# Rechnerstrukturen

Vorlesung im Sommersemester 2008

Prof. Dr. Wolfgang Karl

Universität Karlsruhe (TH)

Fakultät für Informatik

Institut für Technische Informatik

# Kapitel 1: Grundlagen

## 1.3 Einführung: Bewertung der Leistungsfähigkeit

# Bewertung der Leistungsfähigkeit

- Ziele
  - Auswahl der Rechenanlage
  - Veränderung der Konfiguration einer bestehenden Anlage
  - Entwurf von Anlagen

# Laufzeitmessung bestehender Programme

- Benchmarks
  - Bewertung der Leistungsfähigkeit aufgrund von Messungen
    - Programm oder Programmsammlung im Quellcode
    - Übersetzung notwendig
    - Messung der Ausführungszeiten
    - In die Bewertung fließt „Güte“ des Compiler und Betriebssoftware ein
    - Zugriff auf die Maschinen notwendig

# Benchmarks

- Kernels
  - Rechenintensive Teile realer Programme
  - Vorwiegend numerische Algorithmen
  - Beispiele:
    - Lawrence Livermore Loops:
      - Zur Bewertung vektorisierender Compiler
    - LINPACK Softwarepaket:
      - Lösung eines Systems linearer Gleichungen
      - Verwendet für die TOP500 Liste (http://www.top500.org)
    - BLAS (Basic Linear Algebra Subprograms)
  - Wenig Aufwand, aber nur bedingt aussagekräftig

# Benchmarks

- Kernels
  - LINPACK Softwarepaket:
    - Verwendet für die TOP500 Liste
      - http://www.top500.org

# Benchmarks

- Kernels
  - LINPACK Softwarepaket:
    - Verwendet für die TOP500 Liste

# Benchmarks

- Standardisierte Benchmarks
  - Ziel: Vergleichbarkeit von Rechnern (inkl. Betriebssystem und Compiler)
  - Anforderungen:
    - Gute Portierbarkeit
    - Repräsentativ für typische Nutzung der Rechner
  - Sammlung von Benchmark-Programmen (Benchmark Suites)
    - Ausgeglichene Bewertung durch die unterschiedlichen Eigenschaften der Programme

# Benchmarks

- Standardisierte Benchmarks
  - Standardisierungsorganisationen
    - TPC (Transaction Processing Performance Council)
      - Mitte der 80‘er Jahre, http://www.tpc.org
      - Zusammenschluss von Datenbank- und Rechnerherstellern
      - Ziel: Bewertung von Datenbanksystemen

# Benchmarks

- Standardisierte Benchmarks
  - Standardisierungsorganisationen
    - EEMBC (Embedded Microprocessor Benchmark Consortium)
      - http://www.eembc.org
      - Anwendungen aus dem Bereich Eingebettete Systeme
      - Automotive/Industrial (Automatisierungstechnik)
        - » 6 Microbenchmarks (arithmetische Operationen, Pointer, Speicherleistung, Matrizenarithmetik, Tabellensuche, Bitmanipulation), 5 Steuerungsprogramme aus dem Automobilbereich, 5 Filter und FFT-Benchmarks
      - Consumer (Unterhaltungselektronik)
        - » 5 Multimedia-Benchmarks (JPEG compress/decompress, Filtering, RGB Coversions)
      - Networking (Netzwerk)
        - » Kürzeste-Wege-Berechnung, IP Routing, Paketfluss-Operationen
      - Office Automation
        - » Graphik- und Text-Benchmarks, (Bézier-Kurvenberechnung,´Dithering, Bildrotation, Textverarbeitung)
      - Telecommunication
        - » Filtering und DSP-Anwendungen

# Benchmarks

- Standardisierte Benchmarks
  - Standardisierungsorganisationen
    - SPEC (Standard Performance Evaluation Corporation)
      - Gegründet 1988, http://www.spec.org
      - Zusammenschluss von mehr als 40 Firmen (Rechnerhersteller)
      - Festlegung von Richtlinien für eine gemeinsame Rechnerbewertung

