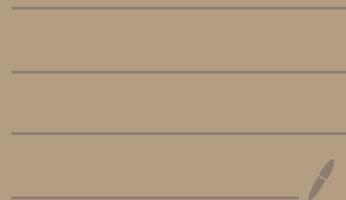


# Rechnerstrukturen

---

SS 20



## CISC vs RISC

### CISC ISA

- Befehlssatz mit komplexen Maschinenbefehlen
- Mikroprogrammierte Implementierung des Steuerwerks
- Variabiles Befehlsformat → Befehslänge

### RISC ISA

- einfache Maschinenbefehle
  - ↪ einheitliches, festes Befehlsformat
- Load / Store Architektur
  - ↪ Befehle arbeiten auf Registeroperanden
  - ↪ Load-, Speicherbefehle greifen auf Speicher zu
- Einzyklus - Maschinenbefehle
  - ↪ ermöglicht effizientes Pipelining
- optimierende Compiler

### Pipeline

1 Takt = 1 Stufe



Inst <sub>i</sub>	IF	ID	EX	MA	WB			
Inst <sub>i+1</sub>		IF	ID	EX	MA	WB		
Inst <sub>i+2</sub>			IF	ID	EX	MA	WB	
Inst <sub>i+3</sub>				IF	ID	EX	MA	WB
Inst <sub>i+4</sub>					IF	ID	EX	MA WB

IF = Instruction fetch = Befehl aus Register holen

ID = Instruction decode

EX = Execution

MA = Memory Access

WB = Write Back

### Moore's Law

Anzahl Transistoren die auf einem IC integriert werden können, verdoppelt sich alle 2 Jahre

## RECHNERARCHITEKTUR

- Entwurf, Beschreibung, Vergleich, Beurteilung, Verbesserung bestehender, zukünftiger Rechenanlagen
- Betrachtet Aufbau + Eigenschaften des Ganzen (Rechenanlage), seiner Teile (Komponenten) und seiner Verbindungen (Globalstruktur, Infrastruktur)

### Problem Kompromiss finden zwischen

- Zielsetzungen
  - Einsatzgebiet, Anwendungsbereich, ...
- Randbedingungen
  - Technologie, Größe, Geld, Energieverbrauch, Umwelt, ...
- Gestaltungsgrundsätzen
  - Modularität, Sparsamkeit, Fehlertoleranz, ...
- Anforderungen
  - Kompatibilität, OS-Anforderungen, Standards

### Zielsetzungen

#### Einsatzgebiete

##### Personal Mobile Devices

- drahtlose Geräte mit Multimedia Schnittstelle  
z.B. Smartphone, Tablet
- niedrige Kosten, niedriger Energieverbrauch
- hohe Leistung für Multimedia-Anwendungen

##### Desktop Computing

- PCs bis Workstations
- günstiges Preis-/Leistungsverhältnis
- ausgewogene Rechenleistung für breites Anwendungsspektrum

## Server

- rechen-, datenintensive Anwendungen, transaktionsorientierte Anwendungen
- hohe Anforderungen an Verfügbarkeit, Zuverlässigkeit, Energieeffizienz, Skalierbarkeit
- große Datei-Systeme, Ein-/Ausgabesysteme
- abgeschlossene Räume, kostenintensiv

## Server : Höchstleistungsrechner

- hohe Rechenleistung bzgl. Gleitkommaverarbeitung
- große, kommunikationsintensive Anwendungen
- Batch- Programme

## Kommerzielle Server, Warehouse-Scale Computers

- Mainframes, Clusters
- Durchsatzorientierte Anwendungen
- SaaS

## Embedded Systems

- Mikroprozessorsysteme eingebettet in Geräten
- hoch spezialisiert  
↳ für diese Aufgaben hohe Leistungsfähigkeit
- breites Preis-/Leistungsspektrum
- Echtzeitanforderungen
- Abwägen der Anforderungen an Rechenleistung, Speicherbedarf, Kosten, Energieverbrauch

## Anwendungsbereiche

### Technisch-wissenschaftlich

- hohe Anforderungen an Rechenleistung, insb. Gleitkommaverarbeitung
- z.B. Simulationen, Modellierungen, Medizintechnik, ...

### Kommerziell

- DB, web, Social Networks, ...

### Eingebettete Systeme

## Rechenleistung

- Latenz, Antwortzeit
- Bandbreite, Durchsatz
- Bewertung der Leistungsfähigkeit durch Benchmarks

## Zuverlässigkeit

- bei Ausfall von Komponenten muss betriebsfähiger Kern bereit sein
- Fehlertoleranzmechanismen:
  - z.B. Redundante Komponenten
- Bewertung durch stochastische Verfahren

## Verfügbarkeit

## Energieverbrauch, Leistungsaufnahme

- Laufzeit evtl. durch Akkus begrenzt
  - ↳ maximiere Laufzeit
  - minimiere Wärmeerzeugung

## Randbedingungen

### Technologische Entwicklung

- Mikrominiaturisierung setzt sich fort (ITRS Roadmap)
  - Verkleinerung der Strukturbreiten
  - Verdopplung # Transistoren alle 18 Monate
  - Erhöhung Integrationsdichte
- Entwicklung DRAM - Technologie
- Steigerung DRAM - Kapazität
- -- Flash - Kapazität
- Magnetspeicher - Technologie

# Elektrische Leistung und Energie

Elektrische Leistung = Energiefluss pro Zeit  
 $P = \frac{E}{t}$

↳ Ziele:

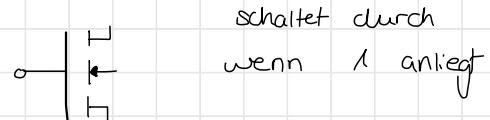
- Verringerung Energieverbrauch  $\rightarrow$  Erhöhung Betriebszeit
- Reduktion Temperatur  $\rightarrow$  weniger Verlustleistung

## CMOS-Schaltungen

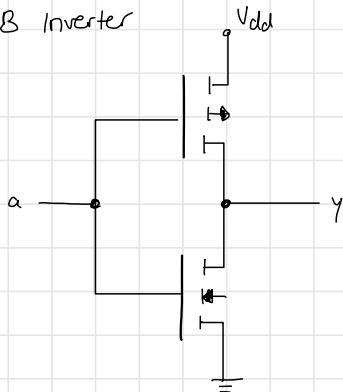
pMOS:



nMOS:



ZB Inverter

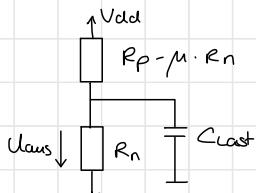


a	y
0	1
1	0

- $a = 0 \rightarrow$  nMOS sperrt  
pMOS schaltet Vdd durch
- $a = 1 \rightarrow$  pMOS sperrt  
nMOS schaltet GND durch
- Stromverbrauch nur bei Übergängen  
 $0 \rightarrow 1 / 1 \rightarrow 0$

↳ Ersatzschaltbild: Widerstände + Kapazitäten

- $R_p$  = Wrd. des p-Netzes  
Leitet p-Netz ist  $R_p$  klein, ansonsten groß
- $R_n$  = Wrd. des n-Netzes  
Leitet n-Netz ist  $R_n$  klein, ansonsten groß
- $C_{load}$  = Lastkap. der am Ausgang angeschlossenen Leitungen und weiteren Schaltungen



$$\begin{aligned}
 \text{Schaltwahrscheinlichkeit } P_{\text{Schalt}} &= \\
 P(0 \rightarrow 1 \vee 1 \rightarrow 0) &= P(0 \rightarrow 1) + P(1 \rightarrow 0) \\
 &= P(0) \cdot P(1) + P(1) \cdot P(0) = 2P(1)P(0) \\
 &\text{wkt dass } P_{\text{out}} = 1
 \end{aligned}$$

## Leistungsaufnahme

$$P_{\text{total}} = P_{\text{switching}} + P_{\text{shortcircuit}} + P_{\text{static}} + P_{\text{leakage}}$$

- $P_{\text{switching}} = C_{\text{eff}} \cdot V_{\text{dd}}^2 \cdot f$  = Laden / Schalten einer kapazitiven Last
  - $C_{\text{eff}} = C \cdot \alpha$  → bei Zustandsübergang  $\alpha = \text{relative Häufigkeit eines Wechsels}$
  - wesentlicher Anteil am Leistungsverbrauch
- $P_{\text{shortcircuit}} = I_{\text{mean}} \cdot V_{\text{dd}}$  = Leistungsverbrauch bei Eingabewechsel
  - $I_{\text{mean}}$  = mittlerer Strom während Wechsel des Eingabesignals
- $P_{\text{leakage}} = \text{kriechströme} \sim \# \text{Transistoren}$ 
  - da Widerst. zw. Leiterbahnen nicht unendlich hoch
  - größer mit zunehmender Integrationsdichte
- Ideal:  $P \sim \text{Taktfrequenz } f$   $P \sim V_{\text{dd}}^2$   $P \sim f \cdot U^2$   
aber:  $V_{\text{dd}}$  und  $f$  voneinander abh.:  $V_{\text{dd}} \sim f$   
 $\Rightarrow$  Kubus-Regel:  $P \sim V_{\text{dd}}^3$  oder  $P \sim f^3$

## Energieverbrauch

- Ideal: const. Zeit  $t_k$ : Energieverbrauch  $E \sim f$   
Zeit für Aufgabe  $t_a \sim \frac{1}{f}$   
 $\hookrightarrow E$  wächst mit sinkender Taktfrequenz

## Kosten eines Rechners / Herstellungskosten

- Lernkurve: Herstellungskosten sinken im Laufe der Zeit

## Preis

- DRAMs: ~ zu Herstellungskosten
- Mikroprozessoren: abh. von Herstellungsmenge, Volumen, Wettbewerb

## Kosten integrierter Schaltkreise

$$\text{kosten des Dies} = \frac{\text{kosten des Wafers}}{\text{Dies pro Wafer} \cdot \text{Ausbeute}}$$

$$\text{Cost}_{\text{die}} = \frac{\text{Cost}_{\text{wafer}}}{\text{DPW} \cdot Y_{\text{die}}}$$



$$\text{Anzahl Dies} = \frac{\pi \cdot \left(\frac{1}{2} \text{ Durchmesser Wafer}\right)^2}{\text{Fläche des Dies}}$$

Anteil Wafer pro Chip

$$\frac{\pi \cdot \text{Durchmesser Wafer}}{\sqrt{2 \cdot \text{Fläche des Dies}}} = \frac{\text{dWafer}}{\text{Fläche}}$$

= Anteil unbrauchbar

$$\text{Ausbeute} = \text{Die Yield} = \frac{\text{Wafer Ausbeute}}{\text{Y}_{\text{Wafer}}} = Y_{\text{Die}}$$

$$\frac{1}{1 + \text{Defekt pro Flächeneinheit} \cdot \text{Fläche}}$$

$N = \text{Technologiefaktor}$   
↑  
abh. vom Hersteller

## Hardware-Entwurf

Abstraktionsebenen (Y-Diagramm)

System

Verhalten:

was passiert Leistungsanforderung

Datenfluss, Steuerfluss

Bool'sche Gleichung

Differentialgleichung

Algorithmen

Register-Transfer

Logik

elektr.

elektr.

symbolisches Layout

platzierte Zellen

Floorplan

Struktur: wie passiert es?

