Zusammenfassung Rechnerstrukturen SS97

INHALT:

1. Grundlagen

1. *Klassifizierung nach Flynn*

- SISD - Single Instruction, Single Data
- SIMD - Single Instruction, Multiple Data
- MISD - Multiple Instruction, Single Data
- MIMD Multiple Instruction, Multiple Data

SISD: klassische Von-Neumann-Architekturen (Einprozessorsysteme)

SIMD: (Feldrechner, Vektorrechner)

MISD: leer

MIMD: Multiprozessorsysteme

Nachteile der Klassifizierung:

- Hohes Abstraktionsniveau (sehr unterschiedliche Architekturen in einer Klasse)

- Moderne Techniken wie Superskalar, Pipelining, VLIW lassen sich nicht einordnen, bzw. gehören mehreren Klassen an (z.B. Vektorrechner-Multiprozessoren: MIMD/SIMD (mehrere Prozessoren und internes Vektorpipelining))

Körnigkeit: Anzahl der Befehle in einer *sequentiellen Befehlsfolge* (total geordnete Teilmenge von Befehlen eines parallelen Programms)

1. ***Ebenen der Parallelität***

- **Programmebene (Jobenbene):**
  parallele Verarbeitung verschiedener Programme (vollständig unabhängig voneinander); auf der Betriebssystemebene organisiert
- **Prozessebene (Taskebene):**
  parallel auszuführende Prozesse innerhalb eines Programms (schwergewichtige Prozesse, UNIX-Prozesse); viele tausend auszuführende Befehle pro Prozess; Prozessverwaltung, Prozesssynchronisation und Prozessverwaltung werden benötigt.
- **Blockebene:**
  Anweisungsblöcke (leichtgewichtige Prozesse, threads) mit wenigen Anweisungen, die auf denselben Adreßraum, wie andere leichtgewichtige Prozesse zugreifen. Leichtgewichtige Prozesse meist innerhalb von schwergewichtigen Prozessen.
- **Anweisungsebene:**
  Maschinenbefehle oder elementare Anweisungen können parallel zueinander ausgeführt werden. (Meist erkennt dies der Compiler und setzt das Programm entsprechend um; hier selten explizite Angaben der Parallelität (Beispiele: Befehlspipelining, Superskalar, VLIW, Überlappung von E/A- mit CPU-Operationen)
- **Suboperationsebene:**
  elementare Anweisungen werden durch den Compiler oder die Maschine in parallel ausführbare Suboperationen zerlegt (z.B. Vektor-/Feldoperationen)

VHSIC = very high speed integrated circuit

VLSI = very large scale integrated circuit

1. ***Ziele beim Entwurf eines Rechners***

- **Hohe Leistung (Kosteneffektivität bzgl. Verarbeitungsgeschwindigkeit)**
- **Erweiterbarkeit (Erweiterung der Fähigkeiten, in jedem Zwischenzustand korrekt funktionsfähig)**
- **(Objektcode-)Kompatibilität (gleicher Arbeitsauftrag à gleiche Ergebnisse)**
- **Flexibilität (große Klasse von Problemen kann bearbeitet werden - "Universalrechner" statt "Spezialrechner")**
- **Benutzerfreundlichkeit (Programmierung und Benutzung)**
- **Verläßlichkeit (Gewährleistung einer gewissen Minimalverfügbarkeit des Systems)**

1. Gestaltungsgrundsätze beim Entwurf eines Rechners

- **Konsistenz (folgerichtiger, schlüssiger Aufbau)**
- **Orthogonalität (funktionell unabhängige Teilelemente auch unabhängig voneinander spezifiziert und realisiert)**
- **Symmetrie (mathematisch symmetrische Eigenheiten des Systems auch symmetrisch entwerfen)**
- **Angemessenheit (Funktionen des Systems sollen bei der Lösung der Problemstellung auch ausgeschöpft werden)**
- **Sparsamkeit (Ökonomie) (Kosten möglichst gering halten)**
- **Wiederverwendbarkeit (von Komponenten)**
- **Transparenz (verschiedene Funktionen des Gesamtsystems bleiben unsichtbar)**
- **Virtualität (Angebot von real nicht existierenden Funktionen)**

1. ***Bewertung der Leistungsfähigkeit***

Maßzahlen für die Operationsgeschwindigkeit:

MIPS (millions of instruction per second): 70% Additionsbefehle, 30% Multiplikationsbefehle è 1 MIPS = 700.000 Additionen und 300.000 Multiplikationen pro Sekunde

MFLOPS (millions of floating point operations per second) (Bei Vektorrechnern asymptotische Maßzahl)

MLIPS (millions of logical inferences per second) Inferenz = Ableitung einer Aussage aus anderen Aussagen

Laufzeitmessungen bestehender Programme

Benchmarks bestehen aus einem oder mehreren Programmen im Quellcode

SPEC-Benchmark (heute meist verwendet; besteht aus 6 Integer- und 14 Gleitkommaprogrammen)

Whetstone-Benchmark (einziges Programm, bestehend aus: Gleitkommaoperationen, Integeroperationen, Array-Index-Operationen, Prozeduraufrufe, bedingte Sprünge und Aufrufe von mathematischen Standardfunktionen)

Drystone-Benchmark (ursprünglich in ADA geschrieben, meist C-Version benutzt. Ada-Version besteht aus ca. 100 Anweisungen: 53% Zuweisungen, 32% Steueranweisungen und 15% Prozedur- und Funktionsaufrufe.)