# SPEC Benchmarks

- Verschiedene Untergruppen in SPEC
  - Open Systems Group (OSG)
    - SPEC CPU 2000: Prozessorleistung
    - SPEC JVM98: Vergleich von Java Virtual Machine Client Plattformen
    - SPEC MAIL2001: Mail Server Benchmark
    - SPEC SFS 2.0: System: File Server
    - SPEC WEB99: Test für WWW Server

# SPEC Benchmarks

- Verschiedene Untergruppen in SPEC
  - High Performance Group (HPG)
    - SPEChpc96: Industrielle große Anwendungen
  - Graphics Performance Characterization Group (GPC)
    - Zusammenschluss mit SPECopc: OpenGL Leistungsbewertung

# SPEC Benchmarks

- SPEC CPU 2000
  - 12 nichtnumerische Programme in C/C++ (CINT2000)
  - 14 numerische Programme in FORTRAN/C (CFP2000)
  - Strenge, genau festgelegte Regeln
    - Ab CPU95: vollautomatische Messung und Protokollierung
  - Regelmäßige Aktualisierungen (CPU92, CPU95, CPU2000)
    - Laufzeiten werden zu kurz
    - Caches werden größer: größere Datensätze
    - Mehr Praxisnähe: Programme mit schlechterer Datenlokalität

# Beispiel SPEC CPU 2000

- SPEC CINT2000

| Benchmark | Referenzzeit | Bemerkung |
|---|---|---|
| 164.gzip | 1400 | Datenkompression |
| 175.vpr | 1400 | FPGA Layout und Routing |
| 176.gcc | 1100 | C Compiler |
| 181.mcf | 1800 | Lineare Optimierung, KFZ-Scheduling |
| 186.crafty | 1000 | Schachprogramm |
| 197.parser | 1800 | Sprachverarbeitung, Parser für Englisch |
| 252.eon | 1300 | Ray Tracing |
| 253.perlbmk | 1800 | Perl |
| 256.gap | 1100 | Gruppentheorie, Interpreter |
| 255.vortex | 1900 | Datenbank, objektorientiert |
| 256.bzip2 | 1500 | Datenkompression |
| 300.twolf | 3000 | IC Platzierung und Routing |

Referenzzeit bezieht sich auf Sun Ultra10 (300MHz, 256MB)

# Beispiel SPEC CPU 2000

- SPEC CFP2000

| Benchmark | Referenzzeit | Bemerkung |
|---|---|---|
| 168.wupwize | 1600 | Quantenphysik |
| 171.swim | 3100 | Gleichungslösung, Flachwasser-Modellierung |
| 172.mgrid | 1800 | 3D-Mehrgitterverfahren |
| 173.applu | 2100 | Lösung parabolischer u. elliptischer part. DG |
| 177.mesa | 1400 | 3D Graphikbibliothek |
| 178.galgel | 2900 | Strömungsmechanik |
| 179.art | 2600 | Bilderkennung, neuronales Netzwerk Simulation |

# Beispiel SPEC CPU 2000

- SPEC CFP2000

| Benchmark | Referenzzeit | Bemerkung |
|---|---|---|
| 183.equake | 1300 | Seismografische Wellenausbreitung, Finite Elemente Simulation |
| 187.facerec | 1900 | Bildverarbeitung, Gesichtserkennung |
| 188.ammp | 2200 | Chemie, Molekulardynamik-Sim. |
| 189.lucas | 2000 | Zahlentheorie, Mersenne-Primzahltest |
| 191.fma3d | 2100 | Finite Elemente: Crash-Simulation |
| 200.sixtrack | 1100 | Teilchenbeschleunigermodell |
| 300.apsi | 2600 | Sim. Schadstoffausbreitung |

