Inf8

1. Kosten & Performance

SW-Technologie hat Einfluß auf ISA
.) Speicherbedarf/Programm 1
.) Assembler HLL (High Level Language)

Implementierung:

.) IC-Logik-Technologie

.) *Halbleiter DRAM*
Dichte 60% p.a ↑
Zykluszeit 33%, p.10a ↓ → Bandbreite /Chip ↑
Verbesserung Interface → Bandbreite/Chip ↑
Verbesserung höher als bei IC-Logik → Kosten/Bit ↓
DRAM-Größe * 4.p.3a

.) *Magnetplatten*
Dichte 50% p.a ↑
Zugriffszeit 33% p.10a ↓

KOSTEN
Herstellungskosten ↓ (Lernkurve!)
Maßzahl: Ertrag (Yield)
Lernkurve beeinflußt Ertrag (über Lebenszeit)
Kosten pro MB DRAM 40% p.a ↓

Preisreduktion
Faktor 8-10 über Lenbensdauer (konst. USD)
Herstellungsequipment?

Einfluß des Umfanges
.) Lernkurve steiler
.) Herstellungseffizient ↑
.) Umfang * 2 → Kosten 10 % ↓
.) Amortisierung der Entwicklungskosten ↓ → Kosten und Verkaufspreis konvergieren

Commodities = Products that are sold by multiple vendor in large volumes and are essentielly identical.

IC-Kosten: (Gesamtkosten exponential ↓)
Wafer → chopped into <u>dies</u> (+packaged)

Cost of IC = $$\frac{Cost\_of\_die + Cost\_of\_testing\_die + Cost\_of\_packaging}{Final\_test\_yield}$$

Cost of die= $\frac{Cost\_of\_Wafer}{Dies\_per\_Wafer * Die\_yield}$

Dies per Wafer = $\frac{\pi * (Waferdiameter/2)^2}{Die\_area} - \frac{\pi * wafer\_diameter}{(2 * Die\_area)^{\frac{1}{2}}}$

Defekte zufällig verteilt über Wafer.

Ertrag indirekt prop. Zu Komplexität des Herstellungsprozesses.

Die yield = $Wafer\_yield * (1 + \frac{Defects\_per\_unit\_area * Die\_area}{\alpha})^{-\alpha}$

Wafer yield: alle Wafer, die vollständig schlecht sind, kein Test notwendig!!!!!

ZB Defect per unit area 0.6 – 1.2 (95) Herstellungsfehler (abhängig von Prozeßreife)

α ..... Maß für Herstellungskomplexität (~Anz. Maskierungs-Niveaus)
z.B Multilevel Metall-CMOS: $\alpha \sim 3.0$

Kosten des Herstellungsprozesses bestimmen:

- Wafer cost
- Wafer yield α
- Defects per unit area

(Cost of die = f(Die area^4)

PERFORMANCE MESSUNG:

Maßzahlen → Benutzer: .) Response Time
.) Execution Time (ET)
EDV-Zentrum: Durchsatz (X)

Leistungsvergleich: $\frac{ET_y}{ET_X} = n = \frac{PERF_X}{PERF_Y}$ ($ET_i \sim 1/PERF_i$)

Total Cost of Ownership elements: (TOC)

- Hardware Acquisition
- Software Acquisition
- Installation
- Training
- Support
- Maintenance
- Infrastructure

Leistungsverbesserung:

ET ↓, X ↑

$CPU_{total} = CPU_{User} + CPU_{System}$

System Performance = ET on an unloaded system

CPU Performance = CPU_{User} on an unloaded system

Lastspezifikation ist wichtig!!!!!!!!!

(WL= Work load)

MESSPROGRAMME:

- Reale Programme z.B C-Compiler
- Kernels (aus echten Progr.) z.B: Livermore, Loops, Linpack
- Toy Benchmarks z.B Quicksort, Sieb des Eratosthenes – 10-100 LOC
- Synthetischer Benschmark zB Whatstone, Dhrystone, Lastnachbildung
- Benchmark Suie, zB SPEC (System Performance and Evaluation Cooperative)

Wichtig: **Wiederholbarkeit!!!**
Vollst. Systembeschreibung!!!!

Arithm. Mittel:

$$\frac{1}{n} * \sum_{i=1}^{n} T_i \qquad \sum_{i=1}^{n} g_i * T_i$$

Harmonisches Mittel: $R_i = f(1/T_i) * Rate \rightarrow$ indirekt proportional

$$\frac{n}{\sum_{i=1}^{n} * \frac{1}{R_i}} \qquad \frac{1}{\sum_{i=1}^{n} * \frac{g_i}{R_i}}$$

Ungewichtet gewichtet

Normierte Exekutionszeiten (bzgl. Referenzmaschine)
z.B. SPEC VAX 11/780

geom. Mittel

$$[\prod_{i=1}^{n} ET * ratio]^{\frac{1}{n}}$$ wie funktioniert die Formel???????????

Eigenschaft: $$\frac{Geom.Mean(x_i)}{Geom.Mean(Y_i)} = Geom.Mean(\frac{x_i}{Y_i})$$

Ergebnis konsistent (unabhängig von Referenzmaschine)
Hinweise: kein a. M. für Mitteilung nominierter ET
Keine Vorhersage mit g.M.!

AMDAHL's LAW:

Use faster mode of execution

$$ET_{new} = ET_{old}[(1- fe) + \frac{fe}{se}]$$

Fe=fraction enhanced (<=1)
Se=speed-up enhanced (>1)

$$S_{tot} = \frac{ET_{old}}{ET_{new}}$$ Fe:[1,se] se:[1,1/(1-fe)]

Law of diminishing returns!(verminderter(abnehmender) return)

Goal: Spend resources proportional to where time is spent.

CPU PERFORMANCE EQUATION

Time of a clock period
- -Length zB 2 nsec, or
- -Rate 500 MHZ

T_{CPU}= #CPU clock cycles for a program * clock cycle time
→ #CPU clock cycles for a program /clock rate

IC = instruction count = instruction path length

CPI = clock cycles per instruction (durchschnitt)
→ #CPU clock cycles for a program/IC

T_{CPU}= IC * CPI * clock cycle time
→ IC * CPI / clock rate

$$T_{CPU} = (\frac{Instruction}{\Pr ogram}) * (\frac{Clockcycles}{Instruction}) * (\frac{\sec}{clockcycle}) = \frac{\sec}{\Pr ogram}$$

Parameters: -clock cycle time (HW-Technologie + Organisation)
-clock cycles per instruction (Org. + ISA)
-instruction count (ISA + Compiler)

$$\#CPU\_clock\_cycles = \sum_{i=1}^{n} CPI_i * IC_i$$

$$T_{CPU} = [\sum_{i=1}^{n} CPI_i * IC_i] * clock\_cycle\_time$$

$$CPI = \frac{[\sum_{i=1}^{n} CPI_i * IC_i]}{IC} = \sum_{i=1}^{n} CPI_i * (IC_i / IC)$$

(IC_i/IC)=Pro Typ(zB addition, load,...) = %-Satz zB 20/80→1/4 Befehle sind zb additions-Befehle
n=Instruktionstypen

Achtung: CPI_i messen!!!! (Speicherhierarchieeinfluß!!!!)

Beachte:
CPI_i=Pipeline CPI_i+Memory System CPI_i

Programm-Verhalten:

Locality of reference.
-temporal (zeitlich)
-spatial(räumlich) zb Zugriff auf ein Element (zB Array,..)

Speicherhierarchie:

Axiom: **smaller is faster**
(shorter signal delays + more power per memory cell)

More memory bandwidth + lower access time

→ Cache (Blocks) (HW: CPU stalls)
→Virtual Memory (Pages) (SW: CPU does not stall)

Time for cache miss abhängig von
a) memory latency
b) memory bandwidth
(andere Arten von Cache: zb Internet-Cache (zb bereits übersetzte e-mail-Adressen als IP-Adressen, müssen von DNS nicht mehr abgefragt werden).

Cache Hit:
Wenn Info im Cache liegt und **nicht** vom Hauptspeicher geholt werden muß.

PERFORMANCES OF CACHES

-Verwendung von Amdahl's LAW
-Erweiterung der CPU Performance Equation.

T_{CPU}=(#CPU clock cycles + #Memory stall cycles) * clock cycle

Memory stall cycles
*=#Misses*Miss penalty*
*=IC * Misses per Instruction * Miss penalty*
=IC#Memory references per Instruction * Miss rate * Miss penalty*

(Miss rate = zwischen 0 und 1)

2. Instruction Set Architektur

Rechnerarchitektur:
Bestimmt durch Operationsprinzip für Hardware und Struktur ihres Aufbaus aus einzelnen Hardware-Betriebsmitteln.

Operationsprinzip:
Definiert das *funktionelle Verhalten* der *Architektur* durch *Festlegung* einer *Informationsstruktur* und einer *Kontrollstruktur*.