Netzwerk

Blockschaltbild

RT-Diagramm

Logik-Netzliste

Logik-Schaltbild

Geometrie:  
wo passiert es?

Top-down

= Chip-Entwurf

• Schrittweise Verfeinerung ausgehend von höherer Abstraktionsebene

Komplexes Verhalten,  
wenig Struktur



viel Struktur,  
einfaches Verhalten

## Bottom-up

- Rechnerentwurf
- Ausgehend von Platinen + chips : Zusammengesetzte Arbeiten festlegen

## Automatische Synthese

- (+) · Eingabespezifikation auf höherer Ebene
    - kürzere Entwurfszeit
    - komplexere Entwürfe möglich
    - weniger Entwurfsfehler
  - Ausschöpfung des Entwurfsraums
    - mehrere Entwürfe können durchgespielt werden
    - Nutzung des Optimierungspotentials
  - Flexibilität
    - Änderung der Spezifikation
    - Änderung der Zieltechnologie
  - weniger fehleranfällig
- 
- (-) · Auswirkung von Randbedingungen
  - Qualität des Syntheseergebnisses
  - Integration versch. Werkzeuge schwierig

## LEISTUNGSEWERTUNG

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

### Naive Maße

#### Ausführungszeit

- Zeit zw. Beginn, Ende eines Ereignisses
- A n-mal schneller als B :

$$\frac{\text{Ausf.zeit}(B)}{\text{Ausf.zeit}(A)} = n$$

#### Durchsatz

Anzahl ausgeführter Aufgaben in geg. Zeitintervall

#### wall-clock-time, response time

- Latenzzeit für Ausführung einer Aufgabe
- inkl. Speicher-, Plattenzugriff, I/O

#### CPU Time

- Zeit in der CPU aktiv arbeitet
- User CPU time: CPU führt Programm aus
- System CPU time: CPU führt OS-Aufgaben aus (die von Programm angefordert)

## Bewertungsverfahren

- Auswertung von HW-Maßen, Parametern
- Laufzeitmessungen bestehender Programme
- Messungen während Betrieb von Anlagen
- Modelltheoretische Verfahren

## Hardwaremaße

### CPU Leistung

- clock cycles per instruction = mittlere Anzahl Taktzyklen pro Befehl

$$CPI = \frac{T_{exe}}{IC \cdot T_c}$$

$$f = \frac{1}{T_c}$$

$$T_{exe} = \text{CPU-Zeit}$$

(IC = Instruction count)

$$T_c = \text{Zykluszeit}$$

$$T_{exe} = \# \text{ benötigter Zyklen} \cdot T_c$$

- Abschätzung schwierig durch Prozessorkomplexität, Cache-Hierarchie

- Instructions per cycle  $IPC = \frac{1}{CPI}$

### Operationsgeschwindigkeit

- MIPS = Millions of Instructions per Second

$$\text{MIPS} = \frac{\text{Anzahl ausführter Instruktionen}}{10^6 \times \text{Ausführungszeit}} = \frac{f}{CPI \cdot 10^6}$$

- MFLOPS = Millions of floating point operations per second

$$\text{MFLOPS} = \frac{\text{Anzahl ausführter float ops}}{10^6 \times \text{Ausführungszeit}}$$

- (1) - abh. von ISA und ausführter Befehlssequenz
- untersch. MIPS/MFLOPS zahlen bei versch. Programmen
- Angaben oft best-case

## Benchmarks

- Erfassung durch Messungen mit Programm / Programm sammlung

↪ Programme liegen im Quellcode vor → übersetzen

↪ Ausführungszeit messen → Einfluss von Compiler, OS

## Kernels

- Rechenintensive Teile realer Programme

z.B. Lawrence Livermore Loops  
BLAS

LINPACK

## Standardisierte Benchmarks

- Vergleich von Rechnern inkl. OS, Compiler
- Benchmark Suites: Sammlung von Programmen
  - ↳ gute Portierbarkeit, representativ für typische Nutzung

## Standardisierungsorganisationen

- TPC: Bewertung von DBSystemen
- **SPEC**: gemeinsame Rechnerbewertungs Richtlinien für Cloud, CPU, HPC, Storage, Power, ...

### CPU Benchmark

- strenge, festgelegte Regeln
- vollautomatische Messung, Protokollierung

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

→ Endwerte: geom. Mittel der SPEC ratios

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

$$\frac{\text{geom Mittel } (x_i)}{\text{geom Mittel } (y_i)} = \text{geom Mittel } \left( \frac{x_i}{y_i} \right)$$

### Durchsatz

$$\text{SPEC rate}_x = \frac{n_x \cdot \text{Referenzfaktor von } x}{\text{Ausführungszeit (in sec)}}$$

→ Endwerte: geom. Mittel der SPEC rate

## Messung während Betrieb von Anlagen

### Monitore

- untersuchen Verkehrsverhältnisse während normalem Betrieb

#### Hw - Monitore:

- unabh. physikalische Geräte
- keine Beeinflussung

#### SW - Monitore:

- Einbau in OS
- Beeinträchtigt normalen Betrieb

### Aufzeichnungstechniken:

- kontinuierlich oder sporadisch
- Gesamtdata aufzeichnung (Tracing)
  - Realzeitauswertung
- unabh. Auswertungslauf (Post Processing)

### Beispiel: Performance Counter

↳ zählt Anzahl ausgef. Befehle

## Modelltheoretische Verfahren

- unabh. von Existenz eines Rechners

## Modellbildung

- Annahme über Struktur, Betrieb eines Rechners
  - über Prozesse
- Darstellung relevanter Systemmerkmale
  - Komponenten
  - Datenverkehr zw. Komponenten
- Abstrahierung komplexer Systeme

## Analytische Methoden

- mathem. Analyse der Beziehungen zw. Leistungskenngrößen, Systemparam.
- min. Aufwand, aber wenig aussagekräftig

z.B. Warteschlangenmodelle

Petri netze

Diagnosegraphen

Netzwerkflussmodelle

## Warteschlangenmodelle

- Gesetz von Little:  $K = \lambda \cdot t$  bzw  $Q = \lambda \cdot w$
- $K$  = mittlere Anzahl Aufträge
- $\lambda$  = Durchsatz
- $t$  = Antwortzeit
- $Q$  = mittlere Warteschlangenlänge
- $w$  = Wartezeit

Voraussetzung: statistisches Gleichgewicht

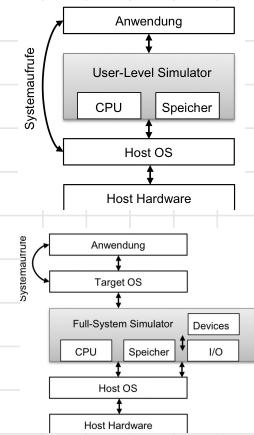
→ Rate mit der Aufträge eingehen = Rate mit der Aufträge abgehen

## Simulation

- Modellierung der wesentlichen Eigenschaften / des Verhaltens der Zielmaschine

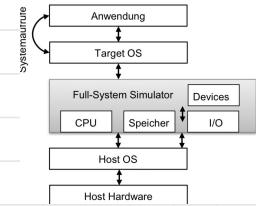
### User-Level-Simulatoren

- Simulation der Mikroarchitektur eines Prozessors
- Keine Berücksichtigung von Systemressourcen



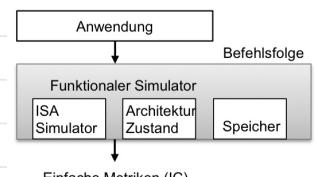
### Full-System-Simulatoren

- modellieren vollen Comp. System inkl. CPU, I/O, Disk, Netzwerk
- Booten, Ausführung von Betriebssystemen
- Beobachten der Interaktion von workload, System



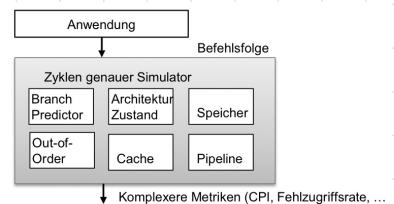
### Funktions-Simulatoren

- Modellieren nur Funktionalität ohne Berücksichtigung der Processorarchitektur
- Emulation der Befehlsarchitektur



### Zyklen-genaue Simulatoren

- inkl. Details der Mikroarchitekturblöcke
- Emulation der Funktionalität + Zeitverhalten



## Trace - driven Simulator

- Ausführung der Benchmarks auf ISA-kompatibler Prozessor oder Simulator
- nicht notwendigerweise selber Prozessor wie Zielprozessor
- ausgeführte Befehle während Benchmark auf Trace geschrieben
- Trace dann als Eingabe fürzyklengenauen Simulator

## Execution - driven Simulator

- ausführbare Datei des Benchmarks = Eingabe für Simulator

## Zuverlässigkeit und Fehlertoleranz

Zuverlässigkeit = Fähigkeit eines Systems während vorgegebener Zeitspanne bei zulässigen Betriebsbedingungen die spezifizierte Funktion zu erbringen  
↳ Quantifiziert durch Zuverlässigkeitskenngrößen

Fehlertoleranz = Fähigkeit eines Systems auch mit begrenzter Anzahl fehlerhafter Subsysteme die spezifizierte Funktion zu erbringen

Sicherheit (Safety) = Nichtvorhandensein einer Gefahr für Menschen/Sachwerte

Gefahr = Zustand in dem Schaden zwangsläufig (zufällig entstehen kann ohne dass ausreichende Gegenmaßnahmen gewährleistet

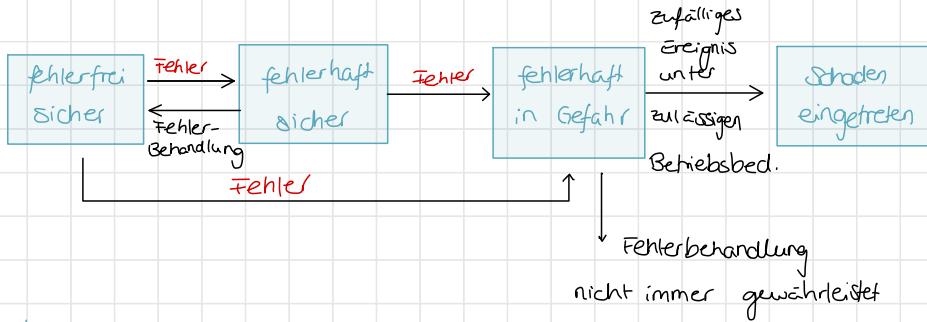
Vertraulichkeit (Security) = betrifft Datenschutz, Zugangsicherheit

Wartungsfreundlichkeit = Bei Erneuerung, Erweiterung, Wechsel des Systems sollen Auswirkungen der Änderungen so wenig wie möglich wahrnehmbar sein

Ausfall durch HW-Komponenten, SW (Programmfehler), menschliche Eingriffe

Nutzungsduer Problem: Nachweisbarkeit, Testmöglichkeit

# Fehler



## Funktionsausfälle

- unzulässige, aussetzende Funktion einer Komponente

Fehlerzustände = unzulässiger Zustand einzelner Komponenten oder Störung  
verantwortlich für Ausfall

## Wirkungskette

Fehler → Fehlerzustand → Ausfall

## FEHLERTOLERANZ

- Tolerieren der Fehlerzustände von Teilsystemen (Komponenten)
- Erhöhung der Zuverlässigkeit

## Fehlerursachen

### Entwurf fehler

- System von vornherein fehlerhaft konzipiert
- Spezifikations-, Implementierungs-, Dokumentationsfehler