# Beispiel SPEC CPU 2006

- Neueste Version:
  - SPEC CINT2006 Benchmarks.htm
  - SPEC CFP2006 Benchmark Descriptions.htm

# Beispiel SPEC CPU 2000

- Mögliche Kategorien

| | **Geschwindigkeit** | **Durchsatz** |
|---|---|---|
| **Aggressive Optimierung** | SPECint2000 | SPECint_rate2000 |
| | SPECfp2000 | SPECfp_rate2000 |
| **Konservative Optimierung** | SPECint_base2000 | SPECint_rate_base2000 |
| | SPECfp_base2000 | SPECfp_rate_base2000 |

# Beispiel SPEC CPU 2000

- SPEC CPU Benchmark-Metrik: Geschwindigkeit

$$\text{SPECratio} = \frac{\text{Referenzzeit}_x}{\text{Laufzeit}_x \text{ auf Testsystem}}$$

Benchmark x

– Endwerte: je ein geometrisches Mittel der SPECratio‘s über alle CINT2000 und CFP2000 Benchmarks
  - SPECint2000, SPECfp2000
    – Aggressive, individuelle Optimierungen erlaubt
  - SPECint_base2000, SPECfp_base2000
    – Nur mit konservativer Standardoptimierung
    – Identische Compileroptionen für alle Programme

# Beispiel SPEC CPU 2000

- SPEC CPU Benchmark-Metrik
  - Warum geometrisches Mittel:

$$\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}$$

mit *Execution time ratio*$_i$: Ausführungszeit der Programms *i* einer Last von *n* Programmen, normalisiert bezüglich der Referenzmaschine

- Eigenschaft des geometrischen Mittels:

$$\frac{\text{geometrisches Mittel } (X_i)}{\text{geometrisches Mittel } (Y_i)} = \text{geometrisches Mittel}\left(\frac{X_i}{Y_i}\right)$$

- Geometrisches Mittel ist konsistent, unabhängig von Referenzmaschine!

# Beispiel SPEC CPU 2000

- SPEC CPU Benchmark-Metrik: Durchsatz

$$\text{SPECrate}_x = \frac{\text{Sekunden pro Stunde}}{\text{Laufzeit}_x \text{ von } n_x \text{ Kopien auf Testsystem}} \times \frac{\text{Referenzzeit}_x}{\text{längste Laufzeit}}$$

  - Endwerte: je ein geometrisches Mittel der SPECrate‘s über alle CINT2000 bzw. CFP2000 Benchmarks
    - SPECint_rate2000, SPECfp_rate2000
    - SPECint_base2000, SPECfp_base2000
    - $n_x$ kann frei gewählt werden, muss aber dokumentiert werden

# Literatur

- Hennessy/Patterson: A Quantative Approach: Kap. 1.5 – 1.9

- Messung während des Betriebs von Anlagen
  - Monitore
    - Aufzeichnungselemente, die zum Zweck der Rechnerbewertung die Verkehrsverhältnisse während des normalen Betriebs beobachten und untersuchen.
    - Hardware-Monitore
      - Unabhängige physikalische Geräte
      - Keine Beeinflussung
    - Software-Monitore
      - Einbau in das Betriebssystem
      - Beeinträchtigung der normalen Betriebsverhältnisse

# Messung während des Betriebs von Anlagen

- Monitore
  - Aufzeichnungstechniken:
    - Kontinuierlich oder sporadisch
    - Gesamtdatenaufzeichnung (Tracing)
    - Realzeitauswertung
    - Unabhängiger Auswertungslauf (Post Processing)

# Bewertung der Leistungsfähigkeit

- Modelltheoretische Verfahren
  - Unabhängig von der Existenz eines Rechners
  - Modellbildung
    - aufgrund von Annahmen über die Struktur und Betrieb eines Rechners und über die Prozesse
    - Darstellung der für die Analyse relevanten Merkmale des Systems:
      - Systemkomponenten
      - Datenverkehr zwischen den Systemkomponenten
    - Abstrahierung komplexer Systeme
      - Nur die interessierenden Größen werden erfasst
    - Ziel:
      - Aufdecken von Beziehungen zwischen Systemparametern
      - Ermitteln von Leistungsgrößen (Auslastung von Prozessoren und Kanälen, mittlere Antwortzeiten, Warteschlangenlängen, ...)