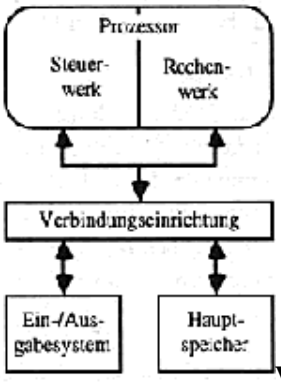
Messungen während des Betriebs der Anlagen

- **Monitore (Hardwaremonitore, Softwaremonitore)**

Modelltheoretische Verfahren

- **Häufig Warteschlangenmodelle**
- **Analytische Methoden**
  deterministische Warteschlangenmodelle
  stochastische Warteschlangenmodelle
  operationelle Warteschlangenmodelle
- **Simulationen**
  deterministische Simulationen
  stochastische Simulationen
  aufzeichnungsgesteuerte Simulationen

1. Architektur und Implementierung von Mikroprozessoren



Von Neumann Architektur (siehe Abb. 1)

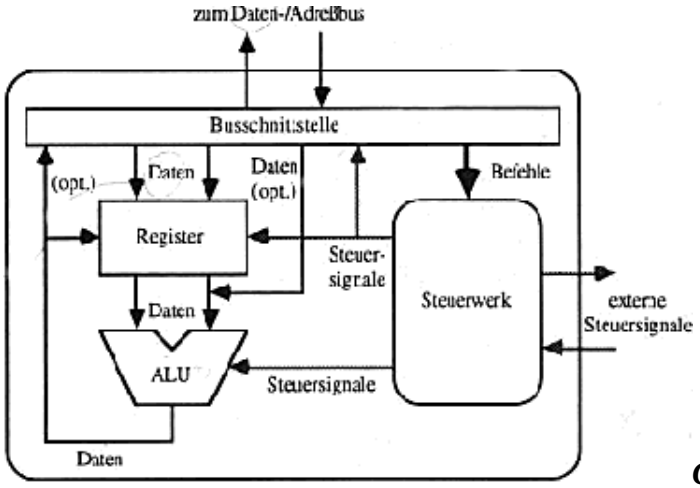
Abb.1: Prinzipielle Struktur eines von-Neumann-Rechners

- **Prozessor (Steuerwerk (mit Befehlszähler), Rechenwerk)**
- **Verbindungseinrichtung (Bus)**
- **Ein-/Ausgabesystem**
- **Hauptspeicher (für Daten *und* Programme)**

1. *Leistungssteigerungen aus der von-Neumann-Ära*

- Zusatzregister (z.B. Indexregister etc.)
- Einführung von Speicherhierarchien (Pyramide: Register, primary cache, secondary cache, Hauptspeicher, Sekundärspeicher, Archivspeicher)
- Trennung von Code und Daten (Harvard Architektur, wenn beides in getrennten Speichern liegt. Nachteil: Universelle Verwendbarkeit des zur Verfügung stehenden Speichers geht verloren.)
- Befehlspipelining (Überlappende Ausführung von Befehlen: Befehl holen, Befehl dekodieren, Operand holen, Operand verarbeiten, Ergebnis abspeichern)
- Arithmetisches Pipelining: Überlappende Ausführung bei der Operandenverarbeitung (z.B: Gleitkommamultiplikation
- Vektorpipelining: Überlappte Verarbeitung ganzer Vektoren von Gleitkommazahlen (Vektorrechner)
- Einsatz von Coprozessoren zur Unterstützung der CPU
- Vervielfachung der Datenprozessoren (ein Befehlsprozessor, superskalar, VLIW)
- Mehrere Verarbeitungseinheiten (Multiprozessoren)
- Analyse des Datenflusses (à Datenflußrechner)

- Selbstidentifizierende Daten (à assoziative Rechner)
- Sprachorientierte Architektur
- Selbstbeschreibende Daten
- Architektur mit Typkennung
- Deskriptorarchitektur

1. 

Grundlegender Aufbau eines Mikroprozessors

Abb.2: Struktur eines einfachen Mikroprozessors

CISC: Complex Instruction Set Computer

RISC: Reduced Instruction Set Computer

Superskalare Prozessoren laden/beenden pro Takt mehrere Befehle und bestehen aus:

- **Lade-/Speichereinheit (Load/Store Unit)**
- **MMU: übersetzt logische Adressen in physikalische Adressen**
- **Gleitpunkteinheiten (Floating-Point Units)**
- **Festpunkteinheiten (Integer Units)**
- **Verzweigungseinheit (Branch Unit) überwacht Ausführung von Sprungbefehlen**
- **Sprungtabelle (Branch History Table, BHT) um zu schätzen, wo es weitergeht**
- **Sprungziel-Adress-Cache (Branch Target Address Cache, BTAC)**
- **Befehlszuordnungseinheit (Instruction Dispatch Unit) verteilt die Befehle, die gleichzeitig ausgeführt werden sollen**
- **Rückordnungspuffer (Reorder Buffer) "bügelt falsch gemachte Sprünge aus". In ihn werden die zugeordneten Befehle von der Befehlszuordnungseinheit in ihrer Programmreihenfolge eingetragen.**

1. *Organisation der Befehlspipeline*

1. **Befehlsbereitstellungsphase (Instruction Fetch)**
2. **Decodierphase (Decode)**
3. **Operandenbereitstellungsphase (Operand Fetch)**
4. **Ausführungsphase (ALU Operation)**
5. **Resultatspeicherphase (Write Back)**

Superpipelining: noch feinere Stufen.

Superskalar: räumliche Parallelilität

Superpipelining: zeitliche Parallelität

1. ***Pipelinekonflikte und deren Behandlung***
    1. **Daten- und Kontrollabhängigkeiten**

Kontrollabhängigkeiten (control hazards)

Auswahl des nachfolgenden Befehls hängt vom Ergebnis des vorhergegangenen ab. (z.B. beim Sprung)

Datenabhängigkeiten (data hazards)

- **Echte Datenabhängigkeit (True Dependence): Befehl schreibt in Register, nächster liest dieses**
- **Gegenabhängigkeit (Anti Dependence): Befehl liest Register, welches der nächste Befehl beschriebt**
- **Ausgabeabhängigkeit (Output Dependence): zwei Befehle beschreiben nacheinander dasselbe Register.**

Gegen- und Ausgabeabhängigkeit werden auch als Pseudoabhängigkeiten (= scheinbare Abhängigkeiten; können durch Registerumbenennung aufgelöst werden)

```
S1:  ADD R2,2,R1 ;  R1 = R2+2
S2:  ADD R1,R3,R4  ;  R4 = R1+R3
S3:  MULT R5,3,R3  ;  R3 = R5*3
S4:  MULT R6,3,R3  ;  R3 = R6*3
```

Abb 3 gibt ein Beispiel:

Abb.3: Abhängigkeitsgraph

1. Sonstige Pipelinekonflikte

Ressourcen-Zugriffskonflikte: (structural hazards)

