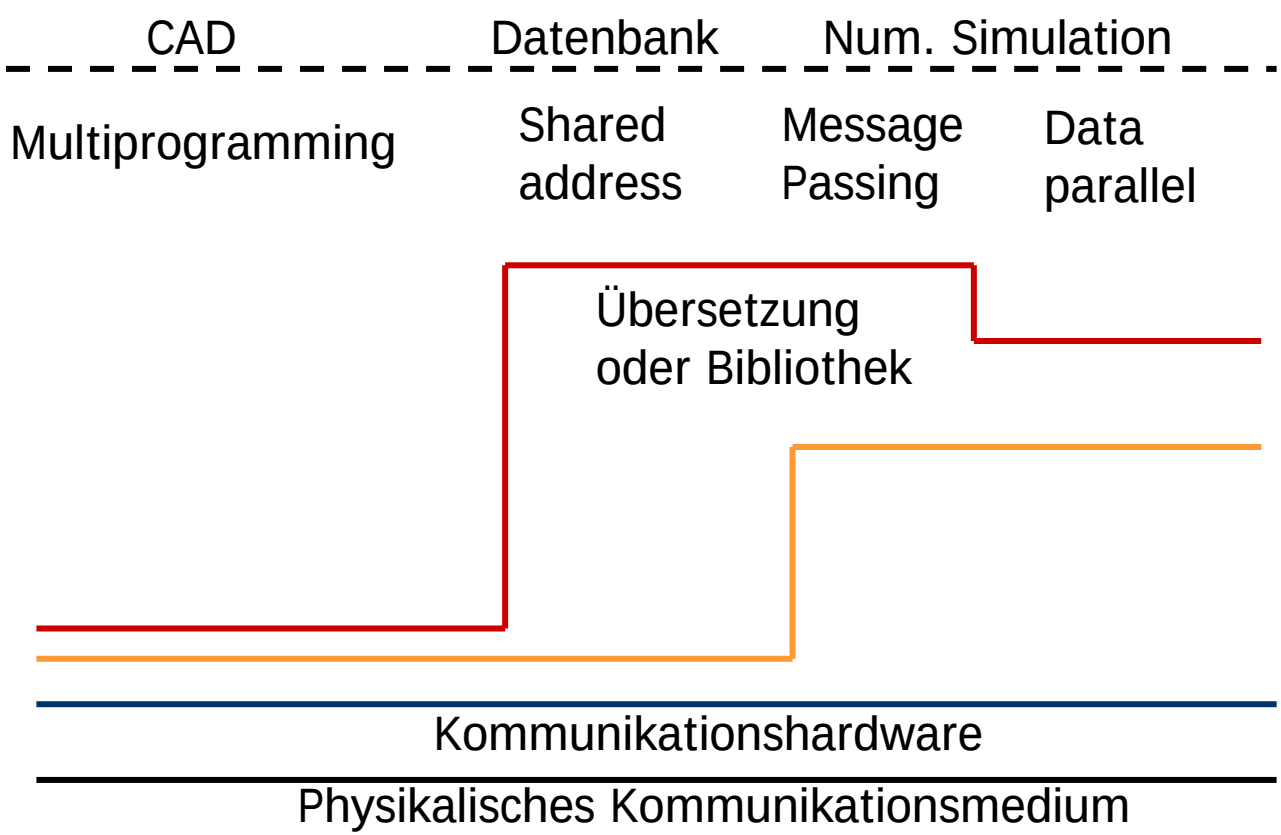
3.2: Allgemeine Grundlagen

- Parallele Architekturen
  - Definition:
    - Parallelrechner:
      - *„A collection of processing elements that communicate and cooperate to solve large problems“* (Almase and Gottlieb, 1989)
      - Betrachtung einer parallelen Architektur als eine Erweiterung des Konzepts einer konventionellen Rechnerarchitektur um eine Kommunikationsarchitektur

- Rechnerarchitektur
  - Abstraktion
    - Benutzer-/System-Schnittstelle
    - Hardware-/Software-Schnittstelle
  - Architektur
    - Spezifiziert die Menge der Operationen an den Schnittstellen und die Datentypen, auf denen diese operieren
  - Organisation
    - Realisierung der Abstraktionen

- Kommunikationsarchitektur
  - Abstraktion
    - Benutzer-/System-Schnittstelle
    - Hardware-/Software-Schnittstelle
  - Architektur
    - Spezifiziert die Kommunikations- und Synchronisationsoperationen
  - Organisation
    - Realisierung dieser Operationen