# Modelltheoretische Verfahren

- Modellbildung
  - Analytische Methoden
    - versuchen auf mathematischem Weg, Beziehungen zwischen relevanten Leistungskenngrößen und fundamentalen Systemparametern herzuleiten
    - oft nur minimaler Aufwand, aber dafür weniger aussagekräftig
  - Warteschlangenmodelle
    - Leistungsanalyse von Rechensystemen
  - Petrinetze
    - theoretische Untersuchungen
  - Diagnosegraphen
    - Zuverlässigkeitsanalysen
  - Netzwerkflussmodelle
    - Kapazitätsüberlegungen

# Modelltheoretische Verfahren

- Modellbildung
  - Analytische Methoden
    - Beispiel Warteschlangenmodelle
      - Deterministische Warteschlangenmodelle
        - » Beispiele für Systemparameter: Rechenzeit, Gerätebedienzeit, Ankunftszeit eines Jobs
        - » Feste Werte → deterministische Ergebnisse für die Leistungsgrößen
      - Stochastische Warteschlangenmodelle
        - » Systemparameter statistisch verteilt, mit vorgegebenen Mittelwerten, Verteilungsfunktionen → statistisch verteilte Leistungsgrößen

# Modelltheoretische Verfahren

- Modellbildung
  - Analytische Methoden
    - Beispiel Warteschlangenmodelle
      - Operationelle Warteschlangenmodelle
        - » Systemparameter: Gemessene Werte aus der Beobachtung eines Systems in einem festen Zeitintervall
        - » Vereinfachung der Gleichungen für die Bestimmung von Leistungsgrößen
        - » Relativ gute Aussagen über das Leistungsverhalten des Systems bei geeignet gewähltem Zeitintervall

- Modellbildung
  - Simulation
    - Vorgänge in einem Rechensystem werden nachgebildet
    - Verwendung üblicher Programmiersprachen oder spezieller Simulationssprachen
    - Verhalten des Simulationsmodells in Bezug auf die relevanten Parameter entspricht weitgehend dem Verhalten des realen Systems
    - Ermittlung der für die Leistungsbewertung interessierenden Größen

# Modelltheoretische Verfahren

- Modellbildung
  - Simulation
    - Deterministische Simulation
      - Alle an dem Modell beteiligten Größen sind exakt definiert oder berechenbar
    - Stochastische Simulation
      - Verwendung von zufallsabhängigen Größen
      - Oft: Einsatz von Zufallsgeneratoren
    - Aufzeichnungsgesteuerte Simulation
      - Verwendung von gemessenen Werten
      - Erfasst an aufgrund der Beobachtung eines Systems in einem festen Zeitintervall

# Modelltheoretische Verfahren

- Modellbildung
  - Simulation
    - Nachteile:
      - Vorbereitung und Ausführung der Simulationsmodelle zeitaufwendig und teuer
      - Planung der Experimente muss sorgfältig durchgeführt werden
      - Auswertung und Interpretation der Ergebnisse nicht immer einfach
    - Beispiel:
      - SimpleScalar Tool Set (http://www.simplescalar.com)
        - » „Standard“-Werkzeug zur Simulation von superskalarer Mikroprozessoren

# Modelltheoretische Verfahren

- Modellbildung
  - Simulation vs. Analytische Methoden
    - Simulation:
      - Realistischere Annahmen über das System bei der Simulation
      - Berücksichtigung vieler verschiedener Systemgrößen
      - Abdeckungen verschiedener Anwendungsbereiche
    - Können sich gegenseitig gut ergänzen