### Herstellung fehler

- aus korrektem Entwurf wird fehlerhaftes Produkt

### Betriebsfehler

- erzeugen zur Nutzung zeit fehlerhaften Zustand
- Störungsbedingt: mechanische, elektr., magnetische, elektr magn., thermische Störung aus äußeren Einflüssen

- Verschleißfehler: treten mit zunehmender Betriebsdauer in der HW auf
- Zufällige physikalische Fehler
- Bedienungsfehler: (un)bewusste Fehleingaben des Benutzers
- Wartungsfehler

### Entstehungsort

#### Hardware fehler

Umfassen alle Entwurfs-, Herstellungs-, Betriebsfehler

#### Softwarefehler

Umfassen alle Fehler die in Programmteilen entstehen

#### Fehlerdauer

##### Permanente Fehler

bestehen ab Auftreten bis geeignete Reparatur-/Fehlertoleranzmaßnahmen ergriffen werden

##### Temporäre Fehler

- treten nur vorübergehend auf
- entstehen evtl. mehrmals spontan, verschwinden wieder

#### Struktur - Funktions - Modell

##### gerichteter Graph

Knoten = Komponenten

Kante von  $k_i$  zu  $k_j$  =  $k_j$  nutzt Funktion von  $k_i$

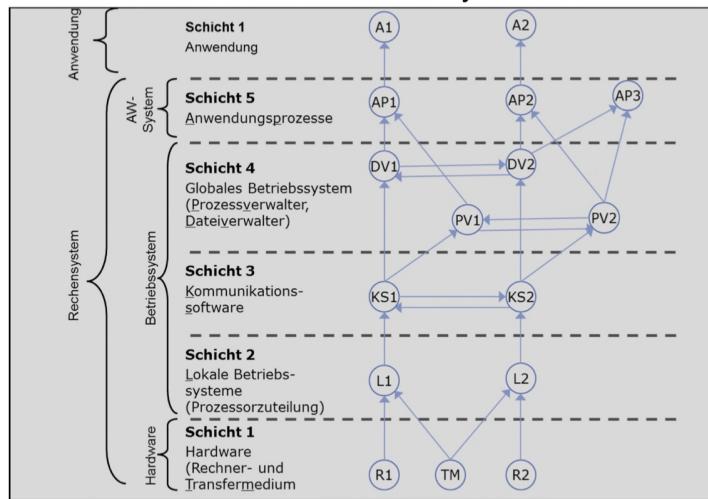
System = Komponentenmenge wenn Funktion nach außen in äußerer Spezifikation festgelegt

Subsystem = System das Teilmenge eines anderen

## Schichtenmodell

- Partitionierung der Komponenten in disjunkte Schichten
- Funktionszuordnung nur von niedrig nach hoch bzw. Schichtintern

Schichtenmodell  
Z-Rechnersystem



## Fehlermodell

- beschreibt mögl. Fehlerzustände eines Systems

## Binäres Fehlermodell

Fehlerzustandsfunktion  $z$  gibt für jede Komponente und das System an ob sie fehlerfrei sind

$$z : (\{S\} \cup \{\emptyset\}) \rightarrow \{0, 1\}$$

0 = Fehler

1 = kein Fehler

- System seit Inbetriebnahme Fehlerfrei :  $z(S, t) = \bigwedge_{\tau \leq t} z(S_\tau, \tau)$

## Nicht redundantes System

- System nur fehlerfrei wenn alle Komp. fehlerfrei
- $$z(S) = z(K_1) \wedge \dots \wedge z(K_n)$$

## Systemfunktion $f(K_1, \dots, K_n)$

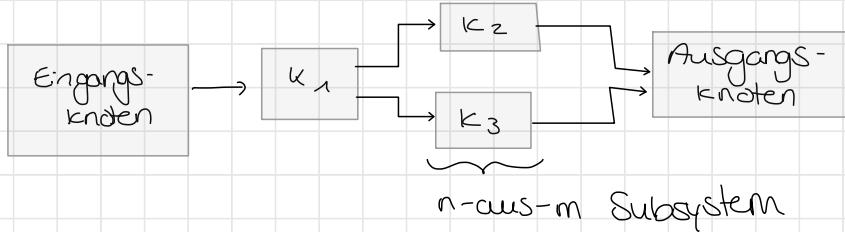
- gibt an wie sich Funktion des Systems aus Funktion der einzelnen Komponenten ableitet

## Zuverlässigkeitssymbolik

• gerichteter Graph mit einem Eingangs-, einem Ausgangsknoten

$$\text{zB } S = K_1 \wedge (K_2 \vee K_3)$$

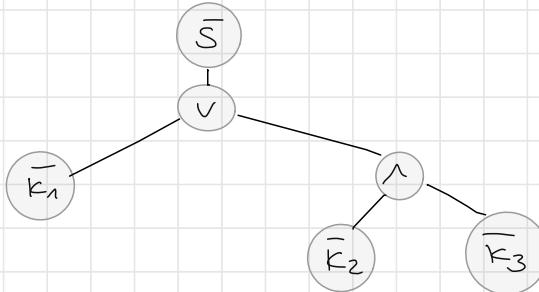
$$Z(S) = Z(K_1) \wedge (Z(K_2) \vee Z(K_3))$$



## Fehlerbaum

Strukturbaum der Negation der Systemfunktion

$$S = K_1 \wedge (K_2 \vee K_3) \rightsquigarrow \bar{S} = \bar{K}_1 \vee (\bar{K}_2 \wedge \bar{K}_3)$$



## Fehlerbereich

Fehlerbereich  $\mathcal{B} \subset S$  = Menge von Komponenten die zugleich fehlerhaft sein müssen ohne dass  $S$  fehlerhaft wird

$$\text{also: } \forall K \in S - \mathcal{B} : Z(K) = \text{wahr}$$

$$\Rightarrow Z(S) = \text{wahr}$$

$$\text{zB } S = K_1 \wedge (K_2 \vee K_3) \rightarrow \mathcal{B}_1 = \{K_1\}, \mathcal{B}_2 = \{K_3\}, \mathcal{B}_3 = \{K_2\}$$

Einzelfehlerbereich = Menge der Komponenten des gleichen Fehlerbereichs  
Perfektionskern = Komplement der Vereinigung aller Fehlerbereiche

## Ausfallverhalten

### Teilausfall

von einer fehlerhaften Komponente fallen eine oder mehrere, aber nicht alle Funktionen aus

### Unterlassungsausfall

fehlerhafte Komponente gibt eine Zeit lang keine Ergebnisse aus, ausgegebene Ergebnisse sind aber korrekt

### Anhalteausfall

fehlerhafte Komponente gibt nie mehr Ergebnis aus

### Haftausfall

fehlerhafte Komponente gibt ständig gleichen Ergebniswert aus

### Binärstellenausfall

Fehler verfälscht eine/mehrere Binärstellen des Ergebnisses

### Fail-Stop System

System, dessen Ausfälle nur Anhalteausfälle sind

### Fail-silent System

System, dessen Ausfälle nur Unterlassungsausfälle sind

### Fail-safe System

System, dessen Ausfälle nur unkritische Ausfälle sind

### Folgefehler

Fehlerursache von Komponente J ist Fehler von Komponente I

## Maßnahmen zur Fehlereingrenzung

- vertikale Fehlereingrenzung von höheren auf niedrigere Schichten
  - niedrigere Schichten prüfen Funktionsaufträge vor Ausführung
- vertikale Fehlereingrenzung von der HW auf höhere Schichten
- vertikale Fehlereingrenzung von niedrigeren auf höhere SW-Schichten
  - Plausibilitäts-, Konsistenzprüfung der Ergebnisse in höheren Schichten
- horizontale Fehlereingrenzung in lokalen Schichten
- horizontale Fehlereingrenzung in globalen Schichten

## Anforderungen Fehlertoleranz

- hohe Überlebenswahrscheinlichkeit
- hohe mittlere Lebensdauer
- hohe Verfügbarkeit
- hohe Sicherheitswahrscheinlichkeit
- hohe Sicherheitsdauer

## Fehlertoleranzverfahren

### Konstruktionsaspekte

- Ableiten der Fehlervorgabe: Fehlermodell + Menge der zu tolerierenden Fehler
  - aus angenommenen Fehlerarten der Komponenten
  - aus Zuverlässigkeitserfordernissen an Gesamtsystem

### Menge der zu tolerierenden Fehler:

- welche Fehler sind im Fehlermodell zu tolerieren
- Formulierung bzgl. Fehlerbereichsannahme: wie viele Einzelfehlerbereiche können gleichzeitig fehlerhaft werden, welche Fehlfunktionen sind zu behandeln

- Behandlung mehrerer aufeinander folgende Fehler: Zeitintervall in dem keine zusätzl. Fehler auftreten bevor Fehlerbehandlung abgeschlossen  
↳ Zeitredundanz
- Fehlerbehandlungsdauer: Zeit die Fehlerbehandlungsverfahren benötigt um Fehler zu behandeln  
! muss kleiner Zeitredundanz sein
- zusätzliche Anforderungen:
  - Nachweis Fehlertoleranzfähigkeit
  - geringer Betriebsmittelbedarf (geringe Kosten)
  - schnelle Ausführung von Fehlertoleranzverfahren (Leistung)
  - Unabhängigkeit von der Anwendungsoftware (Transparenz)
  - Unabhängigkeit vom Rechensystem

## Zuverlässigkeitsskenngrößen

Zuverlässigkeit, Sicherheit quantifizierbar mittels stochastischer Modelle

→ nicht-neg. zu

Lebensdauer  $L$  - Dichte  $f_L(t)$

Fehlerbehandlungsdauer  $\beta$  - Dichte  $f_B(t)$

Sicherheitsdauer  $\delta$  - Dichte  $f_D(t)$

## Fehlerwahrscheinlichkeit $F_f(t)$

- Wkt dass fehlerfreies System im Intervall  $[0, t]$  fehlerhaft wird
- $$F_f(t) = \frac{N_f(t)}{N} = \frac{\# \text{ fehlerhafter Komp. bis Zeitpunkt } t}{\# \text{ Komponenten}}$$

$$\text{Dichte } f_f(t) = \frac{d}{dt} F_f(t) = - \frac{d}{dt} R(t) = - \frac{1}{N} \cdot \frac{d}{dt} N_S(t)$$

## R(t) = Überlebenswahrscheinlichkeit

- Wkt dass fehlerfreies System bis Zeitpunkt  $t$  fehlerfrei bleibt

$$R(t) = \frac{N_S(t)}{N}$$

$$N_f(t) = N - N_S(t)$$

$$= 1 - F_f(t)$$

$$F_L(t) = 0$$

$$\lim_{t \rightarrow \infty} F_L(t) = 1$$

$$R(0) = 1$$

$$\lim_{t \rightarrow \infty} R(t) = 0$$

beide in  $t$  monoton wachsend

### Ausfallrate $z(t)$

- Anteil der in einer Zeiteinheit ausfallenden Komponenten bezogen auf Anteil der noch fehlerfreien Komponenten

Gesamtzahl zu erwartenden ausgefallenen Komp. zum Zeitpunkt  $t$ :

$$f_L(t) \cdot N$$

$$\hookrightarrow z(t) = \frac{f_L(t)}{R(t)} = -\frac{1}{R(t)} \cdot \frac{d}{dt} R(t)$$