- Parallele Architekturen
  - Abstraktion



Parallele Anwendung

Programmiermodell

Kommunikations-abstraktion

Benutzer/System-Schnittstelle

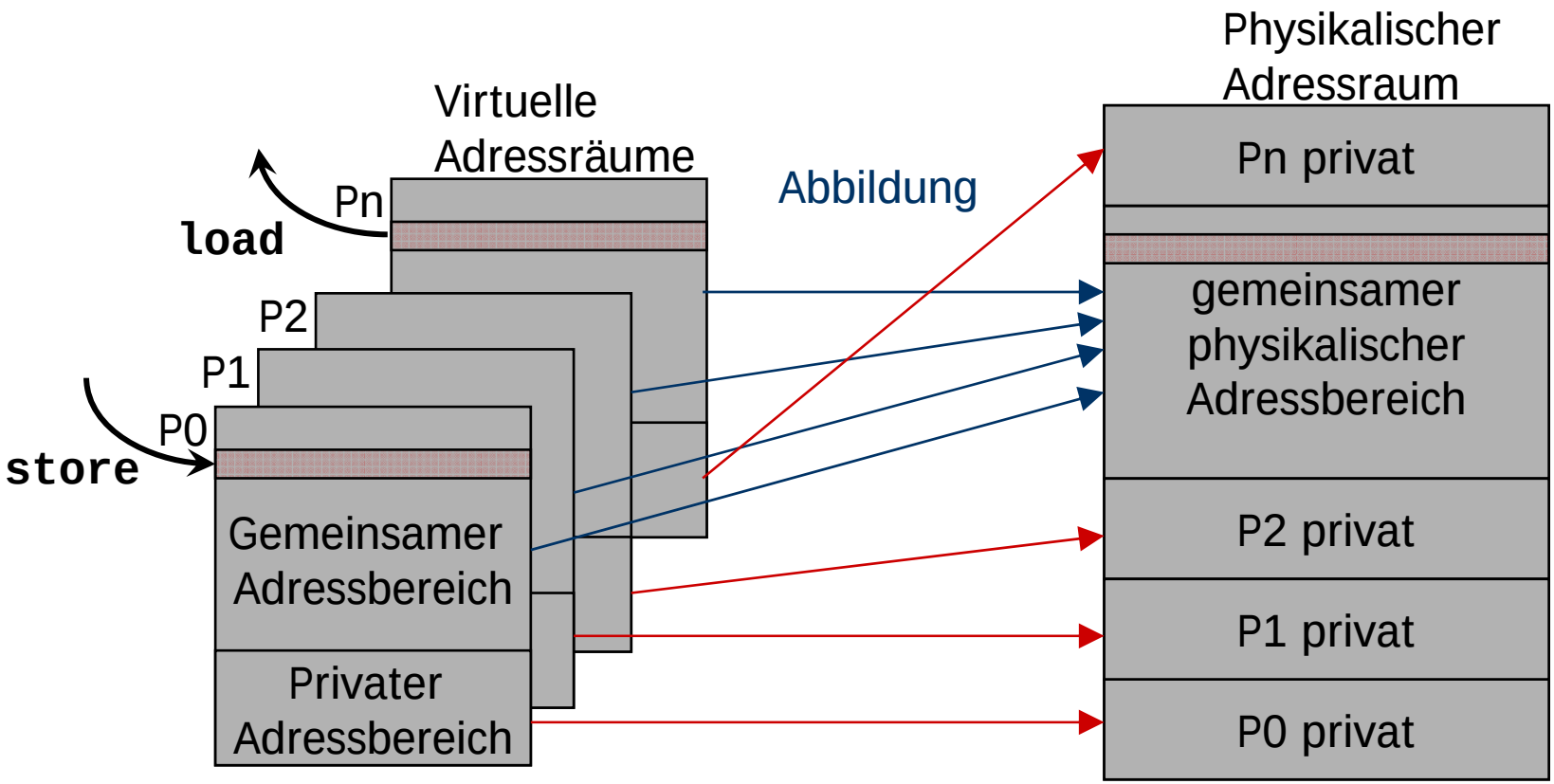
Hardware/Software-Schnittstelle

- Parallele Architekturen
  - Programmiermodell
    - Abstraktion einer parallelen Maschine, auf der der Anwender sein Programm formuliert
    - Spezifiziert, wie Teile des Programms parallel abgearbeitet werden, wie Informationen ausgetauscht werden und welche Synchronisationsoperationen verfügbar sind, um die Aktivitäten zu koordinieren
    - Anwendungen werden auf der Grundlage eines parallelen Programmiermodells formuliert

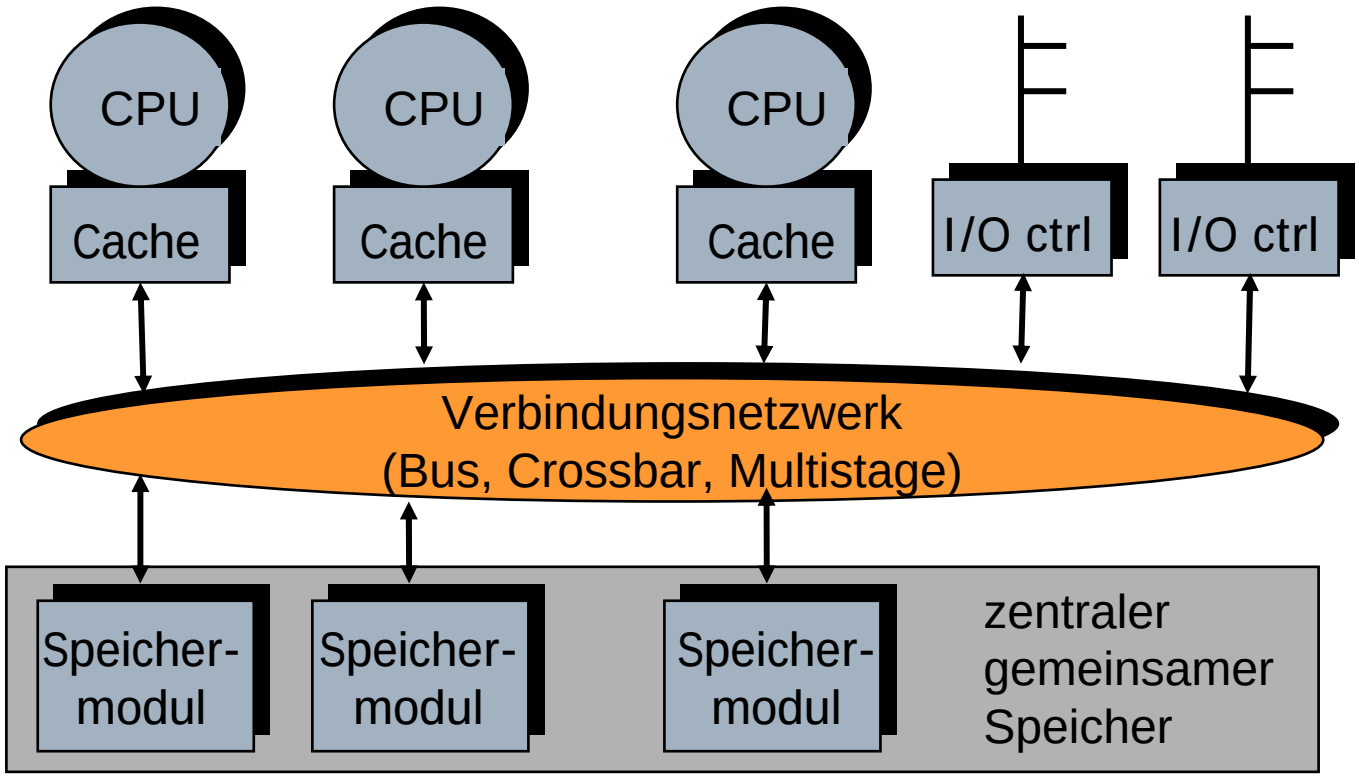
- Parallele Architekturen
  - Programmiermodell
    - Multiprogramming
      - Menge von unabhängigen sequentiellen Programmen
      - Keine Kommunikation oder Koordination