# Bewertung der Leistungsfähigkeit

- Zusammenfassung
  - Rechnerleistung abhängig von
    - Blickwinkel (Nutzer, Administrator)
    - Nutzung des Rechners (Programm)
    - System- und Betriebssoftware
    - Compilertechnologie
    - Befehlssatz
    - Mikroarchitektur
    - Halbleitertechnologie

# Bewertung der Leistungsfähigkeit

- Zusammenfassung
  - Einfache Maßzahlen sind wenig aussagekräftig
    - Beschreiben nur einzelne Aspekte der Leitung
    - Nie isoliert betrachten
  - Zur Rechnerauswahl / Bewertung von Systemen vor bzw. beim Kauf:
    - Benchmarks
      - Problem: Aufwand vs. Repräsentativität
      - De-facto-Standard für CPU-Leistung: CPU2000 Benchmark-Suite

# Bewertung der Leistungsfähigkeit

- Zusammenfassung
  - Bewertung von Systemen im Betrieb
    - Monitore: Messung und Auswertung des Verhaltens
  - Bewertung von Systemen, die noch nicht existieren
    - Analytische Methoden und Simulation
    - Problem: Aufwand vs. Genauigkeit

# Bewertung der Leistungsfähigkeit

- Übersicht über Rechnerbewertungsverfahren

| | **Auswertung von Hardware-maßen und Parametern** | **Laufzeit-messungen bestehender Programme** | **Messungen während des Betriebs der Anlagen** | **Modell-theoretische Verfahren** |
|---|---|---|---|---|
| **Rechnerauswahl** | Maßzahlen für die Operations-geschwindingkeit<br>Kernprogramme | Benchmarks | | |
| **Rechner-„Tuning“** | | | Hardware-Monitore<br>Software-Monitore | |
| **Rechnerentwurf** | | | | Analytische Methoden<br>Simulation |

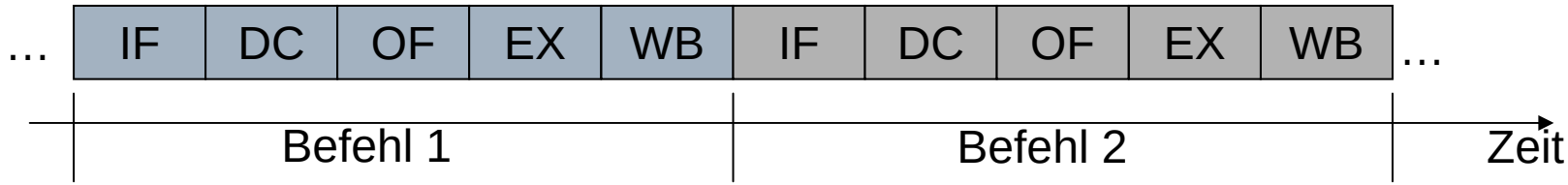
# Kapitel 1: Grundlagen

## 1.4 Parallelverarbeitung

- Sequentielle Sichtweise
  - Berechnungen werden schrittweise oder seriell ausgeführt
  - Algorithmen sind als Folge von Berechnungsschritten organisiert
  - Sequentielle Programme
  - Schrittweise Ausführung einer Folge von Befehlen auf einer sequentiellen Maschine
- Beschleunigung der Ausführung
  - Erhöhung der Taktfrequenz
  - Parallele Ausführung der Aufgaben