(Hardware-)Struktur:
Struktur einer *Rechnerarchitektur* ist *gegeben durch Art* und *Anzahl* der *Hardware-Betriebsmittel sowie* die *Regeln* für die *Kommunikation* und *Kooperation zwischen* den *Hardware-Betriebsmitteln*.

Informationsstruktur:
Wird durch die (semantischen) Typen der Informationskomponenten in der Maschine bestimmt, der Repräsentation dieser Informationskomponenten und der Menge der auf sie anwendbaren Operationen. Die Informationsstruktur lässt sich als eine Menge von abstrakten Datentypen spezifizieren.

Kontrollstruktur:
Wird durch Spezifikation der Algorithmen für die Interpretation und Transformation der Informationskomponenten der Maschine bestimmt.

Hardware – Betriebsmitteln
Hauptsächliche Hardware Betriebsmittel einer Rechnerarchitektur sind Prozessoren, Speicher, Verbindungseinrichtungen (Busse, Kanäle, Verbindungsnetzwerke) und Peripherigeräte.

Kommunikations und Kooperationsregeln
Werden gegeben durch die Protokolle, die den Informationsaustausch zwischen den Hardware-Betriebsmitteln regeln. Die Kooperationsregeln legen fest, wie die Hardware-Betriebsmittel zur Erfüllung einer gemeinsamen Aufgabe zusammenwirken (Beispiel: **Master-Slave-Prinzip**, kooperative Autonomie).

Benutzerschnittstelle einer Rechnerarchitektur:
Besteht aus der Sprache, in der Benutzer dem Betriebssystem Anweisungen erteilen kann, einer Maschinensprache und den für den Benutzer wichtigen Direktiven zur Benutzung der Anlage.

Genereller Aufbau SISD Rechner: (Folie UB 1)

Interner Speicher der CPU

Optionen:
- -Stack (implizit)
- -Akkumulator (explizit)
- -Register

Operanden EXPLIZIT oder IMPLIZIT!!!

Heute: general-purpose-register(**GRP) machines:**

Vorteil:
- -schneller
- -Compiler(optimierbar für GRP-Maschine)
- -Variablenspeicher

Unterscheidungen: -2. od. 3 Operanden
-#Speicher-Referenzen(0-3)

-Register-Memory-Architektur (zB Load R1,A; Add R1,B; Store C,R1)
-Load/Store bzw. Register/Register Architektur (Load R1,A;Load R2,B;Add R3,R1,R2; Store C,R3)
-Memory/Memory Architektur (Load A;Add B; Store C;) ???ob gleich mit Akk. F30

SISD (von Neumann-Rechner)

Operationsprinzip = sequentielle Programmabarbeitung (durch einen Prozessor)

d.h. zentrale Steuerung (Leitwerk/Steuerwerk)
+ zentrale Verarbeitung(Rechenwerk/Datenwerk)

Charakteristik (SISD):
Kontrollflussprinzip=Steuerung der Befehlsauswahl durch Befehl selbst
Verwendung von Variablen-Speicherzelle-Zuweisungsprinzip

Unterklassen:

Princeton Architektur (J.v. Neumann)
- **gemeinsamer Speicher und gemeinsame Speicher/Prozessor-verbindung** für *Operanden* und *Instruktionen*

Harvard Architektur:
- **Getrennter Speicher** und Verbindung *für Operanden und Instruktionen*

SISD Klassifikation nach:

- **Anzahl separater Datenpfade** für Instruktionen/Operanden
  (Princeton/Harvard)

- **Ort der Operanden**
  Speicher-Speicher-Maschinen
  Virtuelle-Register-Maschinen
  Register-Register-Maschinen oder Load/Store-Architektur

- **Instruktionssatzarchitekturen**

GENERAL-PURPOSE-REGISTER (GPR)-MASCHINES

-after 1980 virtually every machine

Vorteile:
- Register schneller als Speicher
- Register für einen Compiler leichter zu handeln, kann effizienter verwendet werden
  Register kann verwendet werden, um Variablen zu behalten
- (reduziert Speicher-Traffic → Speed up)

-Anzahl von Registern?

-2 Haupt-Instruktionen set characteristion:
a) Anzahl von Operanden in einer ALU-Instruktion (2 oder 3)
b) Anzahl von Speicher-Operanden in typischen ALU-Instruktion (keine von 3)

-7 mögliche Kombinationen von Attributen

Arten von internen Speichern:
a) stack
b) accumulator
c) set or registers

Operanden werden genannt:
-implizit(Stack-Architektur)
-explizit/implizit(Accumulator Architektur)
-explizit(GPR Architektur) .) Registers, or Memory locations

explizite Operanden:
-Zugriff direct auf Memory
-muß zuerst in temporären Speicher geladen werden
(abhängig von Klasse der Instruktion und Wahl der spezifischen Instruktion)

2 Arten von Register-Maschinen:

- Register-register architecture
  (kann auf Speicher als Teil eines Befehls zugreifen)

- Load/store or register-register architecture
  (kann auf Speicher nur mit load/store Befehlen zugreifen)

- Memory-memory architecture
  (speichert alle Operanden im Speicher)

BESCHLEUNIGUNG DURCH PARALLELITÄT

Unterscheide:

a) Phasenparalellität (Pipilining Konzept, Mit einer Verzögerung werden die Befehle gestartet)

b) Nebenläufigkeit (n-Befehle werden **gleichzeitig** gestartet)

BEARBEITUNGSZEIT:

n Instruktionen → Zeit T(n) → Durchführungszeit
Anm. keine Datentransport-/Kommunikationszeiten

Seriell:

$T_s(n)=n*t_s$
t_s=mittlere Bearbeitungszeit pro Instruktion

Phasenparallel:

$T_p(n)=[k+(n-1)]*t_p$
t_p= Zeit je Phase
k= # Stufen (Phasen) (k>6 → Superpipeline)

Nebenläufig:

$$T_N(n) = [\frac{n+p-1}{p}]*t_N$$

p=#Verarbeitungseinheiten (zB Bild 5.1 p=3)
t_N=Zeit in einer Verarbeitungseinheit

i.d.R: $t_N >= t_s$

$$t_p >= \frac{t_s}{k}$$

Speedup – phasenparallel:

$S_p(n) = \frac{T_S(n)}{T_p(n)}$ $t_s=k*t_p$ (= Phase * Befehl)

$S_p=(n) = \frac{n*k}{k+n-1}$

$n \to \infty \quad S_p(n) = k \rightarrow \frac{k}{1+\frac{k-1}{W}} = k \longrightarrow 0$

Effizienz (Auslastung):

$E_p(n)= \frac{S_p(n)}{n} = \frac{n}{k+n-1}$

wenn $n \to \infty \rightarrow E_p(n)=1$ (wenn unendlich viele Prozessoren, dann gilt dies immer)

Andere Sichtweise:

t_o=Vorbereitungszeit
t_e=Zeiverbrauch pro Instruktion nach der Vorbereitung

seriell: $t_e=t_s$, $t_o=0$

phasenparallel: $t_o=(k-1)*t_p$
$t_e=t_p$

$T(n) = t_o+n*t_e=t_e*[\frac{t_o}{t_e+n}] = r_\infty^{-1} \ [n_{\frac{1}{2}}+n]$

$r_\infty = \frac{1}{t_e}$ ..... Maximalrate (abh. von Rechnertechnologie) soll mögl. groß sein

$n_{\frac{1}{2}} = \frac{t_o}{t_e}$ .... Halbwert (abh. von Architektur) soll: mögl. klein sein

Durchsatz (Wieviel Instruktionen/Zeiteinheit werden durchgelassen)

$r(n) = \frac{n}{T(n)} = r_\infty * \frac{n}{n_{\frac{1}{2}}+n}$

$r(n_{\frac{1}{2}}) = \frac{r_\infty}{2}$

Pipeline: $n_{1/2}$ =k-1

PIPELINING
.) Überlappung der Ausführung von Aufgaben
.) Aufgabe als Folge von Schritten
(Durchsatz durch Pipelining erhöht)

Machine Cycle

.) perfekt ausbalanziert: (maximale Speed(Optimum)

$$\frac{Zeit}{Instrutkionen(pipelined} = \frac{\frac{Zeit}{Instruktionen(unpipelined)}}{\#Stufen}$$

Abweichung durch:
- **Unbalanziertheit**
- **Overhead** (zb durch Zwischenspeichern, Weiterleitung,...)

Pipelining führt zur Redutkion von:
- **CPI** oder
- **clock cycle time**

Beachte:
- **Durchsatz erhöht**, aber
- **Exektuionszeit** der **einzelnen Instruktionen** nicht kürzer (**eher länger**: **Overhead** durch **Pipeline Register** Delay+clock skew)

PIPELINE HAZARDS:

Drei Klassen:

a) Struktur (Ressourcenkonflikt, Instruktionen greifen auf gleiche Variable)
b) Daten (Datenabhängigkeiten, Instruktion wartet auf Ergebnis von Instruktion x)
c) Kontrolle (Änderung des Programmzählers)

Konsequenz:
stall the pipeline (NOP (=No operation) einfügen)
→
- **Fortsetzung** aller **früheren Instruktionen**.
- **Anhalten** aller **späteren Instruktionen**

PIPELINE PERFORMANCE:

$$S_{pipe} = \frac{avg.Instruction\_time\_unpipelined}{avg.Instruction\_time\_pipelined}$$

$$\rightarrow \frac{CPI\_unpipelined}{CPI\,pipelined} * \frac{clock\_cycle\_unpipelined}{clock\_cycle\_pipelined}$$

(cycle per Instruction)

VERWENDUNG VON CPI zum Vergleich:

CPI pipelined = Ideal CPI + Pipeline stall clock cycles per Instruction

⇨ 1 + Pipeline stall clock cycles per Instruction.