- Parallele Architekturen
  - Programmiermodell
    - Gemeinsamer Speicher (Shared Address Space)
      - Kommunikation und Koordination von Prozessen (Threads) über gemeinsame Variablen und Zeiger, die gemeinsame Adressen referenzieren
      - Kommunikationsarchitektur
        - » Verwendung konventioneller Speicheroperationen für die Kommunikation über gemeinsame Adressen
        - » Atomare Synchronisationsoperationen

- Parallele Architekturen
  - Gemeinsamer Speicher (Shared Address Space)

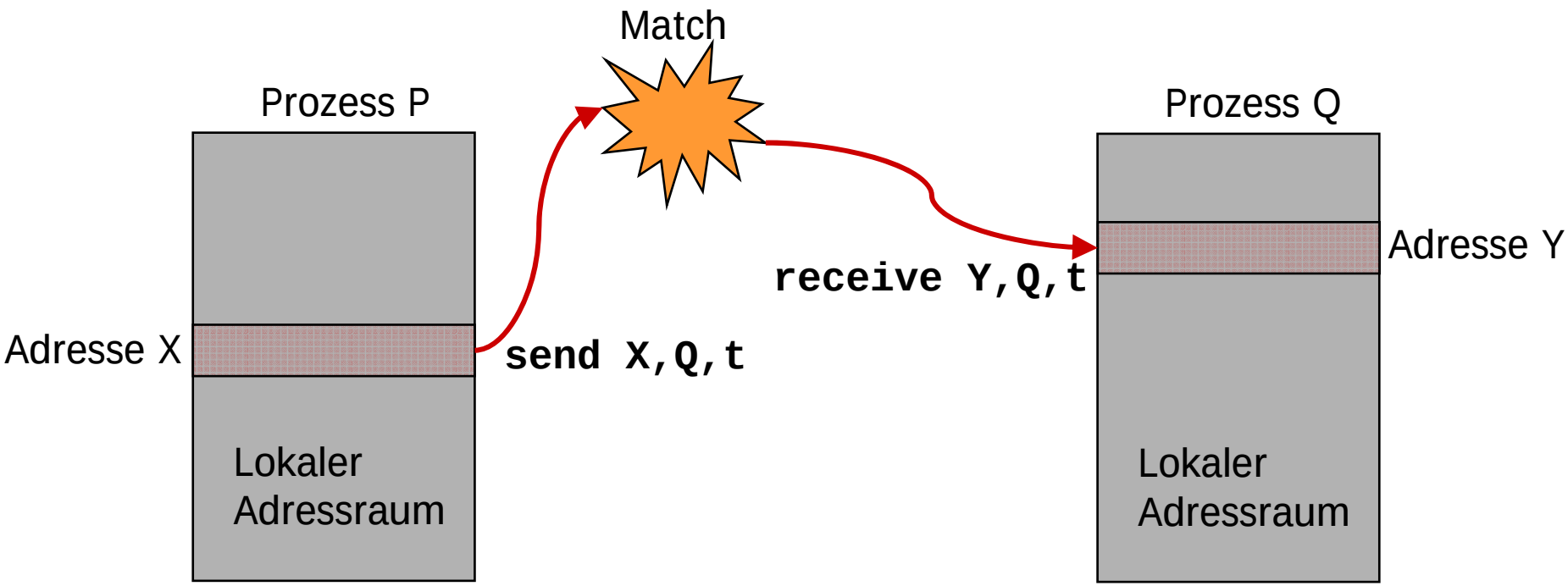


- Parallele Architekturen
  - Multiprozessor mit gemeinsamem Speicher

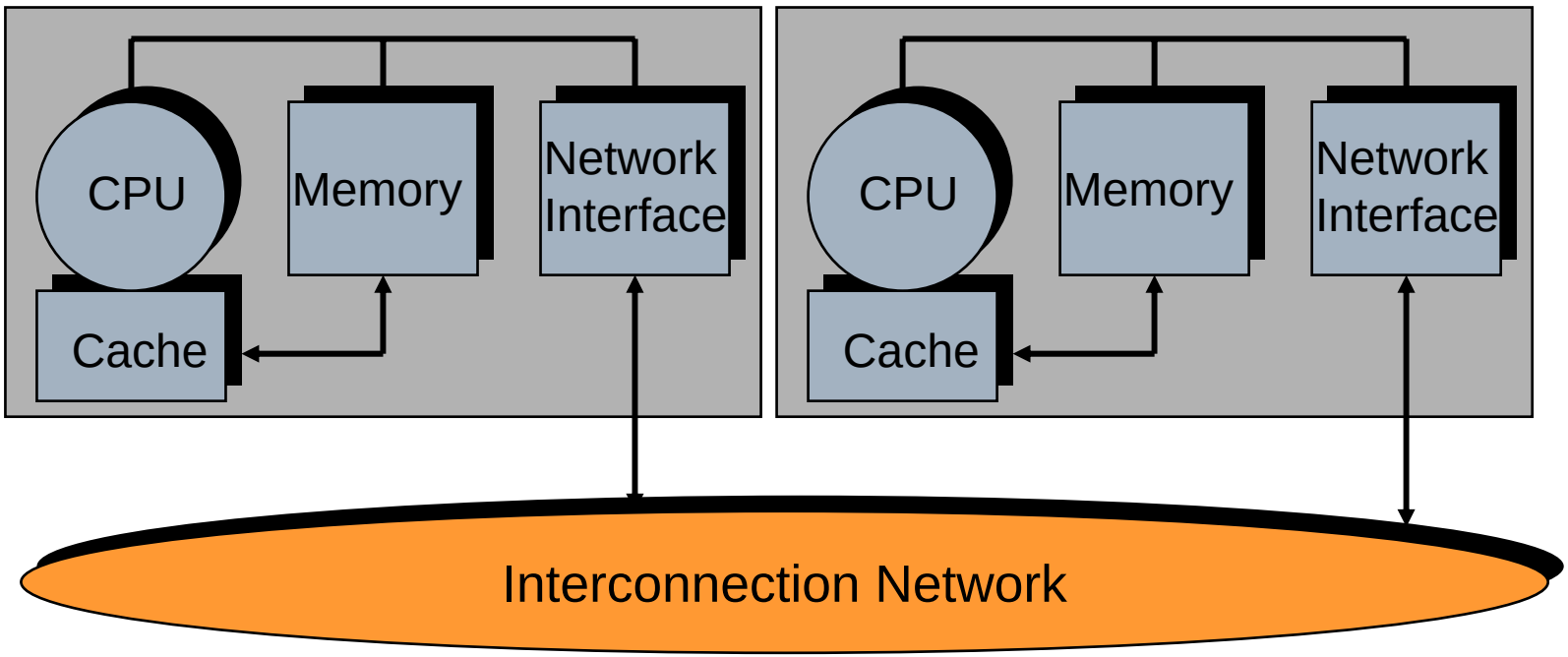


- Parallele Architekturen
  - Programmiermodell
    - Nachrichtenorientiertes Programmiermodell (Message Passing)
      - Kommunikation der Prozesse (Threads) mit Hilfe von Nachrichten
        - » Kein gemeinsamer Adressbereich
      - Kommunikationsarchitektur
        - » Verwendung von korrespondierenden Send- und Receive-Operationen
        - » Send: Spezifikation eines lokalen Datenpuffers und eines Empfangsprozesses (auf einem entfernten Prozessor)
        - » Receive: Spezifikation des Sende-Prozesses und eines lokalen Datenpuffers, in den die Daten ankommen

- Parallele Architekturen
  - Nachrichtenorientiertes Programmiermodell (Message Passing)



- Parallele Architekturen
  - Multiprozessor mit verteiltem Speicher



- Parallele Architekturen
  - Programmiermodell
    - Datenparallelismus
      - Gleichzeitige Ausführung von Operationen auf getrennten Elementen einer Datenmenge (Feld, Matrix)