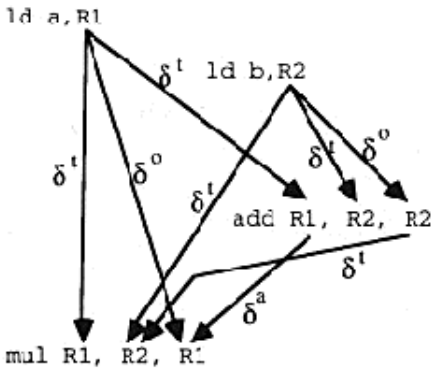
- **Intra-Pipeline-Ressourcenkonflikt: Befehle aus zwei aufeinanderfolgenden Pipelinestufen benötigen im selben Takt die selbe Ressource (z.B. bei mehreren Divisionen hintereinander)**
- **Befehlsklassenkonflikt (nur bei Superskalar und VLIW): Befehle aus der gleichen Pipelinestufe benötigen im selben Takt die selbe Ressource, die jedoch nur einmal vorhanden ist.**

Auch die drei Arten von Datenabhängigkeiten können zu Pipelinekonflikten führen:

- **Lese-nach-Schreibe-Konflikt (read after write)**
- **Schreibe-nach-Schreibe-Konflikt (write after write)**

```
m1:  lda, R1        ; R1 := [a]
m2:  ldb, R2        ; R2 := [b]
m3:  add  R1, R2, R2     ; R2 := R1 + R2
m4:  mul  R1, R2, R1     ; R1 := R1 * R2
```

Der zugehörige Abhängigkeitsgraph



- **Schreibe-nach-Lese-Konflikt (write after read)**

Abb.4: Beispiel

1. Hardwaretechniken zur Behandlung von Pipelinekonflikten

- Leerlaufen der Pipeline (Einschieben von Wartezyklen, evtl. Arbitrierung notwendig (Schiedsrichterlogik))
- Ressourcen, auf die gleichzeitig mehrfach zugegriffen werden kann
- Trennung der Ressourcen nach logischer Funktion (vergl. Harvard-Architektur)
- Vervielfachung der Ressourcen

Scoreboard-Technik zur Erkennung von Datenabhängigkeiten:

In einem Scoreboard-Register werden in der Decodierphase die benutzten Register markiert.

Registerumbenennung: Vermeidung von Konflikten durch Pseudoabhängigkeiten (s.o.)

Register-Mapping-Technik: Register werden durch Hardware dynamisch zugeteilt

Register-Bypass-Techniken zur Vermeidung von Pipelinekonflikten bei echten Datenabhängigkeiten:

- Load Forwarding: zur Vermeidung von Pipelinekonflikten nach Lade-Befehlen. Daten werden direkt aus dem Cache in die ALU geschaltet - nicht über Register.
- Result-Forwarding: das Ergebnis der ersten Operation wird unmittelbar wieder an die ALU gegeben (für die nächste Operation)

Verzweigungseinheit (Branch Unit):

- Zeitverzögerung durch die Berechnung einer neuen Adresse (Umsetzung von logischer in physikalische Adresse)
- Verzögerung durch die Auswertung der Bedingung bei bedingtem Sprung.

Gegenmittel: **Sprungzielcache** (Branch Target Cache)

Delayed-branch-Technik oder **Branch-and-execute-Strategie**: Befehle nach dem Sprungbefehl einfügen, die nichts verändern, doch die Ausführung des ersten dem Sprungbefehl folgenden Befehls solange verzögern, bis der Sprungbefehl komplett ausgeführt ist.

Techniken zur **Sprungvorhersage** (Branch Prediction):

- **Statische Sprungvorhersage:** Richtung der Spekulation steht von vorn herein fest. Es wird z.B. in den bedingten Sprungbefehlen mitcodiert, wie wahrscheinlich der Sprung ist
- **Dynamische Sprungvorhersage:** es wird versucht, zur Laufzeit herauszufinden, wo das Programm wahrscheinlich weitergeht. (z.B. 2 bit branch prediction mit Werten (strong not taken, weak not taken, weak taken, strong taken), welche eine Vorhersagegenauigkeit bis zu 97% haben)

Wenn falsch geschätzt wurde, müssen bereits ausgeführte Operationen wieder rückgängig gemacht werden, z.B. per **History Buffer** (LIFO), welcher auch für präzise Unterbrechungen (Precise Interrupts) verwendet wird.

Dreiadressbefehle: 2 Operandenregister, 1 Zielregister (z.B. ADD R1, R2, R3)

1. ***Registerorganisation***

   Registerfenster bestehen aus INS, LOCALS und OUTS. Beispiel:

   Umlaufspeicherorganisation.

   CALL springt in Unterprogramm

SAVE wechselt im Unterprogramm das Fenster

RESTORE schaltet wieder auf das vorherige Fenster

Window-overflow: wenn zu viele Fenster belegt werden (SAVE)

Window-underflow: wenn genau ein Fenster belegt ist und RESTORE

Diese Fehler werden durch ein gesetztes Bit im WIM (window invalid mask-Register) erkannt.

2. *Speicherverwaltung*

Speicherverschränkung (Memory Interleaving):

A_i wird auf die Speicherbank M_j gesichert Û j = i mod n (Verschränkungsregel - Interleaving Rule)

n-fache Verschränkung = Verteilung auf n-Speicherbänke

Speicherhierarchie: Register, Cache, Arbeitsspeicher, Sekundärspeicher, Hintergrundspeicher

1. Virtuelle Speicherverwaltung

virtuelle Adressen werden über eine Übersetzungstabelle (Translation Table) in physikalische Adressen umgewandelt.

Beim virtuellen Speicher kann es zu einem Seitenfehler (Page Fault - Daten nicht im physikalischen Speicher) kommen à Daten (Page) müssen von einer Betriebssystemroutine nachgeladen werden (Paging)

Oft werden zwei Übersetzungstabellen benutzt: die Seitentabelle und die Segmenttabelle

- In der Seitentabelle werden Seiten fester Größe verwaltet; die virtuelle Adresse wird in Seiten- und Offsetadresse aufgeteilt
- Die Segmenttabelle verwaltet Segmente variabler Länge (für einzelne Prozesse) - damit kann auch ein Speicherschutzmechanismus, der die Zugriffsrechte der einzelnen Prozesse regelt, realisiert werden (?)
- In der Seitentabelle gibt es Verwaltungsbits wie z.B. das PRESENT-Bit. Es zeigt an, ob die Seite im Hauptspeicher vorhanden ist, oder ob sie nachgeladen werden muß (Page Exception)

TLB (beim Power-PC) (Translation Look Aside Buffer) speichert die wichtigsten Adressübersetzungspaare (virtuell zu physikalischer Seitenadresse). Dies erspart Umrechnungszeit.