Anm:
- **perfekt balanzierte Pipeline**
- cycle time **overhead der Pipeline = 0**

→ equal cycle time

$$S_{pipe} = \frac{CPI\_unpipelined}{1 + pipeline\_stall\_clock\_cycles\_per\_Instruction}$$

Einfacher Fall:
alle Instruktionen brauchen gleiche Anzahl von Zyklen (=#Stufen)

$$S_{pipe} = \frac{Pipeline\_depth}{1 + pipeline\_stall\_clock\_cycles\_per\_Instruction}$$

VERBESSERUNG DER CLOCK CYCLE TIME

$CPI_{pipelined} = CPI_{unpipelined} = 1$

$$S_{pipe} = \frac{1}{1 + pipeline\_stall\_clock\_cycles\_per\_Instruction} * \frac{clock\_cycle\_unpipelined}{clock\_cycle\_pipelined}$$

Anm:
- **perfekt balanzierte Pipeline**
- cycle time **overhead der Pipeline = 0**

$$clock\_cycle\_pipelined = \frac{clock\_cycle\_unpipelined}{pipeline\_depth}$$

$$S_{pipe} = \frac{Pipeline\_depth}{1 + pipeline\_stall\_clock\_cycles\_per\_Instruction}$$

SUPERSKALARE PROZESSOREN
(Erhöhte Geschwindigkeit, jedoch Out-of-Order-Ausführung)

= Multi-Unit-Prozessoren
besitzen mehrere nebenläufig nutzbare Funktionseinheiten
Einheiten i.a. Pipelines mit gemeinsamen/getrennten Registerfiles

CPI<1 möglich

ILP=inctruction level parallelism

es entstehen verstärkt Abhängigkeiten zw. den Befehlen sinnvoll, wenn Fus unterschiedl. viele Zyklen benötigen. (**führt zu Out-of-Order Ausführung**)

Bsp: DEC-Alpha

weiters: VLIW-Prozessoren (Very Long Instruction Word)
(nebenläufig ausführbare Operationen für die **einzelnen Fus** des **Prozessors** zu **einem einzigen Instruktionswort zusammengefasst**,
Compiler ist dafür **zuständig**)

3. SPEICHERHIERARCHIE

- Lokalitätsprinzip
  - o räumlich
  - o zeitlich

- Kosten/Leistung

(Kleinere Hardware ist schneller)

Mehrere Niveaus – Schicht (i-1) in Schicht i enthalten!
(zb Info von Cache auch in Hauptspeicher enthalten)

KLASSIFIZIERUNG VON SPEICHERN

- nach der Menge der zu speichernden **Zugriffsgeschwindigkeit**
  -) schneller Datenspeicher
  -) Massenspeicher
- nach der **Zugriffsform** (Art und Weise des Zugriffs auf Speicherinhalt)
  -) *ortsadressiert:*
    -) wahlfreier Zugriff (RAM)
    -) sequentieller Zugriff (FIFO,LIFO, Magnetband)
    -) Indexsequentieller Zugriff (Diskette, Plattenlaufwerk)
  -) *inhaltsbezogen*
    assoziativ: Suchen aufgrund der Beschreibung des Objektes zb nach einer Adresse)

- nach dem zugrundeliegenden **physikalischen Effekt**
  - -) *Halbleiterspeicher* (schnelle Datenspeicher)
  - -) *magnetische Speicher* (Massenspeicher)
  - -) *optische Speicher* (Massenspeicher)
  - -) *mechanische Speicher* (historisch: Lochkarten, Lochstreifen)

KENNGRÖßEN VON SPEICHERN

- Speicherkapazität
  Anzahl der Speicherplätze in Bit oder Byte
- Zugriffszeit
  mittlere *erforderliche Zeit* zum Schreiben und/oder Lesen einer Speicherzelle
- Preis pro Speicherzelle/Bit
- Zykluszeit
  kürzeste Zeit, nach der ein Lese- oder Schreibvorgang wiederholt werden kann
- Art der Adressierung
  orts- oder inhaltsbezogen
- Art des Zugriffs
  wahlfrei oder sequentiell
- Größe der adressierbaren Einheit
- Änderbarkeit des Speicherinhaltes
  read/write oder read only
- Energieverbrauch des Speichers
  bleibt der Speicherinhalt ohne Energiezufuhr erhalten?

!!!!Bedeutung der Hierarchie wächst mit Prozessorfortschritt!!!!

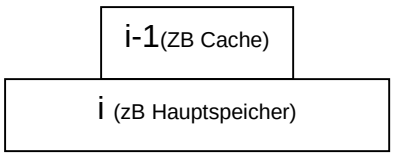
EFFEKTIVITÄT DER SPEICHERHIERARCHIE

Memory stall cycles = IC*Memory_references_per_Instr. *Miss_rate*Miss_penalty

(Miss penalty= additional time to serve the miss)

MITTLERE ZUGRIFFSZEIT EINER SPEICHERHIERARCHIE

Effektive Zugriffszeiten



W!=Wahrscheinlichkeit

Annahme: Daten der Schicht i-1 auch in Schicht i vorhanden(Daten von Cache auch in HSP)
H(i) ....W! Daten in Schicht i anzutreffen (=Präsenzwahrscheinlichkeit)
h_i ....W! Daten in Schicht i, und nicht in Schicht i-1 (=relative Trefferhäufigkeit)
(Daten in HSP jedoch nicht in Cache)

$H(i)=P\{\text{Daten in } i \wedge i\text{-}1\} + P\{\text{Daten in } i \wedge \neg i\text{-}1\}$
$= P\{\text{Daten in } i \mid \text{Daten in } i\text{-}1\} * P\{\text{Daten in } i\text{-}1\} + h_i$
$= H(i\text{-}1)+h_i$

$h_i=H(i)-H(i-1)$ $i>1$ H(0)=0, H(n)=1 n….#Stufen

Effektive mittlere Zugriffszeit

$E(T(n)) = \sum_{i=1}^{n} h_i * T_i$ T_i...effektivfe Zugriffszeit auf i → $= \sum_{k=1}^{i} t_k$

t_k...Zugriffszeit auf k

$$E(T(n)) = \sum_{i=1}^{n} * \sum_{k=1}^{i} t_k * [H(i) - H(i-1)] = \sum_{i=1}^{n} [1 - H(i-1) * t_i$$

Bsp:

n=2:

$E(T(2)=t_1+[1-H(1)]*t_2 = t_1+[1-h_1]*t_2 = t_1*[r_1+(1-r_1)*h_1]$

Zugriffsverhältnis $r_i = \frac{t_i + t_{i+1}}{t_i}$

z.B. t_1=1, t_2=9 bzw. 4 r_1=10 bzw. 5

Cache	HSP	1.9	1.4	← h_1=0.9 (Annahme)
		3.7	2.2	← h_1=0.7 (Annahme)

(h_1=Trefferwahrscheinlichkeit, ob etwas im Cache ist)

(Siehe Bild Zugriffsverhältnis 6.2)

GEMEINSAMES PRINZIP – VIER FRAGEN

a) BLOCK PLACEMENT (Abbildungsproblem)
   Where to place?
b) BLOCK IDENTIFICATION (Identifikationsproblem)
   How to find?
c) BLOCK REPLACEMENT (Ersetzungsproblem)
   Which to replace?
d) WRITE STRATEGEY (Datenkonsistenzproblem)
   What happens?

Cache:
used as term whenever buffering is employed to reuse commonly occurring items

schneller Pufferspeicher: Am CPU-Chip, oder / zwischen CPU und Systembus

Cachezeilen= Blockrahmen eines Caches

Cacheintrag = Adress-Tag(Etikett)
+ Block(Daten) (Cachezeilen mit einem Blockzugriff gefüllt!!)