      - Typischerweise in Vektorprozessoren

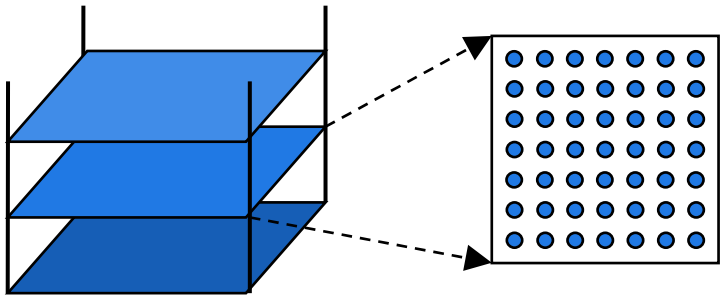
- **Kapitel 3: Multiprozessoren – Parallelismus auf Prozess/Thread-Ebene**

3.3: Parallele Programmierung

- Fallstudie: OCEAN - Simulation der Ozean-Strömung
  - Benchmark-Programm aus der SPLASH-Benchmark-Suite
  - Modellierung des Erdklimas
    - Gegenseitige Beeinflussung der Atmosphäre und der Ozeane, die ¾ der Erdoberfläche ausmachen
  - Simulation der Bewegung der Wasserströmung im Ozean
    - Strömung entwickelt sich unter dem Einfluss mehrerer physikalischer Kräfte, einschließlich atmosphärischer Effekte, dem Wind und der Reibung am Grund des Ozeans;
    - Vertikale Reibung an den „Rändern“: führt zu Wirbelströmung
    - Ziel: Simulation dieser Wirbelströme über der Zeit

- Fallstudie: OCEAN - Simulation der Ozean-Strömung
  - Grundsätzliche Probleme:
    - Gute Modelle für die Beschreibung des Verhaltens sind kompliziert:
      - Vorhersage des Zustands eines Ozeans zu einem Zeitpunkt erfordert die Lösung eines komplexen Gleichungssystems
      - Nur mit numerischen Verfahren möglich
    - Das physikalische Problem ist kontinuierlich über Raum und Zeit:
      - Diskretisierung über beide Dimensionen

- Fallstudie: OCEAN - Simulation der Ozean-Strömung
  - Diskretisierung:
    - Raum:
      - Modellierung des Ozeansbeckens als ein Gitter von diskreten Punkten
      - Jede Variable (Druck, Geschwindigkeit, ...) hat einen Wert an jedem Gitterpunkt
      - Zweidimensionales Gitter:



Modellierung des Ozeans
in einem rechteckigen Becken

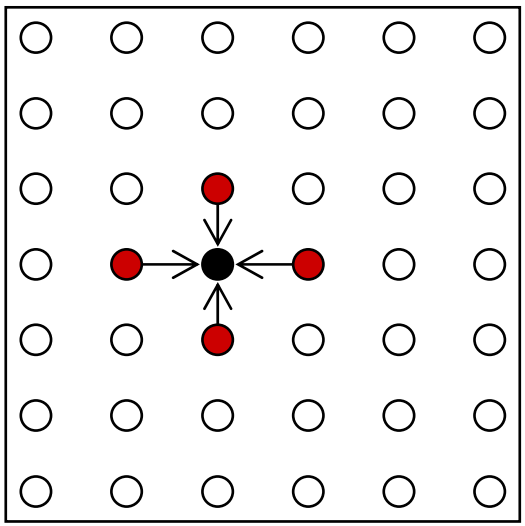
    - Zeit:
      - Endliche Folge von Zeitschritten

- Fallstudie: OCEAN - Simulation der Ozean-Strömung
  - Lösung der Bewegungsgleichungen:
    - An allen Gitterpunkten in einem Zeitschritt
    - In jedem Zeitschritt werden die Variablen neu berechnet
    - Wiederholung der Berechnung mit jedem Zeitschritt
    - Jeder Zeitschritt besteht aus mehreren Phasen

- Fallstudie: OCEAN - Simulation der Ozean-Strömung
  - Lösung der Bewegungsgleichungen:
    - Je mehr Gitterpunkte verwendet werden, desto feiner ist die Auflösung der Diskretisierung und desto genauer ist die Simulation
    - Für einen Ozean wie den Atlantik, der etwa eine Fläche von 2000km x 2000km umspannt bedeutet ein Gitter mit 100 x 100 Punkten eine Distanz von 20 km in jeder Dimension
    - Kürzere physikalische Intervalle zwischen den Zeitschritten führen zu einer höheren Simulationsgenauigkeit
    - Simulation der Ozeanbewegung über einen Zeitraum von 5 Jahren mit einer Aktualisierung des Zustands alle 8 Stunden erfordert 5500 Zeitschritte

- Fallstudie: OCEAN - Simulation der Ozean-Strömung
  - Vereinfachung: Parallelisierung des Gleichungslösers
    - Lösung einer einfachen partiellen Differentialgleichung auf einem Gitter mit Hilfe einer finiten Differenzenmethode (Gauss-Seidel-Verfahren)
    - Reguläres zweidimensionales Gitter mit *(n+2)*(n+2)* Punkten (eine Ebene des Ozeanbeckens
    - Randwerte sind fest
    - Die inneren *n*n* Gitterpunkte werden mit Hilfe des Lösers berechnet, ausgehend von Anfangswerten

- Fallstudie: OCEAN - Simulation der Ozean-Strömung
  - Vereinfachung: Parallelisierung des Gleichungslösers
    - Gitter:



Berechnungsvorschrift für einen Gitterpunkt:

```
A[i,j]=0,2 x (A[i,j]+
             +A[i,j-1]+A[i-1,j]
             +A[i,j+1]+A[i+1,j]
```

Wiederholte Berechnung, bis Verfahren konvergiert

- Sequentielle Version des Lösers:

```
(1) int n,                                  /*size of matrix: (n+2)x(n+2)
(2) float **A, diff=0;
(3) main()
(4) begin
(5)    read(n)                              /*read input parameters*/
(6)    A←malloc (2-d array of size n+2 by n+2 doubles)
(7)    initialize(A);                              /*initialize matrix A*/
(8)    Solve (A);
(9)  end main
```

- Sequentielle Version des Lösers:

```
(1) procedure Solve (A)                    /*solve equation system*/
(2)   float **A;
(3) begin
(4)   int i,j,done=0
(5)   float temp;
(6)   while (!done) do                     /*outermost loop over sweeps*/
(7)     diff=0;                            /*initialize maximum diff*/
(8)     for i←1 to n do                    /*sweep over nonborder points*/
(9)       for j←1 to n do
(10)        temp=A[i,j];
(11)        A[i,j]←0.2*(A[i,j]+A[i,j-1]+A[i-1,j]+A[i,j+1]+A[i+1,j]);
(12)        diff += abs(A[i,j]–temp);
(13)      end for
(14)    end for
(15)    if (diff/(n*n) < TOL) then done=1;
(16)  end while
(17) end procedure
```

Parallele Programmierung

- Parallelisierungsprozess
  - Festlegen der Aufgaben, die parallel ausgeführt werden kann
    - Aufteilen der Aufgaben und der Daten auf Verarbeitungsknoten
      - Berechnung
      - Datenzugriff
      - Ein-/Ausgabe
    - Verwalten des Datenzugriffs, der Kommunikation und der Synchronisation
  - Ziel: Hohe Leistung
    - Schnellere Lösung der parallelen Version gegenüber der sequentiellen Version
      - Ausgewogene Verteilung der Arbeit unter den Verarbeitungsknoten
      - Reduzierung des Kommunikations- und Synchronisationsaufwandes

- Parallelisierungsprozess
  - Ausführung der Schritte bei der Parallelisierung
    - Durch den Programmierer
    - Auf den verschiedenen Ebenen der Systemsoftware
      - Compiler
      - Laufzeitsystem
      - Betriebssystem
    - Ideal:
      - Automatische Parallelisierung
      - Sequentielles Programm wird automatisch in ein effizientes paralleles Programm transformiert
      - Parallelisierende Compiler
      - Parallele Programmiersprachen
      - Noch nicht vollständig möglich!