Zur hardwaremäßigen Adressumsetzung werden MMUs (Memory Management Unit) verwendet.

1. *Cachespeicher*

Mittlere Zugriffszeit auf Cache: $\mathbf{t_{Acess} = Hit\text{-}Rate * t_{Hit} + (1 - Hit\text{-}Rate) * t_{Miss}}$

Cache=SRAM, Hauptspeicher=DRAM

Assoziativität: Anzahl der Blockrahmen in einem Satz

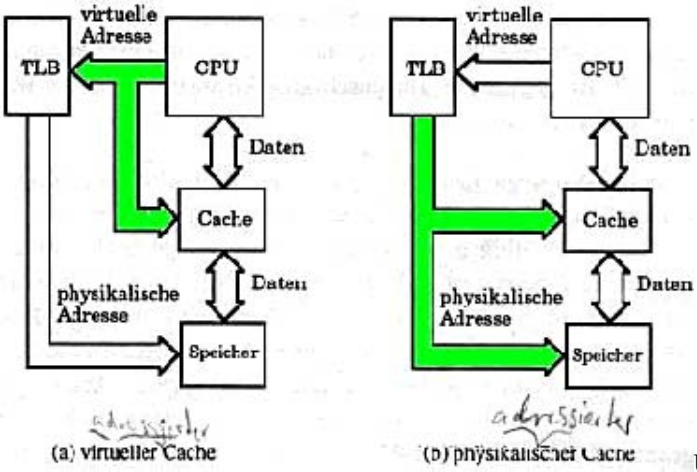
- Vollassoziativ: nur ein Satz **(LRU)**
- Direktabgebildet: jeder Satz hat nur einen Blockrahmen (Cache-Line)
- N-fach assoziativ: n Blöcke pro Satz **(LRU)**

Durchschreibe-Strategie (Write-Through-Cache-Strategie): vom Prozessor wird ein Datenwort immer gleichzeitig in den Cache und in den Hauptspeicher geschrieben

Gepufferte Durchschreibe-Strategie: zur Zeiteinsparung werden Änderungen zuerst im Cache gepuffert und dann im Hauptspeicher vorgenommen, während der Prozessor schon mit nachfolgenden Operationen weiterfährt.

Rückschreibe-Strategie: mit Dirty-Bit wird ein geändertes Wort im Cache markiert - der Hauptspeicher wird erst geändert, wenn das Wort aus dem Cache fliegt (Verdrängungsstrategie)

Sekundär-Cache außerhalb des Prozessors - oft parallel zum Hauptspeicher am Bus (Look-aside-Cache)



Unterscheidung zwischen virtuell und physikalisch adressiertem Cache (siehe Abb. 5):

Abb.5: Die zwei Möglichkeiten der Cache Adressierung

Nicht-blockierender Cache: erlaubt nach einem Fehlzugriff den Zugriff auf andere Cache-Lines (bevor der den Fehler verursachende Befehl abgearbeitet ist)

Cache Kohärenzproblem wenn mehrere Prozessoren mit jeweils eigenen Caches auf den selben Hauptspeicher zugreifen.

Kohärenz: korrektes Voranschreiten des Systemzustands durch ein abgestimmtes Zusammenwirken der Einzelzustände. (Das System muß dafür sorgen, daß immer aktuelle Daten geladen werden)

Ein System ist **konsistent**, wenn alle Kopien eines Datenwortes im Hauptspeicher und in den verschiedenen Caches *identisch* sind (dies stellt die Kohärenz sicher).

Beispiel: Das MESI-Protokoll ist ein Cache Kohärenzprotokoll mit Busschnüffeln (Bus Snooping; Abhören der Daten, die von anderen Prozessoren auf den Bus gelegt werden und Vergleich mit denen, welche sich im eigenen Cache befinden)

1. *Die VLIW-Technik*

- VLIW = Very Long Instruction Word (bis zu mehreren hundert Bits lang)
- Es werden immer stur n-Tupel abgearbeitet
- Flexibler sind superskalare Prozessoren
- Vorteile: Zuordnung
- Nachteile: Cache-Misses

1. *Mehrfädrige Prozessortechnik (multithreaded)*

- Mehrere Registersätze für verschiedene Kontrollfäden (Threads) vorhanden
- Bei Wartezeiten wird per Hardware ein Thread-Wechsel durchgeführt (Umschalten auf Registersatz eines anderen geladenen Kontrollfadens). Man nennt diese auch Rapid-Context-Switching-Prozessoren

Cyle-by-cycle Interleaving: (Fine-grain Multithreading):

> In jedem Takt wird von einem Hardware-Scheduler ein "ausführbarer" Thread gewählt und mit diesem weitergemacht. "Ausführbar" ist ein Thread, wenn sich keiner seiner Befehle mehr in einer Pipeline befindet.

Block-Multithreading (Coarse-grain Multithreading)

Die Threads werden nur bei Wartezeiten umgeschaltet

Vielfädrig superskalar: Kombination aus Multithreading und Superskalar-Technik

1. *Datenflußprinzip*

Datengesteuerte Befehlsausführung: ein Maschinenbefehl wird ausgeführt, sobald alle Operanden zur Verfügung stehen.

- Statische Datenflußrechner:
  Ein Knoten ist schaltbereit, sobald auf allen Eingangskanten (bei der Rückkopplungsmethode auch auf den Bestätigungskanten) Token vorhanden sind.
- Dynamische Datenflußrechner: (Tagged Token Datenflußrechner)
  Ein Knoten ist schaltbereit, sobald auf allen Eingangskanten Token mit identischen Tags vorhanden sind.

Dabei sehen Token so aus: c.s.i mit Context, Activity Number, Iteration Number.

Übergeben wird dann <c.s.i,v> mit zusätzlicher (zusätzichen) Variablen.