Unterscheidungs-Aspekte:

- **dezitierter** oder **gemeinsamer Daten**- und **Instruktionscache** (Bild UB 10)
- **transparente** oder **nichttransparente Integration** des **Cache**
  ->keine Möglichkeit für Programmierer Cache-Aktionen zu steuern
- **virtueller** oder **physikalischer Cache**
  ->arbeitet mit virtuellen, vom Prozessor erzeugten Adressen

virtuelle (für **I**-Cache): -) schneller (gleichzeitiger) Zugriff
(**I**nstruktion,Bild 6.7a) -) *keine eindeutige Zuordnung virt. und phys. Adresse*
(bei F.68, Bild UB10) (adress aliasing)
(MMU=Memory Management Unit ist auf gleicher Ebene wie Cache, bei gehen in HSP)

phys.(für **D**-Cache): -) für Snooping
(**D**ata,Bild 6.7b) →keine Snooping bei Instruktionen
(MMU ist zwischen P und Cache, d.h. beides geht über Cache in HSP)

CACHE – ORGANISATION

Basic Question:

1) Where to place a block
   - Direkt Mapped(nur ein Feld)
     (block adress) MOD (#blocks in cache)
   - Fully associative
     anywhere
   - Set assoziative (ein Set(Gruppe) hat mehrere Felder)
     set=group of blocks
     (block address) MOD (#sets in cache)
     n blocks in a set = n-way set assoziative
     (Siehe Bild Ub 10.1, bei Folie 69)

2) How is a block in a cache?
   - address tag on each block frame that gives the block address
     (in parallel) valid bit

3) Which block should be replaced on a miss?
   - random
   - least-recently-used(LRU) (Referenz am weitesten in der Vergangenheit)
   - FIFO
   - LFU(least frequently used) (Zeile, die am wenigsten oft angesprochen wurde)

ad 1 ABBILDUNGSPROBLEM
(Siehe ad2 Indentifikationsproblem Bild UB 10.2(vor Folie 71))

1.1 **Direkte Abbildung**

HSP in Abschnitte mit jeweils N Blöcken eingeteilt.
(N=#Cachezeilen)
1. Zeile jedes Abschnittes=1. Cachezeile, usf.

1.2 **Assoziative Abbildung**

stärkere Berücksichtigung zeitlicher Referenzlokalität

Voll-assoziativ:

ein HSP Block kann in beliebige Cache-Zeile kommen
Tag-Speicher ist Assoziativ-Speicher (Tag=Schlüssel)

X-Wege assoziativ (x-way set-associative)

ein HSP-Block kann in eine von x Cache-Zeilen kommen
x Cache Zeilen bilden ein Set, es gibt N/x Sets

weitere Verwaltung im Tag-Speicher
z.B **VALID-Bit** (V-Bit)
zeigt an, ob Inhalt der Cache Zeile gültig ist
DIRTY-Bit (D-Bit)
zeigt an, ob auf diese Cache-Zeile seit der letzten Ersetzung(Einlagerung von RAM in Cache) schreibend zugegriffen wurde.

4) What happens on a write?

basic options: (Schreiben → Treffer!!!!)

- write through (=store through) → CPU write stall
- write back (=copy back or store in) **dirty bit**

options on a write miss: (Schreiben → Fehler!!!)

- write allocate (=fetch on write, betroffener Block kommt in Cache+Schreibtrefferoperation)
- no-write allocate (=write around, kein Kopieren in Cache)

CACHE PERFORMANCE

measure of memory-hierarchy performance:
avg. memory acces time= Hit time + miss rate * miss penalty
(hit time= time to hit the cache)

→ *cache optimisations*

*CPU time = (CPU exec. clock cycles + memory stall clock cycles)*clock cycle time*

Q: clock cycles for a cache hit is part of
-) CPU exec clock cycles, or
-) memory stall clock cycles

memory stall clock cylces=

reads*read_miss_rate*read_miss_penalty
+ writes*write_miss_rate*write_miss_penalty

simplified: (combine reads & writes → avg. miss rate + miss penalty for reads&writes)

memory stall clock cycles = memory accesses * missrate*miss penalty

factoring:

$$CPU_{time} = IC * (CPI_{exec} + \frac{memory\_accesses}{instructions} * miss\_rate * miss\_penalty) * clock\_cycle\_time$$

$$\frac{Misses}{Instruction} = \frac{memory\_accesses * miss\_rate}{Instructions}$$ (*architecture dependent!!!*)

with a single computer family:

$$CPU_{time} = IC * (CPI_{exec} + \frac{memory\_stall\_clock\_cycles}{instructions}) * clock\_cylce\_time$$

(Erweiterbar um Fehlerrate)

Bemerkung:

- the lowser the CPI_{exec} the higher the relative impact of a fixed number of cache miss clock cycles
- when calculating CPI, the cache miss penalty is measured in CPU clock cycles for a miss
- CPU with higher clock rate has a large number of clock cycles per miss
  → higher memory portion of CPI

T= (path length) * CPI * (seconds per cycle)

Zykluszeit:
(Kehrwert=clock speed)

cycles per instruction:
festgelegt durch :
system architecture & quality of compilation

=dynamic instruction count:
festgelegt durch:
instruction set architecture (ISA)
application characteristics
operating system interfaces
organization

Bei zunehmender Taktfrequenz steigt die Kurve an, d.h. Fehlerrate steigt (Siehe Bild UB 11)

Folie 77??????????????????????????

IMPROVING CACHE PERFORMANCE

REDUCING:

A) THE **MISS RATE**
B) THE **MISS PENALTY**
c) THE **TIME** TO **HIT** IN THE **CACHE**

(goal: make the whole system fast!!!!)(Balanced perspective)

ad A) (miss rate)
- Larger Block Size
- Higher Associativity (wie?????????)
- Victime Caches (wie funktionierts genau???)
- Pseudo Associative Caches
- HW-prefeching (Instr./Data)
- Compiler Controled prefeching ????????
- Compiler optimisations

Ad B) (Miss penalty)
- Give Priority to Read Misses over Writes
- Sub-Block Placement for Reduced Miss Penalty
- Early Restart and Critical Word First
- Nonblocking Caches to Reduce Stalls on Cache Misses
- Second Level Caches (mehr oder weniger davon?????)

Ad C) (Time to hit Cache)
- Small and Simple Caches
- Avoiding Adress Translation Doing Indexing of the Cache
- Pipelining Writes for Fast Write Hits

nicht prüfungsrelevant
nur übersicht (zum Verständnis)
wichtig sind nur die 3 Begriffe:
.)miss rate,
.)miss penalty und
.)time to hit the cache

ad 4 **DATENKONSISTENZPROBELM:**

<u>konsistenz</u>, wenn sich Kopien <u>nicht</u> unterscheiden

bei *write-through*: <u>immer</u> konsistent
write-back: nur auf Veranlassung konsistent } Monoprozessoren

MEHRPROZESSORSYSTEME: CACHE-KOHÄRENZPROBLEM!!!!!!!!!!

<u>Lösungsmöglichkeiten:</u>

(4.1): *gemeinsame Daten als nicht cache-geeignet kennzeichnen*
→ **nicht transparent**, reduziert Cache-Trefferrate

(4.2): *Zugriff* auf *gemeinsame Daten* **nur** in *kritischem Abschnitt zugelassen*
→ **Serialisieren** der Zugriffe

(4.3): *Cache-Kohärenz-Protokolle*
Jeder Prozessor hat **SNOOPING LOGIC**(Schnüffellogik)
(überwacht Adressverkehr auf gem. Speicherbus,
bei Schreibzugriff Vgl. mit eigenem Cache-Tag-Speicher,
falls ja→invalid setzen oder aktualisieren)
(Tag-Speicher mit 2 Leseports)

MAIN MEMORY:

Technologie:
- **Access Time** = time between when a read requested an when the desired word arrives

- **Cycle Time** =minimum time between requests to memory (>access time)

DRAM vs. SRAM

DRAM(dynamisch, muß refreshed werden, "refreshing des inhalts", z.B. HSP)
SRAM(static, sehr schnell, aber teuer, z.B. Cache)

IMPROVING MAIN MEMORY PERFORMANCE

Techniken zur Erhöhung der Bandbreite:
- Wider Main Memory
- Simple Interleaved Memory
- Independent Memory Banks (unabhängige Speicher-Bänke)
  (Bild Ub 12)
- Avoiding Memory Bank Conflicts
- DRAM-specific Interleaving

PERFORMANCE – INTERLEAVED MEMORY

R ZV {1,2,….,M}

p(k) =p(R=k)

$E[R] = \sum_{k=1}^{m} k * p(k)$ Erwarteter Verschränkungsgrad (optimalerVerschr.grad(Lokalitätsprinzip))

λ...W!, dass **nach** einem Zugriff auf eine Bank ein Konflikt auftritt

Anm: **Zugriffe statisch unabhängig!!!** R geom. verteilt

$p(k) = (1-\lambda)^{k-1} * \lambda$ k=1,2,...m-1
$p(m) = (1-\lambda)^{m-1} * 1$

$$E[R] = \sum k * (1-\lambda)^{k-1} * \lambda + m(1-\lambda)^{m-1} = \frac{1-(1-\lambda)^m}{\lambda}$$ (Bild UB 13)

VIRTUELLER SPEICHER (Folie 85)

Effektiver Speicherraum > physikalischer Adressraum
Voller Adressraum für jeden Prozeß → teuer bei mehreren Prozessen
→VIRTUALISIERUNG DES SPEICHERS
→Unterteilung des Speichers in BLÖCKE
→Schutzmechanismen notwendig (Base<=Adress<=Bound)

BLÖCKE = Seiten (vorgegebene Größe) alle gleich groß (Kacheln)
= SEGMENTE (variable Größe –abhängig von Programmstruktur, Code-/Datenbereich)→ Verwaltungsaufwand

ZWEI HIERARCHIEN: HSP + HINTERGRUNDSPEICHER(Plattenspeicher)

Page/Adress Fault!!