- Parallelisierungsprozess
  - Definitionen
    - Task
      - Beliebige Aufgabe, die durch ein Programm auszuführen ist
      - Kleinste Parallelisierungseinheit
      - Möglichkeiten beim Beispiel Ocean:
        - » Berechnung eines Gitterpunkts in jeder Berechnungsphase,
        - » die Berechnung einer Reihe von Gitterpunkten,
        - » die Berechnung einer beliebigen Teilmenge von Gitterpunkten
      - Granularität
        - » grobkörnig
        - » feinkörnig

- Parallelisierungsprozess
  - Definitionen
    - Prozess oder Thread
      - Paralleles Programm setzt sich aus mehreren kooperierenden Prozessen zusammen, von denen jeder eine Teilmenge der Tasks ausführt
      - Tasks werden über Prozessen zugewiesen
      - Beispiel Ocean:
        - » Falls die Berechnung einer Reihe von Gitterpunkten als Task angesehen wird, dann kann eine feste Anzahl von Reihen einem Prozess zugewiesen werden
        - » Aufteilung einer Ebene in mehrere Streifen
      - Kommunikation der Prozesse untereinander und Synchronisation
    - Prozessor
      - Ausführung eines Prozesses

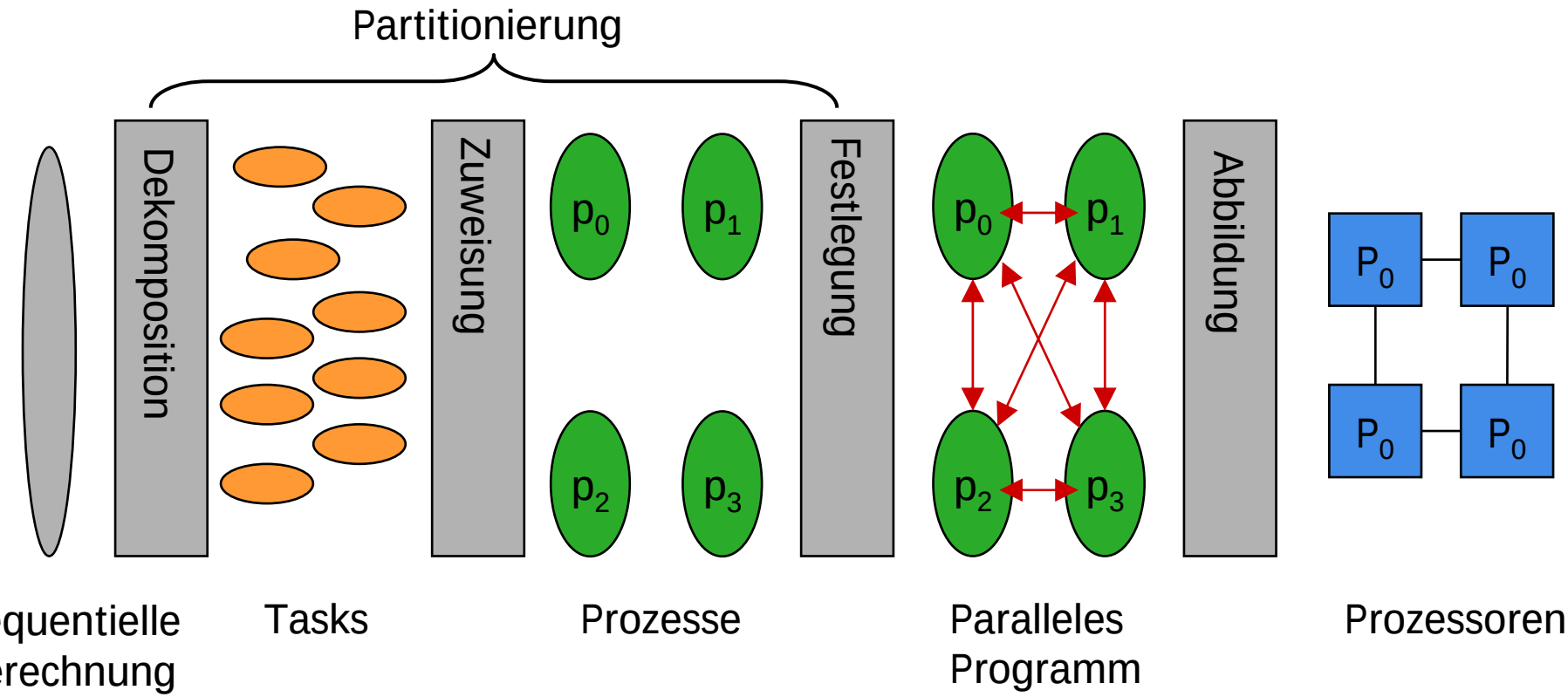
- Parallelisierungsprozess
  - Definitionen
    - Unterscheidung Prozess und Prozessor
      - Prozessor:
        - » Physikalische Ressource
      - Prozess
        - » Abstraktion, Virtualisierung von einem Multiprozessor
        - » Anzahl der Prozesse muss nicht gleich der Anzahl der Prozessoren eines Multiprozessorsystems sein

- Parallelisierungsprozess
  - Schritte bei der Parallelisierung
    - Ausgangspunkt ist ein sequentielles Programm
    - Aufteilung oder Dekomposition
      - der Berechnung in Tasks
    - Zuweisung
      - der Tasks zu Prozessen

    } Partitionierung

    - Festlegung (Orchestration)
      - des notwendigen Datenzugriffs, der Kommunikation und der Synchronisation zwischen den Prozessen
    - Abbildung
      - der Prozesse an die Prozessoren

- Parallelisierungsprozess
  - Schritte bei der Parallelisierung und die Beziehung zwischen Tasks, Prozessen und Prozessoren



- Parallelisierungsprozess
  - Dekomposition
    - Aufteilung der Berechnung in eine Menge von Tasks
      - Tasks können dynamisch während der Ausführung generiert werden
      - Anzahl der Tasks kann während der Ausführung variieren
      - Maximale Anzahl der Tasks, die zu einem Zeitpunkt zur Ausführung verfügbar sind, ist eine obere Grenze für die Anzahl der Prozesse, die effektiv genutzt werden können
    - Ziel:
      - Finden von parallel ausführbaren Anteilen
      - Verwaltungsaufwand (Overhead) gering halten