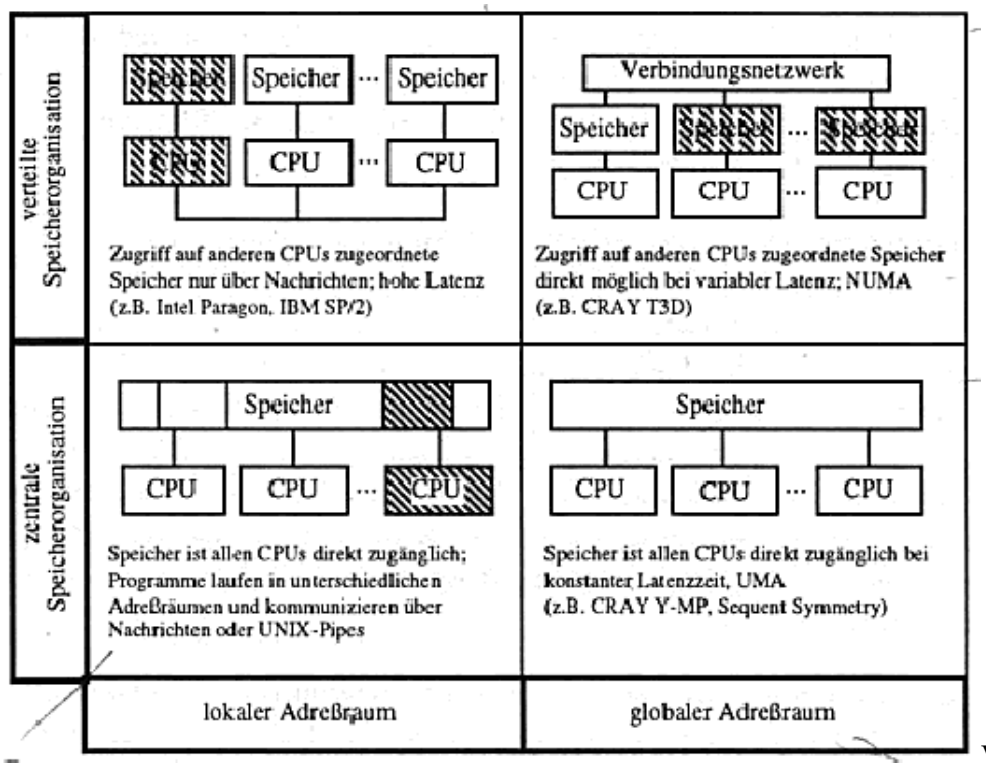
Matching: Vergleichen des Tags eines Tokens mit dem Tags aller bereits vorhandenen Token.

(MIT Tagged Token Dataflow Machine)

1. Multiprozessorsysteme

- Sind aus der MIMD Klasse
- Rechnernetze sind keine Multiprozessorsysteme: räumliche Trennung; niedrigere Übertragungsgeschwindigkeit
- Homogenes Multiprozessorsystem: hardwaretechnisch gleichartige Prozessoren; sonst: inhomogen
- Symmetrisches Multiprozessorsystem: Prozessoren sind bezüglich ihrer Rolle im System vergleichbar; sonst: asymmetrisches Multiprozessorsystem.
- Zentrale Systemaufsicht: Multiprozessoren stehen unter der Kontrolle eines für alle Prozessoren gemeinsamen Betriebssystems.

- Speichergekoppelte Multiprozessoren hängen an ein und demselben Speicher (stark oder eng gekoppelt)
- Nachrichtengekoppelte Multiprozessoren besitzen nur lokale Speicher (schwach oder lose gekoppelt)



Vier Möglichkeiten der Speicherverwaltung (siehe Abb. 6):

Abb.6: Konfiguration von Multiprozessoren

Verschiedene speichergekoppelte Multiprozessoren:

- Uniform-Memory-Access-Modell (UMA); gemeinsamer Speicher, Zugriffszeit für alle gleich, lokale Caches möglich.
- Nonuniform-Memory-Access Modell (NUMA); Zugriffszeiten variieren, gemeinsamer Speicher physikalisch auf Prozessoren verteilt (Distributed-Shared-Memory-Systeme). Zugriff auf "lokalen" Speicher geht schneller.
- Cache-Coherent-Nonuniform-Memory-Access-Modell (CC-NUMA); wie NUMA, aber Zugriff auf entfernte Speicher über Cache
- Cache-Only-Memory-Architecture-Modell (COMA); Spezialfall des NUMA; gesamter Speicher besteht ausschließlich aus den Caches der Prozessoren. Sie besitzen einen gemeinsamen Adressraum, organisiert durch verteilte Cache-Directories. Daten wandern praktisch in den Cache-Speicher des Prozessors, der sie benötigt. Sie stehen nicht fest im Speicher.

Multicomputer: Multiprozessoren, die aus mehreren autonomen Rechnern bestehen.

1. ***Verbindungsstrukturen***

1. Beurteilungskriterien

- Komplexität oder Kosten
- Diameter oder Durchmesser
- Regelmäßigkeit
- Notwendige Leitungslängen

- Blockierung (blockierungsfrei)
- Erweiterbarkeit
- Skalierbarkeit
- Ausfalltoleranz oder Redundanz
- Durchsatz oder Übertragungsbandbreite
- Pfadberechnung oder Wegfindung
- Übertragungszeit einer Nachricht (Startzeit, Transferzeit)
- Minimale Bisektionsbreite (b)
- Diskonnektivität (D=N/b) (N = Anzahl der Knoten)

1. Unterscheidungsmerkmale

- **Statische Netze: fest installierte Verbindungen zwischen Paaren und Netzknoten**
- **Dynamische Netze: enthalten ein Schaltnetz, an das alle Knoten über Ein- und Ausgänge angeschlossen sind**
- **Art des Datentransfers:**
- **Paketvermittlung (packet switching): Daten werden zerhackt und als Pakete mit Zieladresse verschickt (verwendet im Store and Forward Modus - Knoten speichert Paket und schickt es dann irgendwann weiter)**
- **Durchschaltevermittlung oder Leitungsvermittlung (circuit switching): Es wird eine direkte Verbindung zwischen zwei oder mehr Knoten geschaltet (physikalisch oder logische Verbindung). Dies ist meist schneller als die Paketvermittlung, allerdings ist eine Verbindung nicht immer möglich.**

1. *Dynamische Verbindungsnetze*

Multiprozessorsysteme mit:

- **Einfachbus**: ein Bus, an dem bis zu 30 Prozessoren hängen
- **Mehrfachbus**: mehrere Busse verbinden jeden Prozessor mit jedem Speicher
- **Hierarchischem Bus**: "Clusterbus-Systeme"
- **Kreuzschienenverteiler**
- **Schalternetzwerk**: z.B. aus Zweierschaltern (Permutationsnetze) mit zwei Eingängen und zwei Ausgängen (=Betaelemente oder Permutationszellen), die entweder durchschalten oder Ein- und Ausgänge kreuzen.