RELOCATION vereinfacht

MEMORY MAPPING = ADRESS TRANSLATION
virtuelle Adresse → physikalische Adresse

PAGING vs. SEGMENTATION (abh. von Prog.Struktur)

	Page	Segment
Word per adress	One	Two (segment and offset)
Programmer visible	Invisible to appl.programmer	May be visible to appl.programmer
replacing a block	trivial (all blocks are the same size)	hard(must find contiguous, variable-size, unused portion of main-memory
memory-size inefficiency	internal fragmentation (unused portion of page)	external fragmentation (unused pieces of main memory)
efficient disk traffic	yes (adjust page size to balance access time and transfer time)	Not always (small segments may transfer just a few bytes)

UNTERSCHIEDE CACHE – VIRTUAL MEMORY

- Ersetzung
  Cache: Hardware
  V.M: Operation System

- Größe
  Cache: **unabh.** von Adreßgröße des Prozessors
  V.M: **abhängig** von Adreßgröße des Prozessors

-Hintergrundspeicher:
  V.M: zusätzlich für Filesystem

Vier FRAGEN:

Q1: **WO PLAZIEREN**: überall

Q2: **WIE FINDEN**: Adressübersetzung
PAGE Table (Größe = #virt. Seiten)
INVERTED PAGE TABLE (Größe=#Seitenrahmen, Hashfkt.)
Unterstützung durch translation look-aside buffer (TLB-Cache)
(Bild 6.26, 6.28, nach Folie 88)
Q3: **WAS ERSETZEN**: z.B: LRU (last recently used)(gleiches möglich wie bei cache)
Unterstützung durch Use-/Reference-Bit
Q4: **WAS PASSIERT BEI SCHREIBEN**?

Weitere Fragen:

a) **WIE GROSS SOLL EINE SEITE SEIN**?
größere Seiten:
Vorteil:
- größere Trefferrate pro Seite → weniger Einlagerungen nötig (räumliche Referenzlokalität)
- kleinere Pagetable

Nachteil:
- weniger Seiten im HSP (zeitliche Referenzlokalität schlechter)
- größere interne Fragmentierung → schlechtere Ausnutzung (restliche Seite, die sich nicht mehr im Speicher ausgeht, Folie 89)

N.....Programmlänge (Bytes)
n.....#Seiten, die vom Programm belegt werden
Z.....Seitengröße
c.....Größe des Seitendeskriptors(Bytes)
(Referenzierung auf Page)

interne Fragmentierung

Speicherverschnitt: $V=\overbrace{(n*Z-N)}+\underbrace{(c*n)}$

IPT....interne Patch-Tabelle

50%-Regel: n*Z=N+1/2 * Z

$$V=(c+Z)*[\frac{N}{Z}+\frac{1}{2}]-N$$

$$[\frac{dV}{DZ}]Z_{opt}=0 \rightarrow Z_{opt}=(2*c*N)^{\frac{1}{2}}$$

Speicherausnutzung: $U=\frac{N}{V+N}$

SEKUNDÄRSPEICHER:
I/O-System → Performance nicht nur CPU! (Amdahl)

Massenspeichersystem → Übersicht Bild UB 14+15

Plattenspeicher:

Zylinder: alle Spuren auf ein einzelnen Platten mit gleichen Abstand zum Zentrum

Anzahl:
- Umdrehungen
- Spuren (Tracks)
- Sektoren → variable vs. konstante Bitdichte

Seek(time) – Positionierung (Sepzifikation: min, max, avg.)
avg. seek time(ms) –(un)abhängig von Zugriffsmuster

∅ Rotation latency = rotational delay (=1/2 Umdrehungszeit) (=Rotationsverzögerung)

zwischen Diskcontroller und HSP: Hierarchie von Controllern + Datenpfaden

Flächendichte = (tracks/inch) on disk surface * (bits/inch) on track

~60%p.a.↑
Kosten/MB ↓

(Trend geht zu kleineren Platten, BxHxT)

SPEICHERDICHTE:

Increased recording density (~60% p.a)

increase in:
- **bits per inch (bpi)**
- **tracks per inch (tpi)**

DISK PERFORMANCE:

components:
- command Overhead(Register laden, initialisieren,…)
- Seek Time = a * (distance –1)^{0,5} + b * (distance-1) + c ~ 8-10 msec
- ∅ Rotation Latency → 0,5/RPM
- Transfer Time (eigentliche Auslese/Transferzeit, TZ= Umfang/Speed)
- Controller Time (+ Queueing delay(lastabhängig))

command Overhead depends on:
- type of drive interface (IDE or SCSI)
- type of operation (read/write)
- wether command can be satisfied from the disk drive's buffer or cache memory (buffer hit/miss)
  Examples: 0,5 ms buffer miss
  0,1 ms buffer hit

Data Transfer Time depends on:
- data rate
- transfer size

Data rate:
- media data rate (to / from recording media)
  → RPM, recording density
  e.g. 5400 RPM, 111 vectors (512 Bytes each) → 5 MBps

- interface data rate (host to disk drive)
  e.g. SCSI3: 20 MBps (8-bit drive)
  IDE Ultra-ATA-Interface: 33,3 MBps
  Common: 10 MBps

"Alternativen":
- solid state disks (SSD)
- expanded storage (ES)
- optische Disks
- "Magnetbänder"
- DRAMs + Batterie
- gr. Speicher mit Blocktransfer

RELIABILITY vs. AVAILABILITY:
reliability(=Zuverläßigkeit) – Is anything broken?
availability (=Verfügbarkeit) – Is the System still available to the user?

Wie verbesserbar?

avail: mehr Hardware

reliability:
- bessre Umgebungsbedingungen
- zuverlässigere Komponenten
- weniger Komponenten

Verfügbarkeit:
parallel:
$A_0 = 1-(1-A_1)*(1-A_2)$ $(1-A_i)$=nicht Verfügbarkeit

seriell:
$A_S = A_1 * A_2$
(Folie 101)

SPIEGELPLATTE (mirroring/shadowing)
beide beschrieben, alternierend gelesen

DISK-ARRAY
Speicherbandbreite erhöhren? ???????????

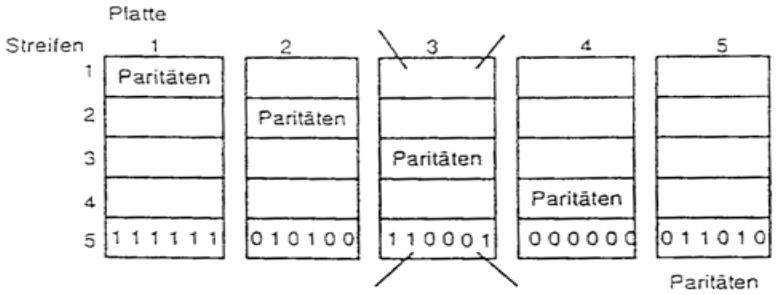
mehrere kleine Plattenspeicher statt eines großen →Durchsatz verbessert (nicht notwendigerweise auch Latenzzeit)

paralleler Zugriff
Speicherung **„streifenweise“** verschränkt.(*Daten über mehrere Platten verteilt*)

Zuverlässigkeit nimmt mit Anzahl der Laufwerke ab → Hinzufügen redundanter Laufwerke (erhöht Verfügbarkeit) →
RAID = redundant array of independent/inexpensive disks

RAID5:
Verteiltest Abspeichern der Paritätbits und Einteilen der Disks in Gruppen: jede Gruppe wird durch Paritätsbits geschützt. Auf die Laufwerke kann unabhängig zugegriffen werden.

Bsp:



Disk Array mit 4+1 Laufwerken
Datum mit 4 Bits (einzeln verteilt) + Paritätsbit

6 Dateneinheiten/Streifen → eine der Platten enthält die 6 Paritätsbits (Sum mod 2)

Raid-Level		Failure Survived
0	Nonredundant	0
1	Mirrored	1
2	Memory-style ECC	1
3	Bit-interleaved parity	1
4	Block-interleaved parity	1
5	Block-interleaved distributed parity	1
6	P+Q redundancy	2

4. VERBINDUNG VON EINHEITEN

Verschiedene Subsysteme → **INTERFACE**

=Übergang an der Grenze zwischen Funktionseinheiten mit vereinbarten Regeln für die Übergabe von Daten und Signalen.