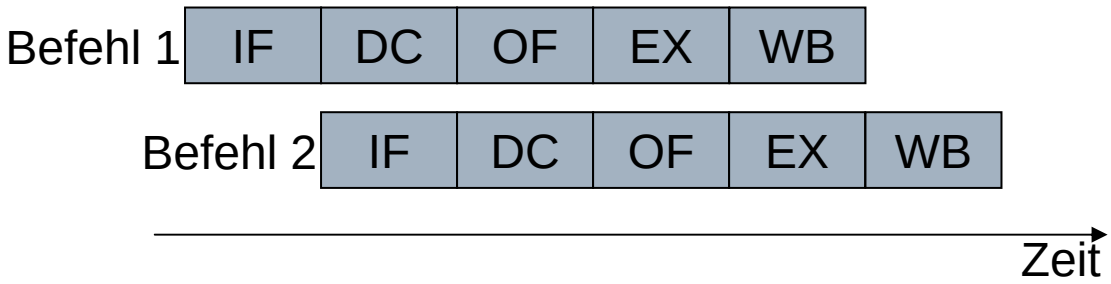
- Parallelverarbeitung
  - Algorithmenentwurf
    - Muss viele unabhängige Operationen umfassen
  - Struktur der Programmiersprache
    - Muss die Spezifikation oder die automatische Identifizierung paralleler Operationen ermöglichen
  - Rechnerarchitektur
    - Gleichzeitige Ausführung paralleler Operationen

- Sequentieller Rechner
  - Sequentielle Ausführung einer Folge von Maschinenbefehlen
    - Maschinenbefehlszyklus

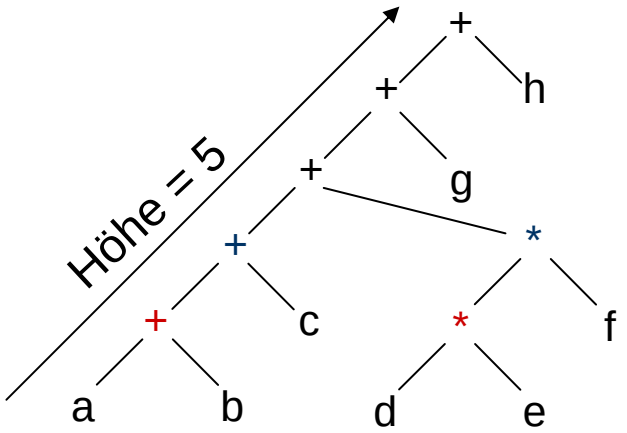



IF: Befehl holen
DK: Befehl dekodieren
OF: Operanden bereitstellen
EX: Befehl ausführen
WB: Ergebnis speichern

- Pipelining des Maschinenbefehlszyklus
  - Überlappte Ausführung der Phasen des Maschinenbefehlszyklus



- Gleichzeitige Ausführung von Operationen
  - Beispiel: Auswertung des arithmetischen Ausdrucks *EXP= a+b+c+(d*e*f)+g+h* durch den Compiler von links nach rechts




**Seq. Ausführung**
S1: t1 = a + c
S2: t1 = t1 + c
S3: t2 = d * e
S4: t2 = t2 * f
S5: t1 = t1 + t2
S5: t1 = t1 + g
S6: t1 = t1 + h

**Parallele Ausf.**
S1: t1 = a + c; S3: t2 = d * e
S2: t1 = t1 + c; S4: t2 = t2 * f
S5: t1 = t1 + t2
S5: t1 = t1 + g
S6: t1 = t1 + h

  - Höhe:
    - längster Pfad von der Wurzel zum Blatt
    - Minimale Anzahl von Schritten zur Auswertung des Ausdrucks

- Formen des Parallelismus
  - Nebenläufigkeit
    - Eine Maschine arbeitet nebenläufig, wenn die Objekte vollständig gleichzeitig abgearbeitet werden.
  - Pipelining
    - Pipelining auf einer Maschine liegt dann vor, wenn die Bearbeitung eines Objektes in Teilschritte zerlegt und diese in einer sequentiellen Folge (Phasen der Pipeline) ausgeführt werden. Die Phasen der Pipeline können für verschiedene Objekte überlappt abgearbeitet werden. (Bode 95)