$\Rightarrow F_L(t)$  ist Lösung von

$$F'_L(t) = f_L(t) = z(t) \cdot R(t) = z(t) \cdot (1 - F_L(t))$$

$$\text{mit } F_L(0) = 0$$

$$\hookrightarrow F_L(t) = 1 - e^{-\int_0^t z(s) ds}$$

mit konst. Ausfallrate  $z(t) = \lambda$

$$\hookrightarrow F_L(t) = 1 - e^{-\lambda t}$$

### Verfügbarkeit

- Wkt dass System zu beliebigem Zeitpunkt fehlerfrei

$$V = \frac{\mathbb{E}(U)}{\mathbb{E}(L) + \mathbb{E}(B)}$$

### Gefährdungswahrscheinlichkeit $F_0(t)$

- Wkt dass sicheres System in Zeitintervall  $[0, t]$  in gefährlichen Zustand gerät

## Sicherheitswahrscheinlichkeit

• Wkt dass sicheres System bis Zeitpunkt  $t$  ununterbrochen sicher bleibt

$$S(t) = 1 - F_0(t)$$

## Mittlere Sicherheitsdauer

$$E(D) = \int_0^\infty t \cdot f_D(t) dt = \int_0^\infty S(t) dt$$

## Funktionswahrscheinlichkeit $\varphi$

System  $S = f(K_1, \dots, K_n)$   $K_i \in \{0, 1\}$  = Fehlerzustand der Komponente  $i$   
 $f^{-1}(1)$  = Menge der Kombinationen von Fehlerzuständen  
 der Komponenten des Systems  $S$  für die  $K_i = 1$

$$\varphi(S) = \sum_{(K_1, \dots, K_n) \in f^{-1}(1)} \varphi(\Lambda_{i=1}^n K_i)$$

## Nichtfunktionswahrscheinlichkeit

$$\varphi(\bar{E}) = 1 - \varphi(E)$$

## Funktionswahrscheinlichkeit für Seriensystem und Parallelsystem

$$\text{Seriensystem } \varphi(\Lambda_{K \in \Lambda} 1) = \prod_{K \in \Lambda} \varphi(K)$$

$\Lambda$  = Menge von kmp./  
Systemfunktionen

$$\text{Parallelsystem } \varphi(\vee_{K \in \Lambda} 1) = \sum_{\emptyset \neq A \subseteq \Lambda} (-1)^{\Lambda + \#A} \cdot \varphi(\Lambda_{K \in A} K)$$

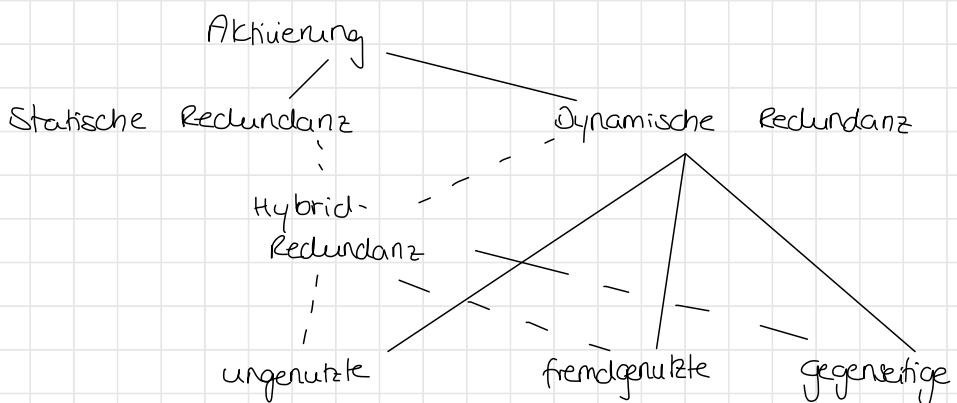
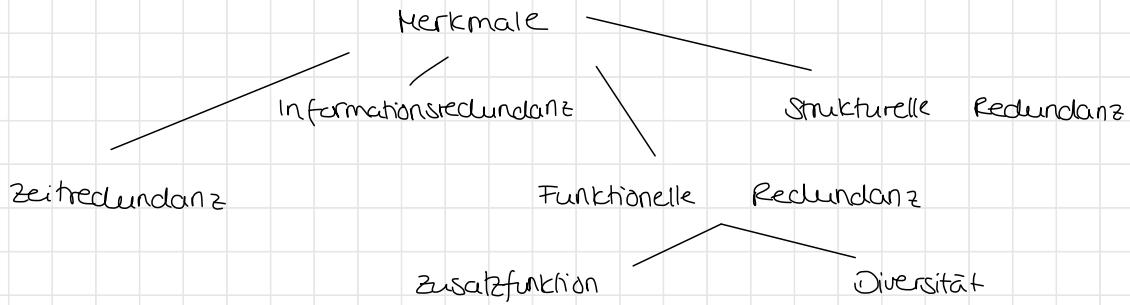
$$\text{System } S = K_1 \cup K_2$$

$$\varphi(S) = \varphi(K_1 \cup K_2) = \varphi(K_1) + \varphi(K_2) - \varphi(K_1 \cap K_2)$$

## Zuverlässigkeitserhöhung

$$\varphi_{S_1 \rightarrow S_2} = \frac{\varphi(\bar{S}_1)}{\varphi(\bar{S}_2)} = \frac{1 - \varphi(S_1)}{1 - \varphi(S_2)}$$

# Redundante Systeme



## Dynamische Redundanz

- redundante Mittel werden erst nach Auftreten eines Fehlers aktiviert  
↳ Unterscheidung Primär-, Ersatzkomponenten

## Ungenutzte Redundanz

- Ersatzkomp. haben keine sonstige Funktion
- bleiben passiv bis zur Aktivierung

## fremdgenutzte Redundanz

- Ersatzkomp. erbringen Funktionen die nicht zum betreffenden Subsystem gehören

## gegenseitige Redundanz

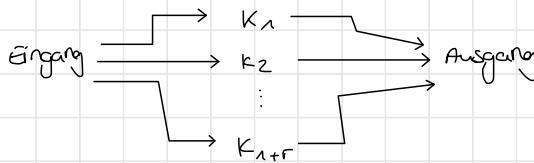
- Ersatzkomp. erbringen die von einer anderen komp. zu unterstützende Funktion  
↳ gegenseitig als Reserve → ermöglicht abgestuften Leistungsabfall

## Statische Redundanz

- redundante Mittel erbringen dauerhaft gleiche Nutzfunktion

## Verbesserung durch Redundanz

### Parallelsystem



ungenutzt fremdgenutzte Red.

Systemfunktion  $S_{1+r} = K_1 \cup \dots \cup K_{1+r}$

Funktionswkt  $\varphi(S_{1+r}, t) = 1 - \prod_{i=1}^{1+r} (1 - \varphi(K_i, t))$

Konst. Ausfallrate  $\lambda = \varphi(S_{1+r}, t) = 1 - (1 - e^{-\lambda t})^{1+r}$

Zuverl.verbesserung  $\Phi_{S_1 \rightarrow S_{1+r}} = (1 - e^{-\lambda t})^r$

### Serialsystem

Eingang  $\rightarrow K_1 \rightarrow K_2 \rightarrow \dots \rightarrow K_n \rightarrow$  Ausgang

$$S_n = K_1 \wedge \dots \wedge K_n$$

Zuverlässigkeit  $\varphi(S_n, t) = \prod_{i=1}^n \varphi(K_i, t)$

## Statisch redundantes System

- statische Redundanz falls zu geringe Fehlerfassung oder keine wiederholten Berechnungen mögl.
- ↪ mehrere Komp. führen gleiche Berechnung aus
  - ↪ vgl. Ergebnisse, wahlredundantes
- max. f fehlerhafte Komp. können übereinstimmt werden wenn insg.  $m = 2 \cdot f + 1$  Komp. gegeben

## Klassifikation Rechnerarchitekturen

M. Flynn: 2dim. Klassifizierung

a) Zahl der Befehlsströme

b) Zahl der Datenströme

SISD: Single Instruction - Single Data

zB Uniprozessor, Von-Neumann-Rechner

SIMD: Single Instruction - Multiple Data

zB Vektorrechner

MISD: Multiple Instruction - Single Data

∅

MMD: Multiple Instruction - Multiple Data

zB Multiprozessoren

## Prinzip der Virtualität

- mit versch. Techniken, Mechanismen in der HW können auf einer Maschine versch. parallele Progr. Modelle unterstützt werden
- zugrunde liegende Organisation + Architektur weitgehend transparent

# PARALLELVERARBEITUNG

Nebenläufigkeit = Maschine arbeitet nebenläufig wenn Objekte vollständig gleichzeitig abgearbeitet werden

Pipelining = Bearbeitung eines Objekts kann in Teilschritte zerlegt und diese in sequentiellen Folgen ausgeführt werden  
Phasen für versch. Objekt können überlappt abgearbeitet werden

## Ebenen der Parallelität

### Programmebene

- parallele Verarbeitung versch. Programme
- vollständig unabh. Einheiten
  - keine gemeinsamen Daten
  - wenig/keine Komm. /Sync.
- Organisation von OS

### Prozessebene (Tasks)

- Programm zerlegt in parallel ausführbare Prozesse
- Prozess = heavy-weight process, zB UNIX Prozesse
- Sync. + Komm.
- OS unterstützt Parallelverarbeitung durch Prozessverwaltung, -sync., -komm.

### Blockebene (Threads / Blöcke)

- Thread = light-weight process
- gemeinsamer Adr. Raum → Komm. über gemeinsame Daten
- Sync über mutex, condition variables + weitere sync.mechanismen

### Anweisungs-/Befehlsebene

- parallele Ausführung einzelner Maschinenbefehle (elementarer Anweisungen)
- Analyse der sequentiellen Befehlsfolge
  - ↳ umordnen, parallelisieren

## Suboperationsebene

elementare Anweisungen zerlegen in Suboperationen → parallel ausführen

## Körnigkeit

- ergibt sich aus Verhältnis Rechenaufwand zu Komm-, Sync-Aufwand
- bemessen an Anzahl Befehle in sequentieller Befehlsfolge

Progr. / Proz. / Block: grobkörnig

Anw. / Subop.: feinkörnig

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

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

## Pipelining auf Maschinenbefehlsebene

Befehlspipelining = Zerlegung der Ausführung einer Maschinenoperation in Teilphasen, die dann von hintereinander geschalteten Verarbeitungsphasen takt synchron bearbeitet werden  
↳ jede Einheit führt genau eine spezielle Op aus

# RISC Reduced Instruction Set Computers

- Einfache Maschinenbefehle: einheitliches, festes Befehlsformat
- Load / Store Architektur: Befehle arbeiten auf Registeroperanden  
Speicherzugriff durch Load-, Speicherbefehle
- Einzyklus-Maschinenbefehle:
  - einheitliches Zeitverhalten (außer Load/Speicher-/Verzweigungsbefehle)
  - ↳ effizientes Pipelining
- optimierende Compiler: Reduzierung der Befehle im Programm

Pipeline:

1 Takt = 1 Stufe



Inst <sub>i</sub>	IF	ID	EX	MA	WB
Inst <sub>i+1</sub>		IF	ID	EX	MA    WB
Inst <sub>i+2</sub>			IF	ID	EX    MA    WB
Inst <sub>i+3</sub>				IF	ID    EX    MA    WB
Inst <sub>i+4</sub>					ID    EX    MA    WB