Zu Schalternetzwerken:

- **Einstufige Permutationsnetze**: enthalten eine Spalte mit diesen Schaltern
- **Mehrstufige Permutationsnetze**: enthalten mehrere Spalten dieser Schalter
- **Reguläre Permutationsnetzwerke**: p Eingänge, p Ausgänge, k Stufen mit jeweils p/2 Zweierschaltern
- **Irreguläre Permutationsnetzwerke**: weisen Lücken auf.

Permutationsarten:

- **Mischpermutation M** (Perfect Shuffle):
  $M(a_n, a_{n-1}, \dots, a_2, a_1) = (a_{n-1}, \dots, a_2, a_1, a_n)$
- **Kreuzpermutation K** (Butterfly)
  $K(a_n, a_{n-1}, \dots, a_2, a_1) = (a_1, a_{n-1}, \dots, a_2, a_n)$
- **Tauschpermutation T** (Negation des niederwertigsten Adressbits)

$T(a_n, a_{n-1}, ... , a_2, a_1) = (a_n, a_{n-1}, ... , a_2, "-"a_1)$ (also a_1 invers)

- **Umkehrpermutation U**
  $U(a_n, a_{n-1}, ... , a_2, a_1) = (a_1, a_2, ... , a_{n-1}, a_n)$

Omeganetzwerk: Netzwerk für $p=2^n$ Ein-/Ausgänge ; n = ld p Stufen von Zweierschaltern, die untereinander nach dem Grundmuster der **Mischpermutation** verknüpft sind.

- **Speichergekoppeltes Omeganetzwerk** verbindet Prozessoren mit Speichermodulen
- **Nachrichtengekoppeltes Omeganetzwerk** verbindet Prozessoren untereinander (diese haben jeweils einen lokalen Speicher)

Gesamtzahl der Zweierschalter in einem Omeganetzwerk mit $p=2^n$ Ein-/Ausgängen: (p/2)* ld p

Omeganetzwerke sind blockierend (weniger Verbindungen als möglich realisierbar) è kein universelles Permutationsnetzwerk

Recirculating Networks: Einstufiges Omeganetzwerk realisierbar, indem Daten mehrfach durch das Netz geführt werden.

Switching-Banyan-Netwerk:

Stufen werden blockweise nach dem Muster der Kreuzpermutation gekoppelt. (inkrementierende Blöcke (2^n mit n=0..), innere Bits werden vertauscht)

1. *Statische Verbindungsnetze*

Ausgänge sind fest zugeordnet:

- **Eindimensionales statisches Verbindungsnetzwerk**: z.B. Kette
- **Zweidimensionales statisches Verbindungsnetzwerk**: {Ring, Chordaler Ring (Ring mit Sehnen), Stern, Baum, Gitter mit vier Nachbarknoten, Gitter mit acht Nachbarknoten}
- **Dreidimensionales statisches Verbindungsnetzwerk**: {Würfel, Pyramide}
- **Vier- oder höherdimensionale statische Verbindungsnetzwerke**: {Hyperkubus (hyper cube, n-cube), Ring-Würfel-Netzwerk (Cubus verbundenes Zyklusnetzwerk ("Würfel mit abgeschliffenen Ecken" (Jede Ecke 3 Rechner))
- **Eigenschaften**:
- **Maximaler Verbindungsgrad**: Maximum von Verbindungen an einem Knoten
- **Diameter**: Längster Weg zwischen 2 Knoten im Netz
- **Minimale Bisektionsbreite**: Netz in der Mitte durchteilen und gucken, wieviele Verbindungen man zerstört hat. (Hier ist natürlich nur der schonenste Schnitt gemeint)
- **Diskonnektivität**: Anzahl der Knoten / minimale Bisektionsbreite

1. *Leistungsfähigkeit von Multiprozessoren*

Wichtig für den Vergleich von Multiprozessorsystemen mit Monoprozessorsystemen ist ein Programm, das auf beiden Systemen läuft.

Maßzahlen von **Lee** für diesen Vergleich:

P(1): Anzahl der auszuführenden (Einheits-) Operationen auf einem Monoprozessorsystem

P(n): Anzahl der auszuführenden (Einheits-) Operationen auf einem Multiprozessorsystem mit n Prozessoren

T(1): Ausführungszeit auf einem Monoprozessorsystem in Schritten

T(n): Ausführungszeit auf einem Multiprozessorsystem mit n Prozessoren in Schritten

T(1)=P(1) , da in einem Monoprozessorsystem jede (Einheits-) Operation in genau einem Schritt ausgeführt werden kann.

T(n)£ P(n) , da in einem Multiprozessorsystem mit n Prozessoren (n ³ 2) in einem Schritt mehr als eine (Einheits-) Operation ausgeführt werden kann.

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

Beschleunigung S(n) (Leistungssteigerung, Speed-up):

Lee'sches Prinzip: 1 £ S(n) £ n (maximal lineare Steigerung (mit den Prozessoren))

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

Effizienz E(n):

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

Es gilt:

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

Mehraufwand R(n) für die Parallelisierung (Redundanz):

Nach Lee gilt: 1 £ R(n)

$$I(n) = \frac{P(n)}{T(n)}$$

Parallelindex I(n):

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

Auslastung U(n):

Beispiel Seite 162!!

$$T(n) = T(1) \cdot \frac{1-a}{n} + T(1) \cdot a$$

Amdahls Gesetz:

(a ist der Bruchteil des Programms, der nur sequentiell ausgeführt werden kann)

$$S(n) = \frac{T(1)}{T(n)} = \frac{T(1)}{T(1) \cdot \frac{1-a}{n} + T(1) \cdot a} = \frac{1}{\frac{1-a}{n} + a} = \frac{n}{(1-a) + n \cdot a}$$

Daraus folgt:

$$S(n) \leq \frac{1}{a}$$

Somit besagt Amdahls Gesetz:

Weitere Effekte bei Multiprozessorsystemen:

- Verwaltungsaufwand (overhead), der mit der Zahl der zu verwaltenden Prozessoren ansteigt
- Möglichkeit der Systemverklemmungen (deadlocks)
- Möglichkeit von Sättigungserscheinungen, die durch Systemengpässe (bottlenecks) verursacht werden

1. *Speichergekoppelte Multiprozessorsysteme*

MESI-Cache-Kohärenzprotokoll funktioniert in Zusammenhang mit dem Bus-Schnüffeln; jedem Cache-Block wird einer der folgenden Zustände zugeordnet:

- Exclusive Modified: Der Cache-Block wurde durch einen Schreibzugriff geändert und befindet sich ausschließlich in diesem Cache
- Exclusive Unmodified: Der Cache-Block wurde für einen Lesezugriff übertragen und befindet sich nur in

diesem Cache
- Shared Unmodified: Kopien des Cache-Blocks befinden sich für Lesezugriffe in mehr als einem Cache
- Invalid: Der Cache-Block ist ungültig

Probleme treten auf, wenn Befehle nicht in der programmreihenfolge parallel verarbeitet werden; z.B. auch bei:

- **Nicht blockierenden Cache-Speichern** (non blocking cache): Im Falle eine Cache Fehlzugriffs können nachfolgende Befehle, die nicht denselben Cache-Block benötigen auf den Cache-Speicher zugreifen, ohne daß auf die Ausführung des den Fehlzugriff auslösenden Befehls gewartet werden muß

Atomischer Speicherbefehl: der neue Wert wird überall gleichzeitig wirksam.

Das **Konsistenzmodell** sagt etwas über die zu erwartende Ordnung der Speicherzugriffe durch parallel arbeitende Prozessoren aus.

Die **Programmordnung** bestimmt, in welcher Reihenfolge die Befehle (insbesondere die Lade-/Speicherbefehle, sowie Synchronisationsbefehle) von den parallel arbeitenden Prozessoren ausgeführt werden.

Die **Speicherzugriffsordnung** regelt die Reihenfolge, in der Lade-/Speicher- und Synchronisationsbefehle bei dem gemeinsamen Speicher oder den einzelnen Speichermodulen ankommen.

Definition einer Variablen: Schreiben ihres Wertes

Benutzung von Variablen: Lesen ihres Wertes

Ein Multiprozessorsystem ist **sequentiell konsistent**, wenn das Ergebnis einer beliebigen Berechnung dasselbe ist, als wenn die Operation aller Prozessoren auf einem Monoprozessorsystem in einer sequentiellen Ordnung ausgeführt würden. Dabei ist die Ordnung der Operationen der Prozessoren die des jeweiligen Programms.

Schwache Konsistenz: dieKonsistenz des Speicherzugriffs ist nicht mehr zu allen Zeiten gewährleistet, sondern nur zu bestimmten, vom Programmierer in das Programm eingesetzten Synchronisationspunkten; drei Bedingungen müssen erfüllt sein:

- Vor Schreib-/Lesezugriff eines anderen Prozessors müssen alle vorhergehenden Synchronisationspunkte erreicht worden sein.
- Vor der Synchronisation bzgl. eines Prozessors müssen alle vorhergehenden Schreib- und Lesezugriffe ausgeführt worden sein
- Die Synchronisationspunkte müssen sequentiell konsistent sein

Speichergekoppelte Multiprozessorsysteme mit globalem Speicher sind auf ca. 30 Prozessoren beschränkt.

1. *Nachrichtengekoppelte Multiprozessoren*

Skalierbarkeit unbegrenzt

Prozessorknoten über Punkt-zu-Punkt-Verbindungen gekoppelt; z.B. Hyperkubus.

Whormhole Routing (Wurmloch-Wegfindung): eine Nachricht wird in eine Kette von kleinen Teilblöcken zerlegt; der erste Block enthält die Empfängeradresse und bestimmt den Weg, alle anderen folgen ihm.

Vorteile: der zweidimensionalen Gitterverbindung (Intel Paragon-XP/S-Rechner) gegenüber beispielsweise der Hyperkubusstruktur:

- Beliebige Skalierbarkeit, da unabhängig von der Größe des Systems für jeden Knoten nur 4 Verbindungskanäle
- Hohe Wirtschaftlichkeit (Kosten durch Zahl der Verbindungskanäle bestimmt)
- Günstiger Aufbau auf Platinen möglich

Distributed-Shared-Memory-Multiprozessoren:

Zwei verschiedene Typen beim Zugriff auf entfernte Speicher:

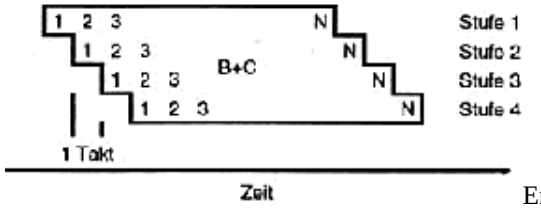
- Der Zugriff geschieht in einer für das Maschinenprogramm transparenten Weise
- Der Zugriff geschieht durch explizite Befehle, die nur dem Speicherzugriff dienen

MPP Systeme von Cray (erstmal weggelassen) [...]

1. Vektorrechner

Unterteilung der Gleitpunktaddition in vier Schritte (beispielsweise):

- Exponenten vergleichen
- Mantisse verschieben
- Ausgerichtete Mantisse addieren