- Ebenen der Parallelität
  - Programmebene
    - Parallele Verarbeitung verschiedener Programme
    - Vollständig unabhängige Einheiten
      - ohne gemeinsame Daten
      - wenig oder keine Kommunikation und Synchronisation
    - Parallelverarbeitung wird vom Betriebssystem organisiert

- Ebenen der Parallelität
  - Prozessebene (Task-Ebene)
    - Programm wird in Anzahl parallel ausführbarer Prozesse zerlegt
      - Prozess: schwergewichtiger Prozess (heavy-weighted process), Beispiel: UNIX-Prozesse
        - » Besteht aus vielen sequentiell ausgeführten Befehlen und umfasst eigene Datenbereiche
      - Synchronisation und Kommunikationsaufwand
    - Betriebssysteme unterstützt Parallelverarbeitung durch Primitive zur Prozessverwaltung, Prozess-Synchronisation, Prozesskommunikation

- Ebenen der Parallelität
  - Blockebene
    - leichtgewichtige Prozesse (Threads)
      - Bestehen jeweils aus sequentiell ausgeführten Befehlen teilen sich gemeinsamen Adressraum
      - Beispiel: Threads gemäß POSIX 1003.a Standard in mehrfädigen (multithreaded) Betriebssystemen
      - Synchronisation über Schlossvariablen (mutex), und Bedingungsvariablen (condition variables) oder darauf aufbauenden Synchronisationsmechanismen
      - Kommunikation über gemeinsame Daten
      - Aufwand für Thread-Erzeugung und-Beendigung, Thread-Wechsel geringer

- Ebenen der Parallelität
  - Blockebene
    - Anweisungsblöcke
      - Innere und äußere parallele Schleifen in FORTRAN-Dialekten
      - Verwendung von Microtasking
      - Hohes Parallelitätspotential durch parallel ausführbare Schleifeninterationen

- Ebenen der Parallelität
  - Anweisungs- oder Befehlsebene
    - Parallele Ausführung einzelner Maschinenbefehle oder elementarer Anweisungen
    - Optimierende (parallelisierende) Compiler für VLIW-Prozessoren oder Anwendung der Superskalartechnik in superskalaren Mikroprozessoren
      - Analyse der sequentiellen Befehlsfolge
      - Umordnen und Parallelisieren der Befehle
    - Datenflusssprachen und funktionale Programmiersprachen erlauben explizite Spezifikation der Parallelität

- Ebenen der Parallelität
  - Suboperationsebene
    - Elementare Anweisung wird durch Compiler oder durch die Maschine in Suboperationen aufgebrochen, die parallel ausgeführt werden
      - Vektoroperationen
        - » Überlappte Ausführung auf Vektorrechner in Vektorpipeline
        - » Komplexe Datenstrukturen (Matrizen, Vektoren, Datenströme) sind in höherer Programmiersprache verfügbar oder werden von einem parallelisierenden, vektorisierenden Compiler aus einer sequentiellen Programmiersprache generiert
        - » Beispiel: Vektoraddition C = A + B statt Abarbeitung einer Schleife

- Körnigkeit der Parallelität
  - Die Körnigkeit oder Granularität (grain size) ergibt sich aus dem Verhältnis von Rechenaufwand zu Kommunikations- oder Synchronisationsaufwand. Sie bemisst sich nach der Anzahl der Befehle in einer sequentiellen Befehlsfolge.
  - Programm-, Prozess- und Blockebene werden häufig auch als grobkörnige (large grained) Parallelität,
  - die Anweisungsebene als feinkörnige (finely grained) Parallelität bezeichnet.
  - Seltener wird auch von mittelkörniger (medium grained) Parallelität gesprochen, dann ist meist die Blockebene gemeint.