IF = Instruction fetch = Befehl aus Register holen

ID = Instruction decode

EX = Execution

MA = Memory Access

WB = Write Back

Leistungsaspekte obere Grenze Durchsatz:  $IPC \leq 1$  bzw  $CPI \geq 1$

Ausführungszeit

- Zeit die zum Durchlaufen der Pipeline nötig
- Ideal: Ausführung eines Befehls in  $K$  Taktzyklen  
gleichzeitige Behandlung von  $K$  Befehlen

Latenz

- Anzahl Zyklen zw. Op die Ergebnis produziert und Op die Ergebnis verwendet

Laufzeit  $T = n + k - 1$  (ideal)

$n = \#$  Befehle im Programm

## Beschleunigung

$$S = \frac{n \cdot k}{T} = \frac{n \cdot k}{n+k-1}$$

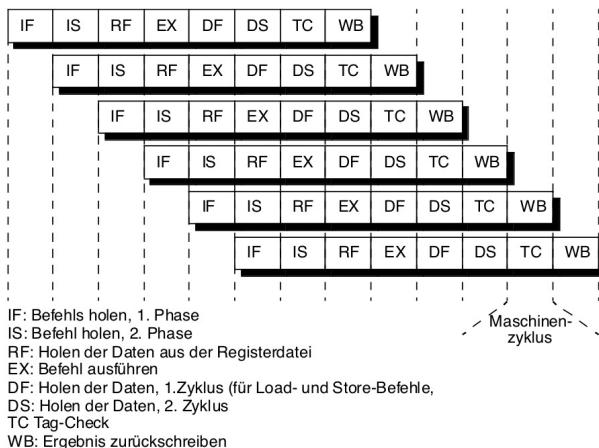
## Superpipelining

Verfeinerung der Pipeline - Stufen

- ⊕ weniger logik- Ebenen pro Stufe
- Erhöhung der Taktrate

- ⊖ Erhöhung der Ausführungszeit pro Instruktion

zB MIPS R4000



## Pipeline - Konflikte

- Situationen die verhindern, dass nächste Instruktion im Befehlsstrom im zugewiesenen Taktzyklus ausgeführt wird
- verursacht Leistungseinbußen

## Strukturkonflikte

- Resultat von Ressourcenkonflikten
- HW kann nicht alle mögl. Kombis von Befehlen unterstützen
  - ↪ zB gleichzeitige Schreibzugriffe auf Register mit 1 Schreibeingang

## Datenkonflikte

- Resultat von Datenabhängigkeiten zw. Befehlen
- Instruktion benötigt Ergebnis noch nicht fertiger Instruktion

## Steuerkonflikte

- bei verzweigungsbefehlen und anderen Befehlen die Befehlszähler ändern

## Lösungen

- Pipeline Stall = Anhalten der Pipeline
- Pipeline Bubble = Einfügen von Leerzyklen

### (2) Leistungseinbußen

bei Stall müssen alle Befehle danach auch warten  
↳ Backward Propagation of Stalling

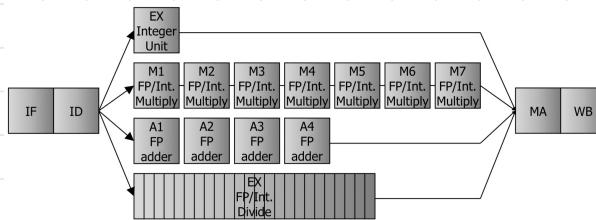
## Nebenläufigkeit

Verbesserungsmöglichkeiten skalarer Pipelines:

### - Exec - Phase:

Integer - Verarbeitung : logische + arithm. Befehle dauern 1 Taktzyklus ☺

Gleitkomma :  
- Zerlegung in weitere Stufen  
- mehrere Floating - Point - Units



→ arithmetisches Pipelining

## statische Ansätze

- VLI = very long instruction word
- EPIC = Explicitly Parallel Instruction Computer

## dynamische Ansätze

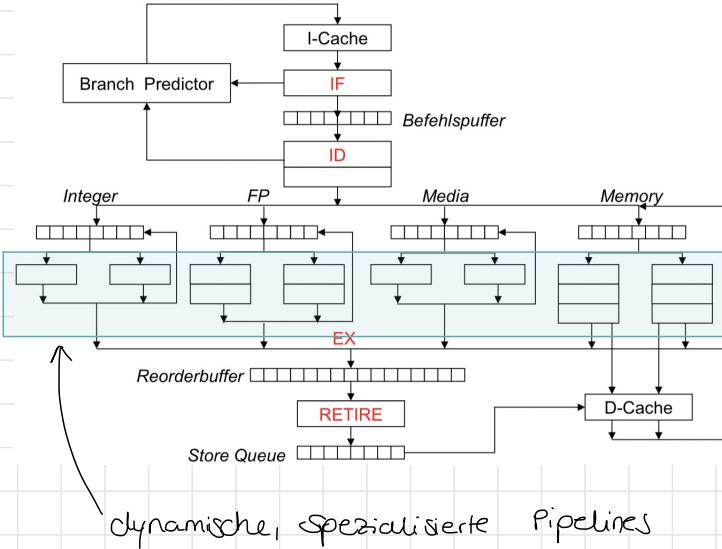
Superskalartechnik

# Superskalar technik

Ziel: Erhöhung des IPC

- mehrere unabh. Ausführungseinheiten
  - ↳ multiple issue: pro Takt mehrere Befehle ausführen, beenden
- HW: dynamische Erkennung, Auflösung von Konflikten zw. Befehlen

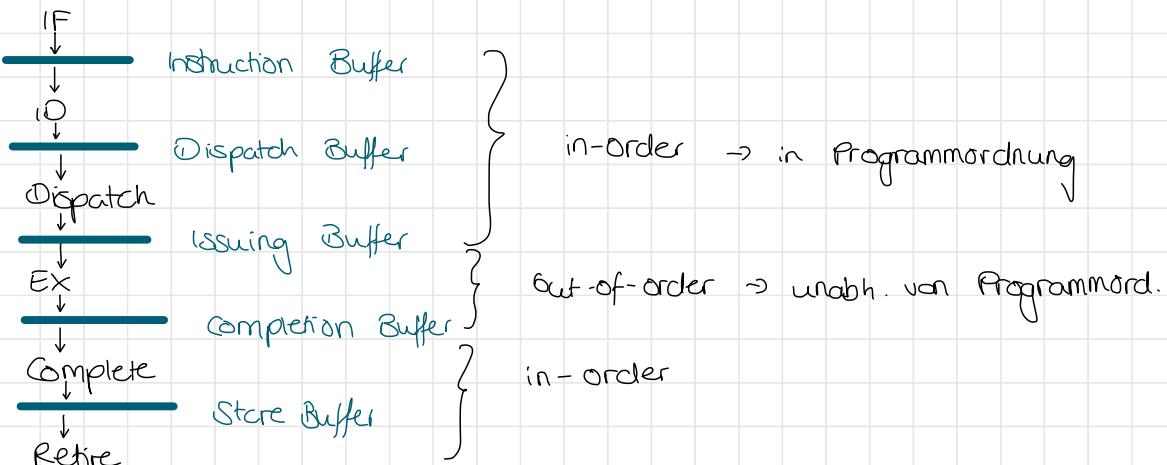
## Superskalarer Prozessor



## Komponenten

- Instruction Fetch Unit
- Instruction Decode Unit
  - ↳ mit Register renaming
- Instruction issue (Zuordnung)
- unabh. functional units
- Retire unit (Rückordnung)
- Register:
  - Allzweck
  - Multimedia
  - Speziell

## Prozessorpipeline



### 1) In-order Abschnitt

- IF, ID, Dispatch
- dynamische Zuordnung Befehle  $\rightarrow$  Ausführungseinheiten
- Scheduler bestimmt # Befehle die im nächsten Takt zugewiesen werden können

### 2) Out-of-order Abschnitt

- Exec

### 3) In-order Abschnitt

- gültigmachen der Ergebnisse entspr. Programmordnung  $\rightarrow$  retire
- Erhalten der korrekten Programmsemantik
  - Ausnahmeverarbeitung, Spekulation

## IF-Phase

### Befehlsbereitstellung

- holen mehrerer Befehle aus Befehls-Cache in Befehlsholpuffer
- #  $\hat{=}$  Zuordnungsbandbreite
- welche abh. von branch prediction

### Verzweigungseinheit

- überwacht Ausführung von Verzweigungen, Sprungbefehlen
- Branch prediction:
  - Spekulative Befehlsholen
  - Fehlspkulation: gewährleistet Rückrollen falscher Befehle

### Befehlsholpuffer

- entkoppelt IF und ID Phase

### Branch Prediction

$\approx$  jeder 5.-7. Befehl - bedingter Sprung

$\hookrightarrow$  Vorschage Sprungziel um möglichst gute Pipeline-Auslastung

- füllt Verzweigungsphasen mit Spekulationen

Korrekt: fertfahrt mit Ausführung

Falsch: Befehle verwirfen

## > Statische Vorhersage:

- Richtung der Vorhersage pro Befehl immer gleich
- ∅ unflexibel  $\rightarrow$  ungeeignet

## > dynamische Vorhersage:

- Richtung abh. von Vorgeschichte
- Berücksichtigung Programmverhalten
- ∅ hoher Hw-Aufwand

## Sprungziel-Cache

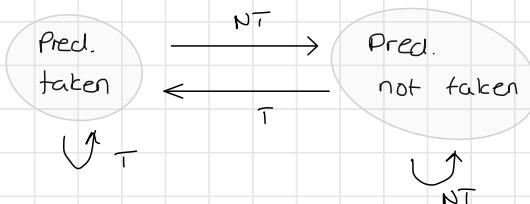
- Branch Target Address Cache, Branch Target Buffer
- speichert Adr. der Verzweigung + entspr. Sprungziel

## Sprungverlaufstabelle

- Branch History Table
- Prädiktoren: Festhalten des Verhaltens der Sprungbefehle während Programmausführung
- Vorhersage Verhalten eines geholten Sprungbefehls

Adresse der Verzweigung	Sprungziel-adresse	Vorher-sagebits

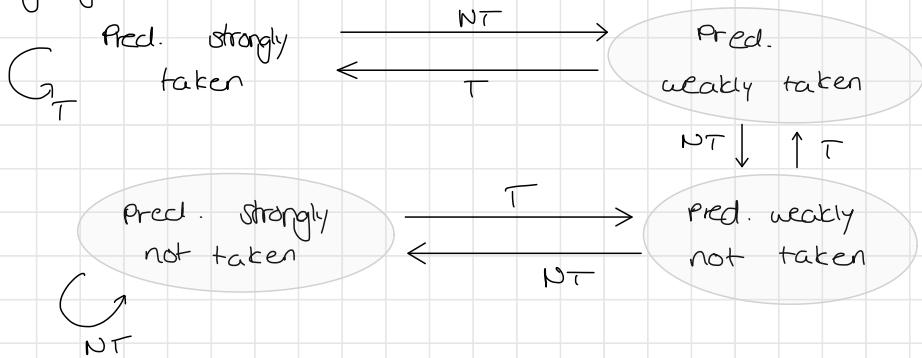
Vorhersagebit : 1 = Sprung wird genommen  
 $\hookrightarrow$  Fehlannahme: Bit invertieren