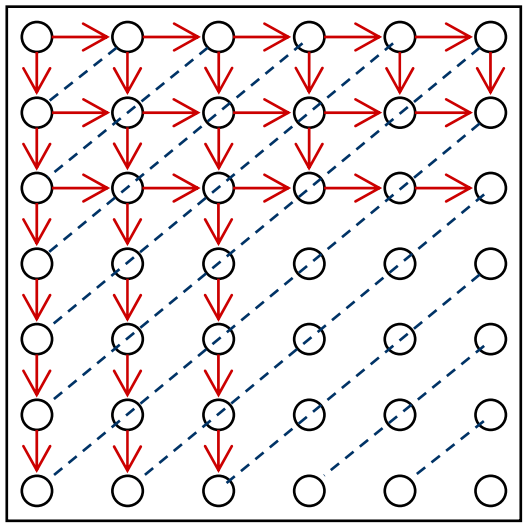
- Parallelisierungsprozess
  - Dekomposition
    - Beispiel: Ocean
      - Programm strukturiert in geschachtelten Schleifen
        - » Betrachtung einzelner Schleifen oder der geschachtelten Schleifen
        - » Können Iterationen parallel ausgeführt werden?
        - » Betrachtung über Schleifengrenzen

- Parallelisierungsprozess
  - Dekomposition
    - Beispiel: Ocean
      - Betrachtung einzelner Schleifen oder der geschachtelten Schleifen

```
(1)  while (!done) do
(2)      diff=0;
(3)      for i←1 to n do
(4)         for j←1 to n do
(5)            temp=A[i,j];
(6)            A[i,j]←0.2*(A[i,j]+A[i,j-1]+A[i-1,j]+A[i,j+1]+A[i+1,j]);
(7)            diff += abs(A[i,j]–temp);
(8)         end for
(9)       end for
(10)      if (diff/(n*n) < TOL) then done=1;
(11)   end while
```

- Parallelisierungsprozess
  - Dekomposition
    - Beispiel: Ocean
      - Betrachtung einzelner Schleifen oder der geschachtelten Schleifen
        - » Äußere Schleife (Zeile 1-11) durchläuft das gesamte Gitter → Iterationen sind nicht unabhängig, da Daten, die in einer Iteration geändert werden, in der nächsten Iteration gebraucht werden
        - » Innere Schleifen (Zeile 3-9)→Iterationen sind sequentiell abhängig, da in jeder inneren Schleife A[i,j-1] gelesen wird, der in der vorherigen Iteration geschrieben wurde

- Parallelisierungsprozess
  - Dekomposition
    - Beispiel: Ocean
      - Betrachtung der Abhängigkeiten (Granularität Gitterpunkte)



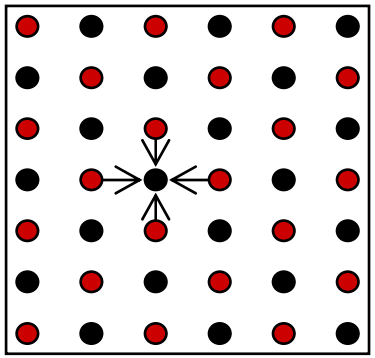
Abhängigkeiten

Verbinden Punkte, zwischen den keine Abhängigkeiten bestehen

- Parallelisierungsprozess
  - Dekomposition am Beispiel: Ocean
    - Aufteilen der Arbeit in einzelne Gitterpunkte, so dass die Aktualisierung eines Gitterpunktes eine Task ist
    - 1. Möglichkeit:
      - Beibehalten der Schleifenstruktur
      - erfordert Punkt-zu-Punkt-Synchronisation wegen Beachtung der Abhängigkeiten
        - » Der neue Wert eines Gitterpunktes in einem Durchlauf ist berechnet, bevor er von dem westlichen oder südlichen Punkten verwendet wird
        - » Verschiedene Schleifenschachtelungen und verschiedene Durchläufe können gleichzeitig ausgeführt werden
      - Hoher Aufwand!

- Parallelisierungsprozess
  - Dekomposition am Beispiel: Ocean
    - Aufteilen der Arbeit in einzelne Gitterpunkte, so dass die Aktualisierung eines Gitterpunktes eine Task ist
    - 2. Möglichkeit:
      - Ändern der Schleifenstruktur
      - Erste FOR-Schleife (Zeile3) geht über Anti-Diagonale und die innere Schleife geht über die Elemente der Anti-Diagonalen
        - » Parallele Ausführung der inneren Schleife
        - » Globale Synchronisation zwischen den Interationen der äußeren Schleife
      - Hoher Aufwand
        - » Globale Synchronisation findet immer noch häufig statt
      - Lastungleichheit
        - » wegen unterschiedlich vielen Elementen auf den Anti-Diagonalen

- Parallelisierungsprozess
  - Dekomposition am Beispiel Ocean
    - 3. Möglichkeit: Red-Black



» Durchlauf über ein Gitter wird in zwei Phasen aufgeteilt:
» Parallele Berechnung der $n^2/2$ roten Punkte
» Globale Synchronisation (konservativ)
» Berechnung der $n^2/2$ schwarzen Punkte
» Konvergiert mit mehr oder mit weniger Durchläufen
» Es können auch unterschiedliche Werte innerhalb der Toleranz berechnet werden

- Parallelisierungsprozess
  - Dekomposition am Beispiel: Ocean
    - 4. Möglichkeit:
      - Asynchrone Methode ohne Aufteilung in rote und schwarze Punkte
        - » Ignorieren der Abhängigkeiten zwischen den Gitterpunkten für einen Durchlauf
        - » Globale Synchronisation zwischen den Iterationen, aber keine Änderung der Durchlaufordnung
        - » Prozess aktualisiert alle Punkte, sequentielle Ordnung
        - » Punkte können auf mehrere Prozesse aufgeteilt werden, dann ist die Ordnung nicht vorhersagbar, sondern hängt von der Zuteilung der Punkte zu Prozessen, der Anzahl der Prozesse und wie schnell die verschiedenen Prozesse relativ zueinander während der Laufzeit ausgeführt werden, ab
        - » Ausführung ist nicht deterministisch!
        - » Anzahl der Durchläufe bis zur Konvergenz kann von der Anzahl der Prozesse abhängen