- Techniken der Parallelarbeit vs. Parallelitätsebenen

| Parallelarbeitstechniken | Programmebene | Prozessebene | Blockebene | Anweisungsebene | Suboperationsebene |
|---|---|---|---|---|---|
| **Techniken der Parallelarbeit durch Rechnerkopplung** | | | | | |
| Grid-Computing | X | X | | | |
| Cluster | X | X | | | |
| **Techniken der Parallelarbeit durch Prozessorkopplung** | | | | | |
| Nachrichtenkopplung | X | X | | | |
| Speicherkopplung (SMP) | X | X | X | | |
| Speicherkopplung (DSM) | X | X | X | | |
| Grobkörniges Datenflußprinzip | | X | X | | |
| **Techniken der Parallelarbeit in der Prozessorarchitektur** | | | | | |
| Befehlspipelining | | | | X | |
| Superskalar | | | | X | |
| VLIW | | | | X | |
| Überlappung von E/A- mit CPU-Operationen | | | | X | |
| Feinkörniges Datenflußprinzip | | | | X | |
| **SIMD-Techniken** | | | | | |
| Vektorrechnerprinzip | | | | | X |
| Feldrechnerprinzip | | | | | X |
| SIMD-Operationen | | | | | |

Quelle: Ungerer, T. : Skript Rechnerstrukturen, SS 2000

# Einführung: Literatur

- Theo Ungerer: Parallelrechner und parallele Programmierung, Kap.1.1 – 1.3

# Kapitel 1: Grundlagen

## 1.5 Einführung: Klassifizierung von Rechnerarchitekturen

- Klassifikationen
  - Aufspannen von Entwurfsräumen
  - Aufzeigen von Entwurfsalternativen
  - Klassifikationsschemata versuchen, der Vielfalt von Rechnerarchitekturen eine Ordnungsstruktur zu geben
  - Frühe Klassifikationen konzentrieren sich auf die Hardware-Struktur
    - Anordnung und Organisation der Verarbeitungselemente
    - Operationsprinzip

- Klassifizierung nach M. Flynn
  - Zweidimensionale Klassifizierung
    - Hauptkriterien:
      - Zahl der Befehlsströme und
      - Zahl der Datenströme sind.
    - Merkmale:
      - Ein Rechner bearbeitet zu einem gegebenen Zeitpunkt einen oder mehr als einen Befehl.
      - Ein Rechner bearbeitet zu einem gegebenen Zeitpunkt einen oder mehr als einen Datenwert.

- Klassifizierung nach M. Flynn
  - Vier Klassen von Rechnerarchitekturen
    - SISD Single Instruction – Single Data
      - Uniprozessor
    - SIMD Single Instruction – Multiple Data
      - Vektorrechner, Feldrechner
    - MISD Multiple Instructions – Single Data
      - ?
    - MIMD Multiple Instructions – Multiple Data
      - Multiprozessor

- Diskussion der Flynnschen Klassifizierung
  - Schwachpunkte:
    - Sehr hohes Abstraktionsniveau ==> sehr unterschiedliche Rechnerarchitekturen fallen in die gleiche Klasse.
    - Viele moderne Parallelarbeitstechniken (z.B. Superskalar, Befehlspipelining, VLIW) lassen sich überhaupt nicht einordnen.
    - In heutigen Rechnern findet man die Kombination mehrerer Techniken.
      Beispiel: Vektorrechner-Multiprozessoren fallen in eine als MIMD/SIMD zu bezeichnende Klasse.
    - Die Klasse MISD wurde nur der Systematik wegen aufgeführt.

- Diskussion von Klassifikationen
  - Kennzeichnend für moderne Rechnerstrukturen:
    - Prinzip der Virtualität:
      - Mit verschiedenen Techniken und Mechanismen in der Hardware können auf einer Maschine verschiedene parallele Programmiermodelle unterstützt werden
      - Die zugrunde liegende Organisation und Architektur ist weitgehend transparent
    - Allgemeiner Trend in der Rechnerarchitektur
      - Verwendung von Standardkomponenten
      - Verständnis über die Implementierungstechniken zur Virtualität
      - Keine allgemeingültige Klassifikation!