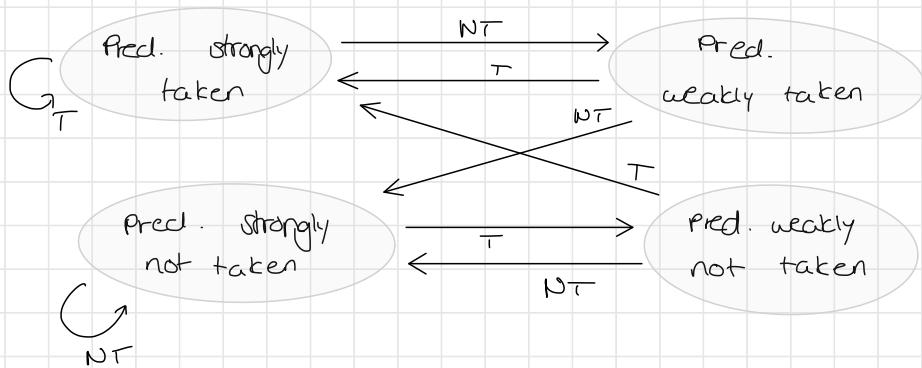
## 2-Bit-Predictor

- 4 Zustände:
  - sicher genommen
  - vielleicht genommen
  - vielleicht nicht genommen
  - sicher nicht genommen

### a) Sättigungszähler



### b) Hysterese methode



### 10 - Phase

- Dekodierung der Befehle im Befehlspuffer
- # = Befehlsbereitstellungsbandbreite
- CISC: mehrere Schritte
  - 1) Bestimmung Grenzen der Befehle
  - 2) Dekodierung
  - 3) Generierung RISC ähnlicher Ops (dynamische Übersetzungstechniken)
- Registerumbenennung:
  - Umbenennung der Operanden-, Ergebnisregister
  - ↳ Auflösung von Konflikten durch Namensabhängigkeiten RAW, WAR, WAW
  - ↳ 2 Instruktionen nutzen dasselbe Register, aber eigentlich kein Datenfluss

## Anti-Dependence

ADD R2, R3, R4  
 XOR R3, R5, R6

## Output-Dependence

ADD R2, R3, R4  
 XOR R2, R5, R6

- Schreiben der Befehle in Instruction Window

## Dispatch-Phase

- Zuordnung wartender Befehle zu execution units
- Zuordnung bis max. Zuordnungsbandbreite pro Takt
- dynamische Auflösung von Konflikten von echten Datenabh., Ressourcenkonflikten

## Echte Datenabh.

Befehl j datenabh. von Befehl i fals

(1) i produziert Ergebnis das j verwendet

oder (2) j datenabh. von k, k datenabh. von i

## Umordnungsbuffer: Reservierungstabellen

Zuordnung nur wenn Platz frei, ansonsten müssen nachfolgende Befehle warten

## Rückordnungsbuffer: reorder buffer

festhalten ursprünglicher Befehlsordnung

jeweils Ausführungsstand der Befehle protokollieren

## Befehlausführung

- Ausführung der im Opcode spez. Op
- Speichern des Ergebnisses im Zielregister
- 1-Zyklus - Ops oder Mehrzyklus - Ops

## Completion

- Instruction beendet Ausführung wenn Ergebnis für nachfolgende Befehle bereitgestellt
- Bereinigung Reservierungstabellen
- Aktualisierung Zustand Rückordnungsbuffer

## Rückordnungsstufe (Retire)

- Commitment: · nach completion beenden Befehle Bearbeitung
- Ergebnisse in Architekturregister persistiert (WB)

Bedingungen für Commitment:

- Befehlausführung ist vollständig
- alle Befehle im Progr. vor Befehl sind fertig/ enden im selben Takt
- Befehl unabh. von Spekulationen
- keine Unterbrechung vor / während Ausführung

Precise Interrupts: bei Auftreten einer Unterbrechung

- alle Resultate von Befehlen in Progr.ordnung vor Ereignis werden gültig gemacht
- nachfolgende Resultate werden verworfen
- Resultat des verursachenden Befehls: gültig/ verworfen je nach Arch./ Art der Unterbrechung

## Tomasulo Algorithmus

- Anstoßen der Befehle dezentral aus den Reservation Stations
- Einlesen der Operanden dezentral über Common Data Bus
- dezentrales Result Forwarding

Reservierungstabelle = reservation station

Empty	InFu	Op		Dest	Src1	Vld1	RS1		Src2	Vld 2	RS2
-------	------	----	--	------	------	------	-----	--	------	-------	-----

Src1, Src2 : Werte der Quelloperanden

Vld1, Vld 2: Flag ob Operand verfügbar

RS1, RS2: falls Operand noch berechnet: wer liefert Operanden?

Dest: Name für Ziel

Empty: ist Reservation Station leer

InFu: wird Op gerade ausgeführt

Op: auszuführende Operation

### 3 Phasen:

- 1) Issue: Instruktion laden, wenn RS frei: Instruktion zuweisen, Operanden laden
- 2) Execution: Befehl ausführen, wenn beide Operanden verfügbar Befehl an FE, ausführen
- 3) Write Result: Write back, RS freigeben

R	registers					
	1	2	3	4	5	6
Value	-	(R4)-(R3)	(R3)	-	(R5)	-
Vld	0	1	1	0	1	0
RS	3	0	0	2	0	4

register status

```

mul Reg1, Reg3, Reg5
sub Reg2, Reg4, Reg3
div Reg6, Reg1, Reg4
add Reg4, Reg2, Reg3

```

reservation stations	S <sub>add</sub>	Empty	InFU	Op	Dest	Src1	Vld1	RS1	Src2	Vld2	RS2
		1	1	sub	2	(R4)	1	0	(R3)	1	0
	S <sub>mul</sub>	0	0	add	4	(R4)-(R3)	1	0	(R3)	1	0
	S <sub>div</sub>	0	1	mul	1	(R3)	1	0	(R5)	1	0
		0	0	div	6		0	3	(R4)	1	0

RS status

cycle 4

token.tag 1  
token.data (R4)-(R3)

### Very long Instruction Word

- breites Befehlsformat aufgeteilt in mehrere Felder
- n unabh. Op-einheiten  $\rightarrow$  n parallele Befehle
- Steuerung der parallelen Abarbeitung: automatisch parallelisierende Compiler

### Multithreading

Ziel: Reduzierung der Untätigkeits-, Latenzzeit

zB durch Speicherzugriffe bei Cache-Misses

$\rightarrow$  Threads parallel ausführen

Prinzip: geg.: mehrere geladene Threads

- Kontext muss pro Thread gesichert werden können
- mehrere getrennte Registersätze, mehrere Befehlszähler
- getrennte Seitentabellen
- Threadwechsel bei Warten

## Cycle-by-cycle Interleaving feingranulares Multithreading

- Prozessor wählt in jedem Takt einen der ausführbaren Threads aus
- nächster Befehl in Befehlsreihenfolge wird ausgeführt

- ⊖ Verarbeitung eines Threads wird verlangsamt wenn er ohne Wartezeiten ausgeführt werden könnte

{ } { } { } { }  
① ② ③ ④      ; = context switch

Scalar:    1 ; 2 { 3 | 4 | 1 ; 2 { 3 | 4

Superscalar:  
$$\begin{array}{ccccccccccccc} 1 & ; & 2 & ; & 3 & ; & 4 & ; & 1 & ; & 1 & ; & 2 & ; & 1 & \dots \\ 1 & ; & 2 & ; & 1 & ; & 4 & ; & 1 & ; & 1 & ; & 2 & ; & 1 & \\ 2 & ; & 1 & ; & 4 & ; & 1 & ; & 1 & ; & 2 & ; & 1 & ; & 4 & \\ & & & & & & & & & & & & & & & \end{array}$$

## Block Interleaving

- ein Thread ausführen bis Instruktion mit langer Wartezeit, dann context switch
- ⊕ Bearbeitung eines Threads nicht verlangsamt
- ⊖ bei Thread-wechsel leeren + Neustart der Pipeline  
→ nur bei langen Wartezeiten sinnvoll

scalar:    1 1 1    ; 2 ...

## Simultaneous

- mehrfach superskalater Prozessor
- mehrere Befehlspuffer
  - jeder stellt anderen Befehlstrom dar
  - jeder hat eigenen Registersatz

$$\begin{array}{ccccc} 1 & 4 & 2 & 2 & 3 \\ 3 & 4 & 2 & 4 & 4 & \dots \\ 3 & 4 & 2 & 1 & 4 \\ 3 & 2 & 3 & 1 & 1 \end{array}$$

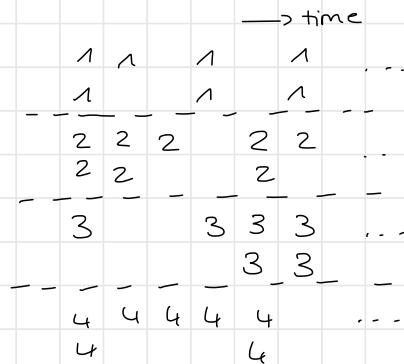
## Übersicht andere Prozessortechniken

Single-threaded scalar: 1 1 1 1 ...

Superscalar:  
1 1 1 1 ...  
1 1 1  
1  
1  
1

VL(W)  
1 1 1 . 1 ...  
1 N 1 1 ...  
N N 1 1  
N N 1 N

Single-chip multiprocessing:



## PARALLELRECHNER: MULTI PROZESSOREN

Parallelismus auf Prozessor-, Blockebene

### shared-memory Multiprozessor

- gemeinsamer Speicher
- komm. über geteilte Variablen → atomare Synchr. operationen
- UMA = Uniform Memory Access

### distributed-memory Multiprozessor

- verteilter Speicher
- komm. über Nachrichtenaustausch → Send/Receive Operationen
- NUMA: non-uniform memory access
- CC-NUMA: Cache-coherent NUMA

## Parallele Programmiermodelle

Programmiermodell:

- Abstraktion einer parallelen Maschine auf der Anwender sein Programm formuliert
- Spezifiziert wie Teile des Programms parallel abgearbeitet werden  
wie Infos ausgetauscht werden  
welche Synchr. mechanismen vorhanden

### work partitioning

- ausgehend von sequentiellem Programm: parallelisierbare Teile finden  
↳ S1, S2 parallel ausführbar falls unabhängig

### Datenparallelismus

- Berechnungen versch. Datenelemente unabh.  
↳ zB Matrixmultiplikation
- ↳ SPMD Prinzip

### Funktionsparallelismus

- unabh. Funktionen auf versch. Prozessoren ausführen

## Koordination

- Synchronisation + Kommunikation der parallelen Threads
- Nachrichtenaustausch über gme. Speicher / explizite Nachrichten  
↳ Ⓛ zusätzlicher Zeitaufwand

## Primitive

### Shared - Mem:

CREATE(proc, args)

generiere Prozess der Ausführung bei Prozedur proc mit Argumenten args startet

G- MALLOC(size)

Allocation gem. Datenbereich der Größe size Bytes

LOCK(name)

Fordere wechselseitigen exklusiven Zugriff

UNLOCK(name)

Lock freigeben

BARRIER(name, number)

globale Sync für number Prozesse

WAIT\_FOR-END(number)

Warte bis number Prozesse terminieren

WAIT\_FOR-FLAG

Warte auf gesetzte Flags

↳ while (!flag);  
WAIT(flag)

spinlock  
blockieren

SET FLAG

Setze Flag, welche wartenden Prozess

## Message Passing

CREATE(proc)

Erzeuge Prozess der bei proc startet

SEND(src, size, dest, tag)

Send size Bytes von src nach dest mit tag Identifier

RECEIVE (buffer, size, src, tag) Empfange Nachricht mit ID tag von src Processor, legle size Bytes in buffer ab

BARRIER (name, number) Globale Sync. von number Prozessen

## Maßzahlen

### Parallelitätsprofil

x - Achse: Zeit

y - Achse: # paralleler Aktivitäten

↪ zeigt wieviele Tasks zu einem Zeitpunkt parallel ausführbar

### Parallelitätsgrad

PG(t) = # Tasks die zu Zeitpunkt t parallel ausführbar

### Parallelitätsindex

Mittlerer Grad der Parallelität

$$I = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} PG(t) dt$$

$$I = \frac{\sum_{i=1}^m i * t_i}{\sum_{i=1}^m t_i}$$

PG-Bereich  
Ausführungszeit

## Kennzahlen

P(1) = # auszuführender Tasks des Programms auf Einprozessorsystem  
P(n) = # --- auf Mehrprozessorsystem  
mit n Prozessoren

T(1) = Ausführungszeit auf Einprozessorsystem in Schritten  
T(n) = Mehrprozessorsystem mit n Prozessoren

Vereinfachung:  
 $T(1) = P(1)$  → jeder Task = ein Schritt  
 $T(n) \leq P(n)$  → pro Schritt > 1 Task

## Speedup, Effizienz

$$S(n) = \frac{T(1)}{T(n)}$$

üblicherweise  $1 \leq S(n) \leq n$

$$E(n) = \frac{S(n)}{n}$$

üblicherweise  $\frac{1}{n} \leq E(n) \leq 1$

absolut: algorithmenunabh.