- Parallelisierungsprozess
  - Dekomposition am Beispiel: Ocean
    - 4. Möglichkeit
      - » Dekomposition in individuelle innere Schleifeniterationen
      - » for_all: weist darunter liegende HW/SW an, dass Schleifeiterationen parallel ausgeführt werden können

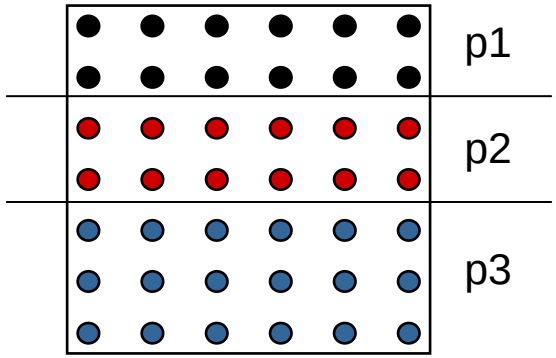
```
(1)  while (!done) do
(2)      diff=0;
(3)      for_all i←1 to n do
(4)         for_all j←1 to n do
(5)            temp=A[i,j];
(6)            A[i,j]←0.2*(A[i,j]+A[i,j-1]+A[i-1,j]+A[i,j+1]+A[i+1,j]);
(7)            diff += abs(A[i,j]–temp);
(8)         end for_all
(9)       end for_all
(10)      if (diff/(n*n) < TOL) then done=1;
(11)   end while
```

- Parallelisierungsprozess
  - Dekomposition am Beispiel: Ocean
    - 5. Möglichkeit
      - » Reihenorientierte Dekomposition

```
(1)  while (!done) do
(2)      diff=0;
(3)      for_all i←1 to n do
(4)         for j←1 to n do
(5)            temp=A[i,j];
(6)            A[i,j]←0.2*(A[i,j]+A[i,j-1]+A[i-1,j]+A[i,j+1]+A[i+1,j]);
(7)            diff += abs(A[i,j]–temp);
(8)         end for
(9)       end for_all
(10)      if (diff/(n*n) < TOL) then done=1;
(11)   end while
```

- Parallelisierungsprozess
  - Zuweisung
    - Spezifikation des Mechanismus, mit dessen Hilfe die Tasks auf Prozesse aufgeteilt werden
    - Ziel:
      - ausgewogene Lastverteilung: Lastbalanzierung
      - Reduzierung der Interprozess-Kommunikation
      - Reduzierung des Aufwands zur Laufzeit
    - Statische oder dynamische Zuweisung

- Parallelisierungsprozess
  - Zuweisung
    - Beispiel:



- Parallelisierungsprozess
  - Festlegung
    - Architektur und Programmiermodell sowie die Programmiersprache spielen eine Rolle
      - Um die zugewiesenen Tasks ausführen zu können, benötigen die Prozesse Mechanismen
        - » für den Zugriff auf die Daten,
        - » für die Kommunikation (Austausch von Daten)
        - » für die Synchronisation untereinander
      - Fragen
        - » Organisation der Datenstrukturen
        - » Ablauf der Tasks
        - » Explizite oder implizite Kommunikation
    - Ziel:
      - Reduzierung des Kommunikations- und Synchronisationsaufwandes (aus der Sicht des Prozessors)
      - Erhalten der Lokalität der Datenzugriffe, soweit möglich
      - Reduzierung des Parallelisierungsaufwandes
      - Rechnerarchitekt: bereitstellen effizienter Mechanismen, die die Festlegung vereinfachen

- Parallelisierungsprozess
  - Festlegung
    - Programmiermodelle:
      - Shared-Memory-Programmiermodell
      - Nachrichten-orientiertes Programmiermodell
      - Datenparalleles Programmiermodell

- Parallelisierungsprozess
  - Festlegung
    - Shared-Memory Programmiermodell
      - Primitive:

Name	Syntax	Funktion
CREATE	**CREATE(p,proc,args)**	Generiere Prozess, der die Ausführung bei der Prozedur **proc** mit den Argumenten **args** startet
G_MALLOC	**G_MALLOC(size)**	Allokation eines gemeinsamen Datenbereichs der Größe **size** Bytes
LOCK	**LOCK(name)**	Fordere wechselseitigen exklusiven Zugriff an
UNLOCK	**UNLOCK(name)**	Freigeben des Locks

- Parallelisierungsprozess
  - Festlegung
    - Shared-Memory Programmiermodell
      - Primitive:

Name	Syntax	Funktion
BARRIER	**BARRIER(name,number)**	Globale Synchronisation für **number** Prozesse
WAIT_FOR_END	**WAIT_FOR_END(number)**	Warten, bis **number** Prozesse terminieren
wait for flag	**while (!flag); or WAIT(flag)**	Warte auf gesetztes flag; entweder wiederholte Abfrage (spin) oder blockiere;
set flag	**flag=1; or SIGNAL(flag)**	Setze flag; weckt Prozess auf, der **flag** wiederholt abfragt

- Parallelisierungsprozess
  - Festlegung
    - Message Passing
      - Primitive:

Name	Syntax	Funktion
CREATE	**CREATE(procedure)**	Erzeuge Prozess, der bei **procedure** startet
SEND	**SEND(src_addr,size, dest,tag)**	Sende **size** Bytes von Adresse **src_addr** an **dest** Prozess mit **tag** Identifier
RECEIVE	**RECEIVE(buffer_addr ,size,src,tag)**	Empfange eine Nachricht mit der Kennung **tag** vom **src**-Prozess und lege **size** Bytes in Puffer bei **buffer_addr** ab
BARRIER	**BARRIER(name,number)**	Globale Synchronisation von **number** Prozessen