BUS = shared communication link (between subsystems)
("Ansammlung von Leitungen")

VT der Busorganisation:
-) niedrige Kosten
-) Vielseitigkeit

NT:
-) möglicher Engpaß

Bus besteht aus mehreren Komponenten mit unterschiedlichen Funktionen:
z.B. für -) Daten und Adresstransfer
-) Übertragung von Steuersignalen
-) Bereitstellung von Versorgungsgrößen (Takt, Spannung)

Max **Busgeschwindigkeit** abhängig von physikalischen Größen:
- phys. Grenzen
- Länge
- #Geräte (→ Last)

Busklassifikation:

1) CPU MEMORY-BUS
-) kurz
-) hohe Geschwindigkeit
-) angepasst an Speichersystem

2) I/O BUS
-) lang
-) verschiedene Geräte
-) hohe Varianz der Bandbreite
(der angeschlossenen Geräte, zB Tastatur)
-) Standards (PCI,SCSI,...)

Bustransaktion:
Zwei Teile: -) Adresse senden
-) Daten senden/empfangen

Operationen bezüglich Speicher:
-) Lesen (von Speicher)
-) Schreiben (in Speicher)

BUSZYKLUS:
1) Busanforderung/-vergabe
2) Übertragung Adresse/Daten
3) Busfreigabe

Busprotokoll: spezifiziert die einzelnen Schritte im Buszyklus
Busmaster: kann eine Transaktion initiieren!

BUSOPTIONEN:

mehrere Master → ARBITRATION scheme (Wer bekommt bus, bei gleichzeitiger Anfroderung)
zB fixe Priorität
zufällige Auswahl→Fairness!! (darauf achten, dass jede Komponente dran kommt).

split transaction (=connect/disconnect, pipelined bus packet-switched bus)
(ähnlich wie bei paketorientiert) → higher bandwidth, higher latency!!

synchron vs. asynchron:

Synchron: -) clock (control lines)
(getaktet) -) fixes Protokoll für Adressen/Daten bezüglich Uhr

Vorteil:	Nachteil:
-) billig	-) alles muß mit gleicher Uhrrate laufen
-) schnell	-) Länge beschränkt (clock skew=Taktverzerrung)

Asynchron: -) self-timed (ereignisgesteuert)
-) handshaking-protocol (Austausch spez. Signale zw. Teilnehmern
→ Überwachung mittels Timeout)

Vorteil: -) versch. Geräte
-) Länge
-) skaliert besser mit technolog. Anwendungen

SYSTEMBUS = verbindet ganze Rechnereinheiten

BUSMERKMALE
-) Datenbreite
-) Transfergröße (Wort/Block)
-) #Master
-) Split transaction (ja/nein)
-) rechnerspezifisch/rechnerunabhängig
-) Funktion (Adressen, Daten, Befehle, Steuerinfo)
-) uni-/multifunktional(=gemultiplext)
-) synchron/asynchron
-) gemeinsamer / dezitierter Bus
(zB Adressen+Daten) (für best. Komponenten)
-) seriell/parallel

Wann ist was besser?
Synchron dann besser, wenn Distanz kurz ist und die I/O-Geräte am Bus die gleiche Geschwindigkeit(Traktrate) verwenden.

BUSZUTEILUNG:

Problem: mehrere Master fordern den Bus an, nur einer erhält die Zuteilung

Ziel: Auflösung von Konflikten bei der Busanforderung
faire Vergabe
Minimierung von Leerzeilen

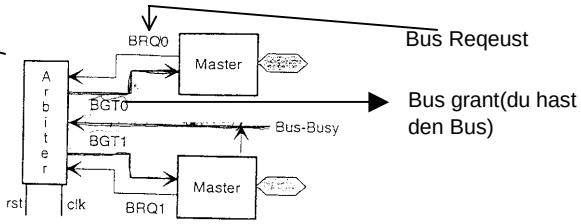
Zuteilung: zentral durch eine Schiedsrichter (Arbiter) oder verteilt

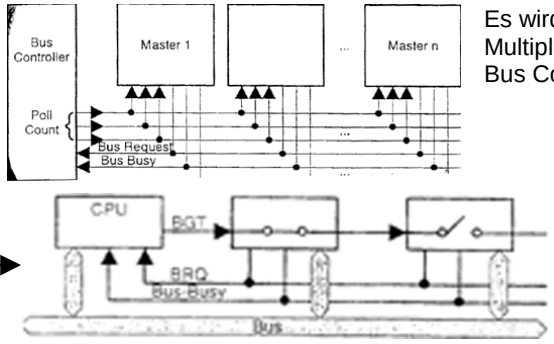
ZENTRALE vs. DENZENTRALE ZUTEILUNG:

ad **Zentral:**
Arbiter = Buscontroller

Vorteil: kurze Reaktionszeit Nachteil: „Zuverläßigkeits-Engpaß“

Bsp: -) individuelle Steuerung
(Pro Master 2 Ltg. schlechte Skalierbarkeit)
-) Pollen (Abfragen)
-) zentrale Daisy-Chain





Es wird nach der Reihe gefragt, bei Antwort, Bus Request (wie Multiplexer) und BUS BUSY wird von Master(n) zurückgeschickt an Bus Controller (asynchrones Verfahren).

ad DEZENTRAL:

Bsp: -) paralleles Polling
-) dezentrale Daisy-Chain (alle gleich, wie Token-Verfahren→ kein Konflikt)
-) Mehrfachzugriff (wie CSMA, kann explizit zu Konflikten führen, wird akzeptiert, es kommt nix zurück, wie bei Netzwerk (collission detection)

EIN/AUSGABE-Organisation:

Wie adressiert die CPU ein I/O Gerät?
1) Memory-Mapped I/O
   Zuordnung: Speicherbereich ←→Gerät
2) Dedicated I/O – OPCODES

Register in I/O-Gerät: Status/Kontroll-Information

CPU-„Rolle“: welche Möglichkeiten?

-) Polling: lfd. Statusabfrage
-) Interrupt-driver I/O: Nachteil: OS-Overhead
-) Hybride Lösung: period. Clock-Interrupts, dann Polling der Geräte

Problem: CPU-Zyklen wd. Datentransfer verbraucht → Delegation

DELEGATION:

-) DMA (Direct Memory Access) – Hardware: (Prozessorentlastung)
spezialisierter Prozessor; ist ein Bus-Master
oft realisiert als DMA-Controller
=DMA-fähiger I/O-Controller, die Datenübertragung zw. Speicher und Gerät selbständig (ohne Zuhilfenahme des Prozessors) abwickeln.
Prozessor initialisiert Schnittstellenbaustein & Controller (Laden von Steuer- u. Adressregister) dann startet er I/O-Operation.

Integration in den Rechnerkern?
-) transparent → Cycle stealing
-) Blocktransfer → DMA-Controller erhält den Bus (für Blocktransfer)

-) EIN/AUSGABEKANÄLE

realisiert durch attached DMA-fähige Spezialprozessoren mit einem eigenen Befehlssatz für I/O.
Enge Kopplung mit dem Hauptprozessor
gemeinsamen Speicher enthält Kanalprogramm + Kommunikationsbereich
CPU initialisiert den Kommunikationsbereich

Unterscheide:
-) *Multiplexer* (mehrere langsamere Geräte)
-) *Block-Multiplexer* (wie Multiplexer, jedoch Übertragung von Blöcken)
-) *Selektor* (für eine spezielle I/O-Operation, für schnelle Geräte)

Virtualisierung des Kanals!!

-) **I/O-Prozessor (=I/O Controller**, Channel Controller)

arbeiten von einem fixen oder einem durch das OS downgeloadede Programm
OS setzt Reihe(queue) von I/O-Kontroll-Blöcken ab
universelle Mikroprozessoren (erledigen zusätzlich. Funktionen)
(z.B. Fehlererkennung, Formatierung,...)

I/O-Prozessor – Unterschied zu CPU:
.) weniger allg. (haben spez. Aufgaben)
.) Information nicht verarbeitet, nur übertragen.

5. I/O PERFORMANCE

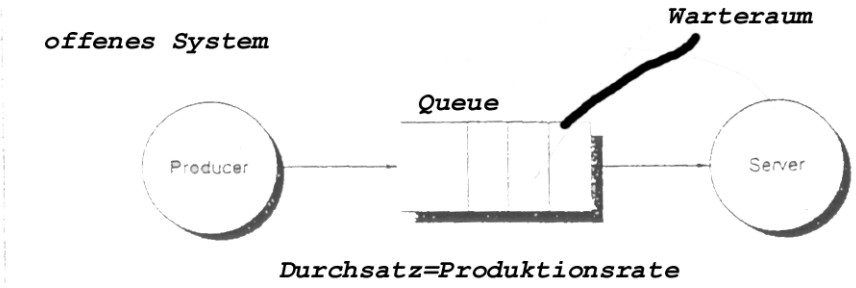
MASSZAHLEN: -) DURCHSATZ (=I/O-BANDWIDTH)
-) ANTWORTZEIT(=I/O-LATENCY)

Bedienzentrum (=Queue+Server)

R_Q + S = R

Wartezeit + Servicezeit = Antwortzeit

→ $\underline{R = R_Q + S}$



(Was drinnen passiert wissen wir nicht, ist wie eine BlackBox, es kommt etwas rein und nachher wieder raus. Vgl. Postamt)

(Traditionelles producer-server-Model oder response time and through-put)

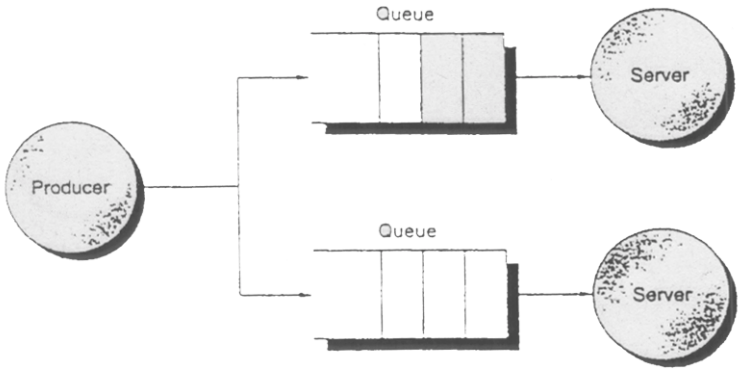


FIGURE 6.18 The single-producer, single-server model of Figure 6.16 is extended with another server and buffer. This increases I/O system throughput and takes less time to service producer tasks. Increasing the number of servers is a common technique in I/O systems. There is a potential imbalance problem with two buffers: Unless data is placed perfectly in the buffers, sometimes one server will be idle with an empty buffer while the other server is busy with many tasks in its buffer.