$T(1)$  = bester bekannter sequentieller Algo

relativ: algorithmenunabh.

$T(1)$  = paralleler Algo der sequentiell ausführt

## Parallelisierungsaufwand

$$R(n) = \frac{P(n)}{P(1)}$$

$1 \leq R(n)$

Auslastung - normierter Parallelindex

$$U(n) = \frac{l(n)}{n} = R(n) \cdot E(n) = \frac{P(n)}{n \cdot T(n)}$$

wie viele Tasks hat jeder Prozessor im Schnitt pro Zeiteinheit durchgeführt

$$1 \leq S(n) \leq l(n) \leq n$$

$$\frac{1}{n} \leq E(n) \leq U(n) \leq 1$$

## Skalierbarkeit

- Hinzutigen weiterer Verarbeitungselemente führt zu kürzerer Gesamt-ausführungszeit ohne dass Progr. geändert werden muss
- lineare Steigerung der Beschleunigung mit Effizienz  $\approx 1$
- Skalierbarkeit ist begrenzt bei fester Problemgröße

## Amdahl's Gesetz

$a = \text{sequentieller Programmanteil}$

$$T(n) = T(1) \cdot \underbrace{\frac{1-a}{n}}_{\substack{\text{Ausführungszeit} \\ \text{paralleler Teil}}} + \underbrace{T(1) \cdot a}_{\substack{\text{Ausführungszeit} \\ \text{sequentieller Teil}}}$$

$$S(n) = \frac{T(1)}{T(n)} = \frac{1}{\frac{1-a}{n} + a} \xrightarrow{n \rightarrow \infty} \frac{1}{a}$$

↪ Speedup hängt erheblich vom sequentiellen Anteil ab

## Superlinearer Speedup

$S(n) > 1$  möglich durch:

- paralleles Backtracking
- Caching-Effekte

## Probleme

- Verwaltungsaufwand / Overhead
  - ↪ steigt mit Zahl der zu verwaltenden Prozessoren
- Möglichkeit von Deadlocks
- Möglichkeit von Sättigungsscheinungen

## Verbindungsstrukturen

- Netzwerk zur Komm., Kooperation zw. Verbindungs elementen
- ↪ zuverlässiger Informations austausch

## Charakterisierung

Kanal = phys. Verbindung (Leitungen) zw. Schalterelementen / Knoten mit Puffer zum Halten der Daten während Übertragung

### Latenz

Übertragungszeit einer Nachricht  $T_{msg}$

= Zeit zum Verschicken einer Nachricht bestimmter Länge zw. 2 Prozessoren

Startzeit  $t_s$  = Zeit für Komm. init.

Transferzeit  $t_w$  = pro übertragenem Datenwort, abh. von physik. Bandbreite des Komm. Mediums

### Kanalverzögerung

Dauer für Belegung des Komm. Kanals durch eine Nachricht

### Schaltverzögerung, Routing - Verzögerung

- Zeit um weg zw. 2 Knoten aufzubauen
- Routing = Pfadberechnung, Wegfindung

### Blockierungszeit

- Falls mehr als eine Nachricht auf Netzwerkressource zugreift

### Blockierung

Verbindungsnetzwerk blockierungsfrei falls jede gewünschte Verbindung zw. Prozessoren / Prot. + Speicher unabh. von bestehenden Verbindungen herstellbar

### Durchsatz, Übertragungsbandbreite

max. Übertragungsleistung des Verbindungsnetzwerks / einzelner Verbindungen  
MBit/s oder MB/s

## Bisektionsbandbreite

max. Anzahl MB/S die Netzwerk über Bisektionslinie, die Netzwerk in zwei gleiche Hälften teilt transportieren kann

## Diameter

max. Distanz für komm. Zer Prozessen, also Anzahl der Verbindungen die durchlaufen werden müssen

## Verbindungsgrad eines Knotens P

Anzahl direkter Verbindungen die von einem Knoten zu einem anderen bestehen

## Mittlere Distanz zw. 2 Knoten

Anzahl Links auf kürzestem Pfad zw. 2 Knoten

$\frac{P}{da} > \text{max. Anzahl neuer Nachricht die von jedem Knoten je Zyklus in Netzwerk einbringbar}$

## Komplexität / Kosten

- Kosten für HW Implementierung
- Aufwand für Verbindungsnetz gemessen in #, Art der Schaltelemente und Verbindungsteilungen

## Erweiterbarkeit

- Multiproz. können begrenzt, stufenlos oder nur durch Verdopplung der # Proz. erweiterbar sein

## Skalierbarkeit

Fähigkeit die wesentlichen Eig. des Verbindungsnetzes auch bei beliebiger Erhöhung der Knotenzahl beizubehalten

## Ausfalltoleranz, Redundanz

- Verbindungen schalten noch trotz Ausfall einzelner Netzelemente → mind einen redundanten Weg zw. jedem Knotenpaar

## Verbindungsnetzwerk IN

Verbindet  $\neq$  Knoten miteinander sodass Informationen von Quellknoten zu Zielknoten  $\Rightarrow$  verschickbar

### Knoten

- zB Cache - Modul, Speichermodul, Rechentknoten
- über Netzwerkschnittstelle (NIC) mit IN verbunden

### Switch - Schaltelement

- $n \times n$ :  $n$  Eingänge,  $n$  Ausgänge mit Grad  $n$
- setzt Verbindung zw. Ein-, Ausgang auf um Info zu übertragen

### Link

- verbindet Ausgang eines Switches/NIC mit Eingang eines Switches/NIC
- besteht aus Leitungen über die digitale Infos transportiert
- synchr. Übertragung:  
Links, Switches haben gleiche Taktquelle
- asynchron. Übertragung:  
komp. untersch. Takt, Sync. über Handshake
- Breite = # Bits die parallel pro Taktzyklus übertragbar
- Bandbreite =  $w \cdot t$  Bits pro Zeit mit Breite  $w$ , Taktzyklus  $t$

### Nachricht

- Informationseinheit die von Quelle zum Ziel gesendet

#### Arten der Übertragung

- Unicast: ein Knoten schickt Anforderungsnachricht an Zielknoten
- Multicast: an mehrere Knoten
- Broadcast: an alle Knoten

### Pakete

- aufteilen der Nachricht in Paket fester Länge

Trailer	Error-Code	Payload	$\left\{ \begin{array}{l} \text{Payload} \\ \text{Header} \end{array} \right.$
---------	------------	---------	--------------------------------------------------------------------------------

Header: · Routing - Info  
· Anforderungs-, Antworttyp

Payload: eigentliche Daten

Errorcode: Fehlerbehandlung, -erkennung, -korrektur

## Switching Strategy

Durchschalte, Leitungsumleitung = circuit switching

- direkte Verbindung zw. Knoten
- Paketübertragung zw. Sender, Empfänger ohne Unterbrechung
- Keine Routinginfo im Paket
- Netzwerkressourcen auf dem Weg geblockt

Paketvermittlung = packet switching

- Datenpakete entsprechend Routing algo verschicken
- Adr. wird mitgeschickt → Weiterleitung in den Switches
- Ressourcen nur kurz belegt

Flusskontrolle: bestimmt wann ein Paket von Switch zu nächstem Switch transportiert wird

## Latenz-, Bandbreitenmodelle

End-to-end packet latency model

- Betrachtet Übertragung eines Pakets vom Sender- zum Empfängerknoten

### E2E latency

· Zeit um gesamtes Paket zu übertragen und in Zielpuffer abzulegen

$N_p$  = # Bits des Payloads

$N_E$  = # Bits drumum (Trailer, ErrorCode, Header) = Envelope

→  $N_p + N_E$  zu übertragende Bits

↪ E2E packet latency = Sender OH + time of flight + transmission time  
+ routingtime + receiver OH

• Sender OH = Zeit für Vorbereiten des Pakets (Hinzufügen Envelope) + Ablegen in Sendepuffer

• Time of flight = untere Grenze um ein Bit vom Sender zum Empfänger zu schicken  $\rightarrow$  abh. vom Switching incl.

Switching time = abh. davon wie Paket zw. 2 Schaltelementen übertragen

- Store-and-forward: Paket vollständig nacheinander übertragen
- Cut-through: Übertragung überlappt

• Transmission time: zusätzliche Zeit zur Übertragung restl. Bits nachdem erstes ankam abh. von Linkbandbreite  $N_{ph} \cdot f$   
 $\hookrightarrow$  Zykluszeit  $1/f$ , Physical Transfer Unit: Bits pro Übertragungseinheit  
 $\hookrightarrow N_p + N_E$   
 $N_{ph} \cdot f$

• Routing time = abh. von Switching

Circuit: Zeit um weg aufzusetzen

Packet: Zeit um weg in jedem Schaltelement aufzusetzen

• Receiver Overhead = Ablegen der Verwaltungsinfo, Weiterleitung aus Empfangspuffer

$\rightarrow$  Nachricht größer als Payload eines Pakets?

$\hookrightarrow$  Sender OH + Time of flight + Receiver OH + Transm. time + Routing time  
 $+ (N-1) (\max(\text{Sender OH}, \text{transm. time}, \text{receiver OH}))$

## effektive Bandbreite

Paketgröße

$\max(\text{Sender OH}, \text{transm. time}, \text{receiver OH})$

## Switching Strategy

wie wird weg in Verbindungsnetzwerk aufgebaut

Modellannahmen:

- Pfad Quelle  $\rightarrow$  Ziel = Übertragungspipeline
- unterwegs muss Paket L Schaltelemente passieren
- Paket = N Phits (Physical Transfer Units)

## Circuit Switching

- Aufbau direkter Weg Quelle  $\rightarrow$  Ziel, dann Übertragung
  - Routing time = Zeit um 1 Phit von Quelle nach Ziel zu übertragen  
 $= L \cdot R + \text{time of flight}$   
 $= L \cdot R + L = L(R+1)$
- $R = \# \text{ Netzwerzyklen für Routing-Entscheidung in einem Schaltelement}$

$\hookrightarrow$  E2E packet latency = Sender OH +  $L(R+2)$  + N  $\cdot$  Receiver OH

⊖ Modell berücksichtigt nicht, dass während Übertragung Netzwerkressourcen belegt

## Packet switching

- Wegfindung wird je Paket entsprechend Routing algo je Schaltelement berechnet
- $\hookrightarrow$  Ressourcen nur solange belegt wie benötigt

## Store-and-forward

- Paket wird vollständig empfangen vor Weiterleitung  
 $\hookrightarrow$  Puffer je Knoten

Paket ist N Zyklen in einem Schaltelement

$\hookrightarrow L \cdot N$  Zyklen bis erstes Paket am Ziel

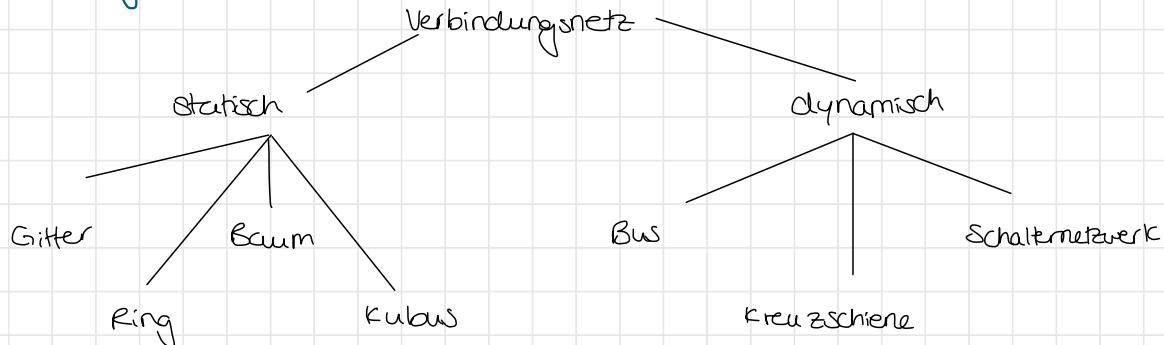
E2E packet latency = Sender OH +  $N(L+1)$  + LR + Receiver OH

## Cut-through

- Paket wird weitergeleitet ohne dass vollst. angekommen

$$\text{E2E packet latency} = \text{Sender OH} + L + N + LR + \text{Receiver OH}$$

## Topologien

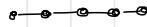


### Statische Verbindungsnetze

- feste Verbindungen nach Aufbau des Netzes
- vollständige Verbindung
- jeder Knoten mit jedem anderen verbunden
- höchste Leistungsfähigkeit
- nicht praktikabel in Parallelrechnern

### Gitterstruktur

1-dim Gitter = Kette

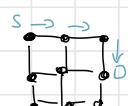


- verbindet  $N$  Knoten mit  $(N-1)$  Verbindungen
- Diameter  $r = N-1$
- disjunkte Bereiche gleichzeitig nutzbar
- mehrere Schritte nötig für Nachrichten zw. nichtbenachbarten Knoten

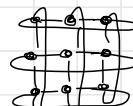
### $k$ -dim Gitter

mit  $N$  Knoten

2dim



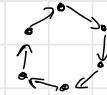
2dim - Torus



## Ring

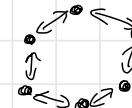
unidirektional

- Nachrichten nur in eine Richtung
- Diameter  $r = N - 1$
- Austall einer Verbindung  $\rightarrow$  Zusammenbruch komm.



## bidirektional

- Austall teilt Netz in 2 disjunkte Teile
- längster Pfad für Nachricht  $\leq N/2$



## chordaler Ring

- hinzufügen redundanten Verbindungen  
↳ höhere Fehlertoleranz



Knotengrad 3



Knotengrad 4

## Baumstrukturen

- binärer Baum mit  $m$ -Ebenen

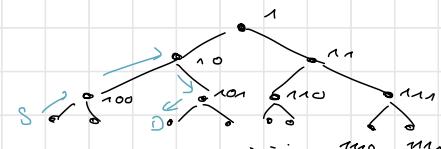
↳ Ebene  $m$ :  $N = 2^m - 1$  Knoten

Diameter  $r = 2(m-1)$

- Addressierung: Wurzel hat ID 1

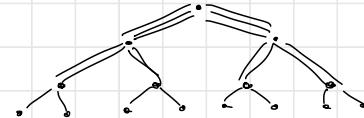
Knoten Ebene  $m$  hat  $m$  Bit ID

linkes Kind: hinzufügen 0 an Eltern-ID  
rechtes Kind: hinzufügen 1 an Eltern-ID



## Fat Tree

- Blockierungsproblem Richtung Wurzel  
↳ größere Komm. Kanäle Nähe Wurzel



## Kubus

$k$ -ärer  $n$ -Kubus

- $n = \text{Dim}$

- $k = \# \text{ Enden die Zyklen in einer Dim}$

- $N = k^n$  Knoten

- Addr. über  $n$ -stellige  $k$ -äre Zahl  $a_0 a_1 \dots a_{n-1}$

↳  $a_i$  entspr. Position in  $i$ -ter Dim ( $0 \leq i \leq n-1$ )

- Knotengrad  $2n$ , Diameter  $r = \log_2 N$

## Hyperkubus

- Knotennr. = Binärzahlen  $\rightarrow$  Nachbarn untersch. sich in 1 Stelle
- ↪ Routing über XOR

## dynamische Verbindungsstrukturen

für Anwendungen mit variablen, nicht regulären Komm. mustern

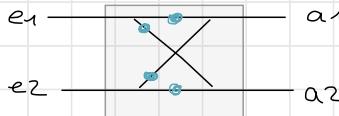
## Bus

- gemeinsam genutzt von allen angeschlossenen Prozessoren
- pro Zeitpunkt nur ein Datentransport
- Bandbreite =  $w \cdot f$ 
  - $w$  = # Leitungen
  - $f$  = Frequenz

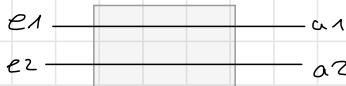
## Kreuzschienenverteiler

- vollst. vernetzt mit allen mögl. Permutationen der  $N$  Netzwerkeinheiten
- $N^2$  Schaltelemente  $\rightarrow$  je Kreuzung eines
- alle disjunkten Paare können gleichzeitig, blockierungsfrei komm.

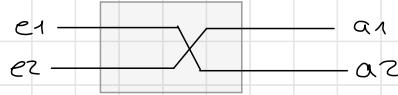
Schaltelemente: 2 Eingänge, 2 Ausgänge



• Kontakt, kann geöffnet/geschl. werden



durchschalten



vertauschen

## mehrstufige Verbindungsnetzwerke

- Kompromiss zw. niedriger Leistungsfähigkeit von Bussen und hohem HW Aufwand von Kreuzschienen

## Permutationsnetze

$p$  Eingänge können gleichzeitig auf  $p$  Ausgänge geschaltet werden  
↪ Permutation der Eingänge

einstufig: einzelne Spalte von 2er Schaltern

mehrstufig: mehrere Spalten

regular:  $k$  Stufen mit je  $p/2$  2er Schaltern,  $p = 2$ er Potenz

irregular: Lücken in regulärer Struktur

- Mischpermutation = Perfect Shuffle

$$M(a_n, a_{n-1}, \dots, a_2, a_1) = (a_{n-1}, \dots, a_2, a_1, a_n)$$

- Kreuzpermutation = Butterfly

$$K(a_n, a_{n-1}, \dots, a_2, a_1) = (a_1, a_{n-1}, \dots, a_2, a_n)$$

- Tauschpermutation = negation niedrigwertigen Bits

$$T(a_n, a_{n-1}, \dots, a_2, a_1) = (a_n, a_{n-1}, \dots, a_2, \bar{a}_1)$$

## Multiprozessoren

### Multiprozessorsysteme

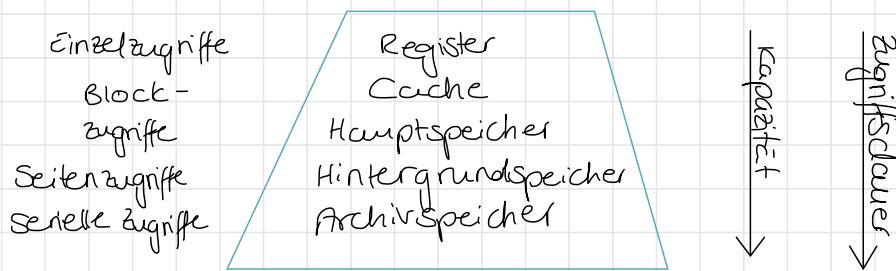
UMA = Uniform Memory Access

- Multiprozessor mit gemeinsamem Speicher
- zB SMP → jeder Prozessor gleichberechtigt Zugriff auf Betriebsmittel

NUMA = non uniform mem access

- Multiprozessor mit gemeinsamem verteiltem Speicher
- globaler Adressraum

### Speicherhierarchie



### Cache

- Pufferspeicher zw. Hauptspeicher, Prozessor
- stellt aktuelle HS Inhalte als Kopien prozessor zur Verfügung

### Aktualisierungsstrategien

- write through + no write alloc:
  - schreiben direkt in HS (bei write miss und write hit)
- write through + write alloc:
  - wie oben
  - zusätzlich bei write miss: Datum in Cache holen
- copy back:
  - write: setze dirty bit → wenn Datum aus Cache raus schreibe in HS zurück

## Cache Kohärenz

- Cache ist kohärent falls Lesezugriff immer Wert des zeitlich letzten Schreibzugriffs liefert

## Problem

- Zugriff auf veraltete Daten (stale data im Cache)

→ Lösung für Systeme mit Prozessor + DMA - Controller :

↑ ohne Cache

- (1) non-cachable data

↳ von Proz. + DMA gemeinsame Daten nicht cachebar

- (2) Cache - Clear / Cache - Flush:

DMA schreibt → sorge dafür dass Proz. Cache neu lädt

## Speichergekoppelte Multiprozessoren

- Kohärenz bestimmt welcher Wert bei Lesezugriff geliefert

- Konsistenz wann

## Kohärenzproblem:

- viele Caches können Kopien des gleichen Datums halten

## SW-Lösungen:

- Write - invalidate:  $P_i$  will in  $X$  schreiben

→  $P_i$  bekommt exklusiven Zugriff auf  $X$

→  $P_j$ ; Caches:  $X$  als ungültig erklären

- Write - update:  $X$  verändert → update  $X$  in allen Caches, spätestens beim nächsten Zugriff

## HW-Lösungen:

- Tabellen-basiert: Zustand der Blöcke in Tabelle festhalten

- Bus-Snooping: schnüffel auf Bus ob jemand ein Datum geändert das ich im Cache habe

## MESI - Protokoll

- jeder Cache : Snooping + Steuersignale
- Invalidate = invalidiert Einträge in Caches anderer Prozess.
- Shared = gibt es geladenen Block als Kopie in anderen Caches
- Retry = breche Laden des Datums in anderen Caches ab, schreibe, dann Laden neu starten
- jede Cache - Zeile 2 zusätzliche Statusbits

Zustandsgraph 1: durch lokale Lese-, Schreibzugriffe