TRANSACTION – PHASES:

1) Entry Time
2) System Response Time +=Transaction Time (Produktivitätsmaß)
3) Think Time

“People need less time to think when given a faster response”

Queueing Theory:
Black Box Approach → Focus: Steady State(=Equilibrium)
nichts erzeugt/vernichtet

Queue (Waiting Line) + Server

Basic Performance Relations/Equations:

Little's LAW !!!!!!!!!!(ganz wichtig!!!)

$$[\lambda] = \frac{Ankünfte}{Zeiteiinheit} = \frac{n}{T}$$

mean no. of tasks in a system N = arrival rate λ * mean response time R
(Ankunftsrate)

-) Response time R = waiting time R_Q + service time S

-) avg. no of tasks in System N = avg. no waiting N_Q + avg. no in service N_S

-) server utilization $U = \frac{arrival\_rate\_\lambda}{service\_rate\_\mu}$ =N_S

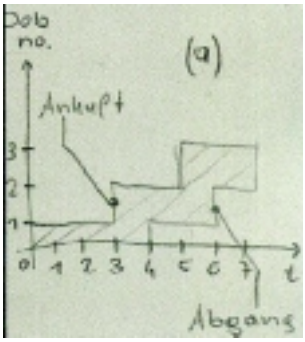
-) waiting time R_Q= no of tasks waiting N_Q * service time S +
avg. residual service time R_S ← (R_s=durchschnittliche Restbedienzeit)

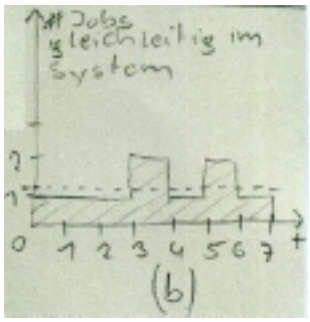
coefficient of variation C = variance/mean² ← (Mittelwert)

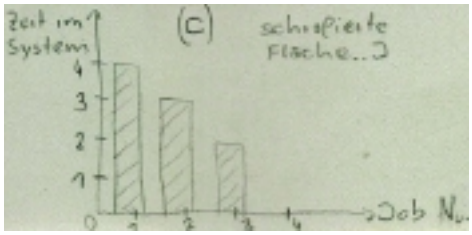
avg. residual service time $R_S = \frac{1}{2} * service\_time * (1 + C)$

R_S=Mittelwert über alle Höhen = $\frac{E[Dreiecksfläche]}{E[Höhe]} = \frac{E[\frac{s^2}{2}]}{E[s]} = \frac{1}{2} * [\frac{E[s^2]}{E[s]} =$

$$\frac{1}{2} * \frac{\vartheta^2 + E[s]^2}{E[s]} = \frac{1}{2} * E[s] * \frac{\vartheta^2 + E[s]^2}{E[s]} = \frac{1}{2} * E[s] * (c + 1)$$





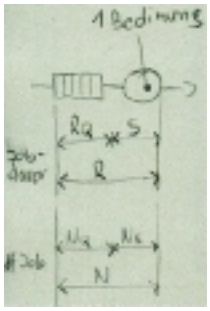


Ankunftsrate = $\lambda = \frac{n}{T}$

mittlere Zeit im System = $\frac{J}{n}$

mittlere Jobanzahl im System = $\frac{J}{T}$

$= \frac{n}{T} * \frac{J}{n} = \lambda * R = N$ $\mu = \frac{1}{s}$

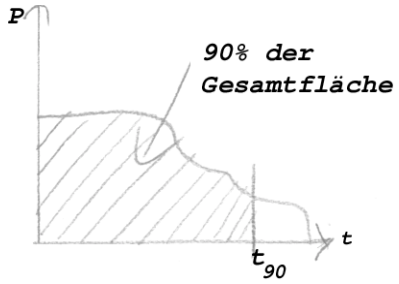


Wenn
C=0,5 → Hypoexponential
C=1 → exponential (Siehe Figure 6.23, bei Folie 126).
C=2 → Hyperexponential

→ R_S=1/2* service time + (1+C)

Mittlere Restzeit = Servicezeit →MEMORYLESS PROPERTY

Vergangenheit spielt keine Rolle (Wie lange braucht eine Operation noch?, für die Beurteilung ist es jedoch egal, wie lange er schon gebraucht hat.)
(Es wird irgendwann nach der Restlichen Zeit gefragt)



Was ist 90% Percentilwert?

Wenn 90% Der Fläche gemessen werden

Utilization = Traffic intensity
Queue discipline: z.B. first-in-first-out (FIFO, FCFS=First-come-first-serve)
Disbribution of random variables → histogram with buckets
Characterization by mean + measure of variance

d.h. weighted arithmetic mean $\sum_{i=1}^{n}\left(\frac{f_i * T_i}{\sum_{i=1}^{n} f_i}\right)$

variance: $\sum_{i=1}^{n}\left(\frac{f_i * T_i^2}{\sum_{i=1}^{n} f_i - weighted\_mean^2}\right)$

Problem: units used to measure
→ squared coefficient of variation C = variance/mean² (Variationskoeffizient U=λ*S)

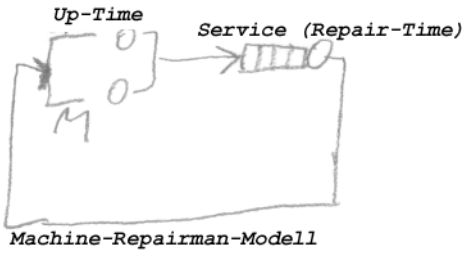
$R_Q = N_Q * S + U * R_S$ entspricht: ∅Kundenanzahl*Servicezeit+Bedienungsbelegt*Rest-Servicezeit

→ $R_Q = \frac{1+C}{2} * \frac{U}{1-U} * S$ c=1 → $R_Q = \frac{U}{1-U} * S$ (exponential verteilt)

U=1→λ=μ

Annahmen – Zusammenfassung:

-) System im Gleichgewicht
-) exponential interarrival times
-) infinite population model
-) work conservative
-) unbeschränkt FIFO=FCFS
-) alle Tasks beenden



Machine-Repairman-Modell

→**M/G/1** (von Kendall)

Anzahl der Bediener

Art des Ankunftsprozesses (Memoryless=Poisson-Prozeß)

General Verteilung der Servicezeit)zB D=deterministisch

DESIGNING AN I/O – SYSTEM

Ziel: Balanced System (Prinzip: schwächstes Glied)

Cost/Preformance Analysis of different I/O – organizations

Beurteilungskriterium: max. Durchsatz
(z.B. bei Annahme: Ressourcen 100% nutzbar)

Engpaß-Analyse + Analyse der Performance bei hoher Ressourcenauslastung!

M/M/m: $U = \lambda * \frac{S}{m}$ $R = S * [1 + \frac{U}{m * (1 - U)}]$

Faustregel: U sollte kleiner als best. Grenzwert sein

6. MULTIPROZESSOR – SYSTEME

Warum?
- PERFORMANCE
- KOSTEN-EFFEKTIVITÄT
- SKALIERBARKEIT
- VERFÜGBARKEIT
- ANPASSBARKEIT AN SPEZ: ANWENDUNGENSPROFILE
- GRACEFUL DEGRADATION
  (Leistungsverlust, langsam gegen 0 bei viel Prozessoren. Je nachdem wie viel Prozessoren vorhanden sind und wie viel ausfallen geht der Wert schneller oder langsamer gegen 0)

homogene vs. heterogene Parallelrechner
gleicher Typ unterschiedl. Prozessoren
zB gleiche CPU

symmetisch / asymmetrische Parallelrechner
alle CPU arbeiten ZB(CPU,VGA,..)
an gleicher
Aufgabe

TAXONOMIE (Flynn):

Instr. / Data	S	M
S	SISD	MISD
M	SIMD	MIMD

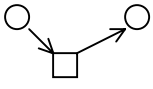
MISD=nicht wirklich existent
MIMD (in einem Vektor mehrere Elemente)
SISD(von Neumann-Rechner)

SIMD:
- **Vektorrechner**: zB Cray 1
  prozessorinterne Parallelität in Form einer/mehrerer Pipelines für Vektoroperationen
- **Feldrechner:** (Matritzenoperationen)
  eine zentrale Steuerung für alle Prozessorelemente (**streng synchronisiert)**

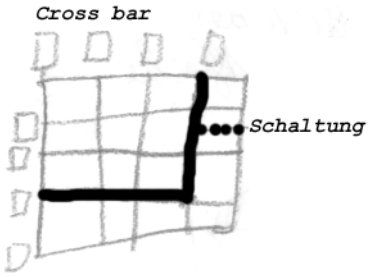
für spezielle Anwendungen mit einem Kontrollfluß und Operationen mehrfach auf unterschiedlichen Daten

MIMD:
KOMMUNIKATION DER VERARBEITUNGSELEMENTE

A) Speicherkopplung
(Cross bar)



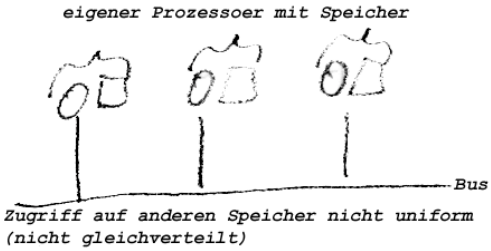
b) Nachricht:



- **UMA** (uniform memory access)(gleicher Zugriff auf Speicher)
  zentraler Hauptspeicher und uniformer Speicherzugriff (centralized shared memory)
  .) Verbindungsstruktur: - gemeinsamer Speicherbus, oder
  - Kreuzschienenverteiler
  Unterklasse SMP (symmetric multiprocessor)

- NUMA (non uniform memory access)
  physikalisch verteilter gemeinsamer Speicher
  kein uniformer Speicherzugriff (distributed shared memory)
  Verbindungsstruktur:
    - Bussysteme
    - statische/dynamische (mehrstufige) Verbindungsnetzwerke

  Sonderform:
  Cache-kohärenz NUMA- und COMA (cache only memory access) - Architekturen



B) BOTSCHAFTENKOPPLUNG

NORMA – (No-Remote-Memory-Access) Architektur
Direkter Zugriff nur auf lokalen Speicher (multicomputers)
Bsp: massiv parallele Rechner (MPP)
Workstation-Cluster (COW)

Weitere Architekturausprägungen:
- **DATENFLUSSRECHNER** (sobald Daten verfügbar sind)
  Ablaufsteuerung beruht nicht auf Kontrollflussprinzip →
  Verfügbarkeit von Operanden löst Ausführung von Op aus
  d.h. Steuerung durch Datenfluß

- SYNCHRONE ARCHITEKTUREN
z.B: systolische Felder, Neuronale Netze
PE von zentralen Taktgeber gesteuert
pipelineartige Datenverarbeitung entlang der Felddimensionen
Erweiterung: Wellenfront-Felder
(zentraler Takt durch Datenflussprinzip ersetzt)

Rechnerklassenauflistung Siehe Bild 8.1 (nach Folie 136)

VERBINDUNGSNETZWERKE

1) Statische: PE direkt miteinander verbunden,
führen Verbindungsaufbau selbst durch

2) Dynamische: keine direkte Verbindung zw. Pes →
Knoten mit „Schaltelementen“ („Router“)verbunden,
die für Verbindungsaufbau zuständig sind.

Beispiel: **BENES-NETZWERK**

Rekursiver Aufbau; Basis: 2 x 2 Crossbar
i.d.F.: 4x4, 8x8,…
$N=2^a$ Eingänge → $2 (\log_2 N)-1$ Stufen
→ N/2 Crossbars/Stufe
→ insges. $N * (\log_2 * N-1/2)$ Crossbars

nicht blockierend

OMEGA-NETZWERK
weniger Schalter ($\log_2$ N Schalterstufen), aber nicht blockierungsfrei

DYNAMISCHER HYPERCUBE
nur zwei Stufen, die mehrmals (max. $\log_2$ N +1) zu durchlaufen sind

ZIELE (bei Verbindungsnetzen):
- hohe Übertragungsleistung
- Zuverlässigkeit
- wirtschaftliche Erstellungs- u. Betriebskosten
- Skalierbarkeit

LEISTUNG: (eines Verbindungsnetzwerkes):
- Kapazität (Bandbreiten)
- Latenz

PERFORMANCE:
- Insufficient parallelism (Amdahl's Law)
- Large Latency or remote access
- cache coherence
- synchronisation

$$Zeit = \frac{Arbeit}{Leistung_{\max} * Wirkungsgrad}$$ (Wirkungsgrad<1, da in % angegeben)

Overhead Funktionen:

Wirkungsgrad verrringern
+ Algorithmic characeristics
  -serial part
  -work imbalance (Ungleichgewicht)
+ Architecture Interaction
  -latency (CPU_A schickt CPU_B Daten)
  -contention (Stau, wenn CPU_A CPU_C und CPU_B an CPU_D etwas schicken will, jedoch die Leitung besetzt ist durch A→C)

Amdahl:

p….# Prozessoren

Speed-up $S(p)=\frac{p}{1+f*(p-1)}$ f…..seq. Anteil(0,1)

(Beschleunigung durch Einsatz von mehreren Prozessoren)

Effizienz $E(p)=S(p)/p = \frac{1}{1+f*(p-1)}$

Effektivität: $\eta(p) = \frac{S(p)}{p/S(p)} = \frac{S(p)^2}{p} = S(p)*E(p)$

(Leistung-Kosten-Verhältnis)

$$\eta(p)=\frac{Leistung}{Kosten}=\frac{S(p)}{\frac{1}{\frac{S(p)}{p}}}=\frac{S(p)}{\frac{p}{S(p)}}=\frac{S^2(p)}{p}=S(p)*E(p)$$

1.Ableitung:

$$\frac{d\eta(p)}{dp}=0--->p_{opt}$$

p → ∞: S= 1/f; **E=0**

Ziel: E>0.5 → f< $\frac{1}{p-1}$

Anm: beliebig viele PE verfügbar → T_∞ = Exekutionszeit

p(t)...# aktiver PE zur Zeit t (Parallelitätsgrad)

mittlere Parallelitätsgrad $S_\infty = \bar{p} = \frac{T(1)}{T(\infty)}$

(PE=Prozesselement)

E(p)<=min { $\frac{\bar{p}}{p}, \frac{1}{[1+f*(p-1)]}$ } Mass für Auslastung der Prozessoren

Prozessor-Thrashing: Kommunikationsaufwand wächst mit Anzahl der PE
sonst skalierbar

SKALIERTER SPEED-UP:

$$S_p^s=\frac{T(1)}{T}=F+(1-f)*p=p-(p-1)*f$$

SKALIERTE EFFIZIENZ:

$E^s(p)=1-(1-\frac{1}{p})*f$

$S_{Amdahl}=\frac{1}{f+\frac{1-f}{p}} --> \frac{1}{f}$

$S_{Gustafson}=\frac{f+p(1-f)}{1} --> linear$

ZUVERLÄSSIGKEIT & LEISTUNG:

T(p)...Laufzeit (fehlerfreies System)
T^F(p)...Laufzeit (fehleranfälliges System)

$S^F(p)=\frac{T^F(1)}{T^F(p)}$ Beschleunigung

$D(p) = \frac{T(1)}{T^F(p)}$

A_1....Verfügbarkeit des Monorechners
$T(1) = T^F(1)*A_1$

$S^F(p)=\frac{F^F(1)}{T(1)}*\frac{T(1)}{T^F(p)} = D(N)/A_1$

Beachte: **speed-up erzielt, auch wenn Applikation nicht parallelisierbar!**

f=1: $S^F(p)=A_p/A_1$ da $T(1)=T(p)=T^F(p)*A_p$
→ $D(p)=A_p$

redudanter Parallelrechner zuverlässiger als nicht redundanter Monorechner.

Speed-up umso größer, je zuverlässiger der Monorechner ist.

PREIS/LEISTUNGSVERHÄLTNIS:

$K(p)=C(p)*T(p)=C(p)*\frac{T(1)}{S(p)}$

T^{-1}(p)....erzielbare Leistung
C(p)....Kosten

Einsatz von p Prozessoren okay, wenn

K(p) < K(1), bzw. $S(p)> \frac{C(p)}{C(1)}$

Bsp: m…Speicherkosten, g....Kosten f. Grundausstattung (Gehäuse,...)
m/4....Kosten für lokalen Speicher je Prozessor

Somit: C(1)=g+5+m
C(p)=2g+5p+m+m*p/4
5p…. Kosten für p Verarbeitungseinheiten (Anm: m=g=10(*10^3DEM)