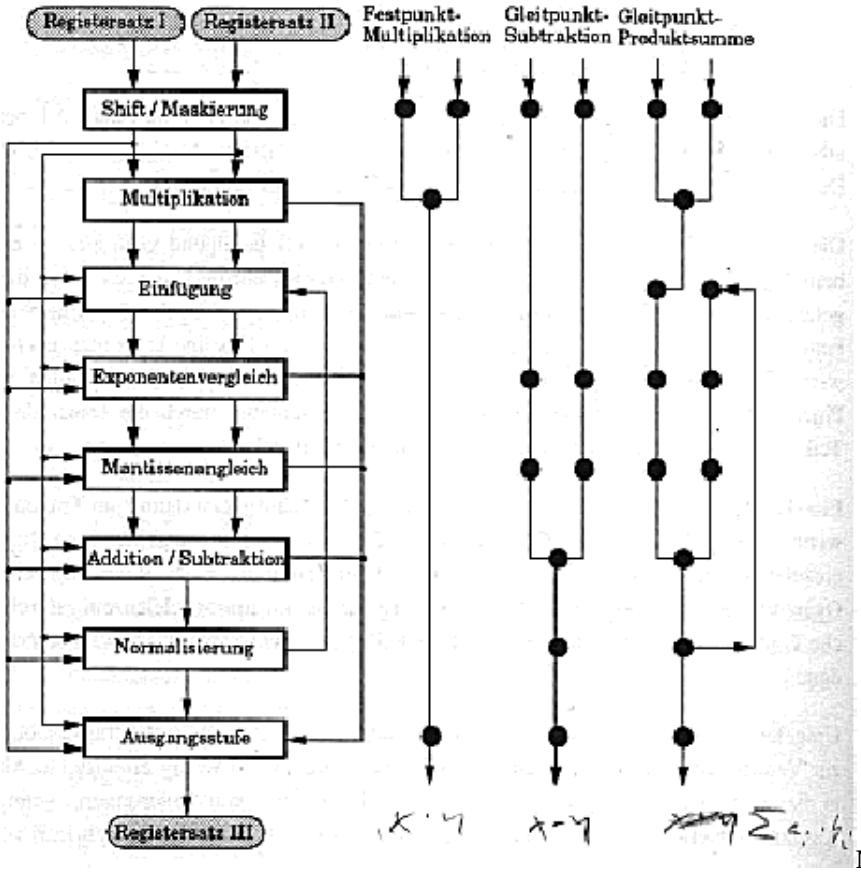
- Ergebnis normalisieren

Abb.7: Pipelineverarbeitung von B(J)+C(J), J=1,2,...,N

Beliebte Prüfungsfrage: Was unterscheidet eine Gleitpunkteinheit von einer Vektoreinheit (=Satz von Vektor-Pipelines (= pipelineartige Funktionseinheit zur Verarbeitung von Gleitpunktzahlen))?

Antwort: Anzahl der Befehle (Vektoreinheit = 1 Befehl für Abb.7)

Der große Vorteil des Vektorrechners liegt darin, daß die in Abb.7 dargestellte Pipelineverarbeitung in **einem** Vektorbefehl für **zwei** Felder von Gleitpunktzahlen durchgeführt wird. Dabei entfallen die bei skalaren Prozessoren nötigen Adreßrechnungen.



Multifunktions-Pipeline

Abb.8: Multifunktions-Pipeline

Wie in Abbildung 8 zu sehen ist, wird eine höhere Stufenzahl, als für die Operation notwendig ist, eingeführt, die jeweils nicht benötigten Stufen werden übersprungen.

Nachteil: aufwendige Hardware

Spezialisierte Pipeline

Zur Durchführung spezieller Funktionen werden mehrere unabhängige Pipelines benutzt; jede Pipeline entspricht dabei einem Rechenwerk für sich.

Verkettung (Chaining) der Zugriffe und Verknüpfung zu einer längeren Pipeline möglich.

Verkettete Bearbeitung kann erreicht werden durch:

- Spezialisierung der Verarbeitung
- Verdeckung der Steuerung
- Topologie der Verbindungsstruktur entsprechend der Problemstellung
- Abstimmung der beteiligten Ressourcen bezüglich der Zeitbedingungen und Datenformate

Register-zu-Register-Architekturen (vector-register-processors):

> Operanden werden aus Registern geholt und Ergebnisse in Register geschrieben (hat sich heute mehr durchgesetzt)

Speicher-zu-Speicher-Architekturen (memory-memory vector processors):

Operanden werden aus dem Speicher geholt und Ergebnisse in den Speicher geschrieben.

Möglichkeiten der Parallelarbeit bei einem Vektorrechner:

- Vektor-Pipeline-Parallelität
  durch die Stufenzahl der Vektorpipeline gegeben
- Mehrere Vektorpipelines in einer Vektoreinheit
  Verkettung möglich
- Vervielfachung der Pipelines
  mehrere Operanden können gleichzeitig verarbeitet werden
- Mehrere Vektoreinheiten
  arbeiten nach der Art eines speichergekoppelten Multiprozessors parallel zueinander

1. *Leistungsfähigkeit von Vektorrechnern*

Im Idealfall kann ein Pipeline-Prozessor mit k Stufen n Verarbeitungsaufträge in $T_k = k+(n-1)$ Taktzyklen ausführen (k Taktzyklen zum Auffüllen; erster Auftrag, (n-1) restliche Aufträge)

Ohne Pipeline-Struktur: $T_1 = n*k$ (Bei Berücksichtigung der Startzeit: s einfach dazuaddieren)

$$S_k = \frac{T_1}{T_k} = \frac{n \cdot k}{k + (n-1)}$$

Beschleunigung S_k (Leistungssteigerung, Speed-up):

Somit: S_k à k für n à ¥

Effizienz Ek (Verhältnis zw. Tatsächlicher Leistungssteigerung und max. möglicher $S_{max}=k$):

$$E_k = \frac{S_k}{k} = \frac{n \cdot k}{k \cdot (k + (n-1))} = \frac{n}{k + (n-1)}$$

Damit: E_k à 1 für n à ¥

Durchsatz D_k (gibt an, wie viele Verarbeitungsaufträge n in einem Beobachtungszeitraum T_k*t tatsächlich ausgeführt werden):

$$D_k = \frac{E_k}{\tau} \quad D_k = \frac{n}{T_k \tau} = \frac{n}{\tau \cdot (k + (n-1))} = \frac{n}{\tau \cdot k + \tau \cdot (n-1)}$$

Damit gilt:

$$D_k \to \frac{1}{\tau}$$

Es gilt als Grenzwert: für n à ¥

$$D_{max} = \frac{1}{\tau} = f$$

Maximal möglicher Durchsatz bei Vektorlänge n=¥ :

$n_{1/2}$: **Half-performance length** = Vektorlänge, bei der die Hälfte des maximalen Durchsatzes erreicht wird.

Maximum-Performance: D_{max} (gemessen in MFLOPS) unter Berücksichtigung der Startzeit s (Dekodierung der Befehle etc.):

$T'_k = k + (n-1) + s$ Taktzyklen.

Trade-off-Punkt: Vektorlänge n_t, ab der sich der Einsatz eines Vektorprozessors lohnt (bei Berücksichtigung der Startzeit s).

Es gilt dort $T_1(n_t) = T'_k(n_t)$;

$n < n_t : T_1(n_t) < T'_k(n_t)$;

$n > n_t : T_1(n_t) > T'_k(n_t)$

$$\Rightarrow n_t = 1 + \frac{s}{k-1}$$

Berechnung des trade-off Punktes aus obigen Formeln: ...

Maßnahmen zur Eindämmung der schlechten Startzeiteinflüsse:

- Startzeit einer Pipeline wird mit anderen nützlichen Aktionen überlappt, indem eine Pipeline im Rechenwerk mit einer Pipeline im Steuerwerk kombiniert wird (Befehls-Pipelining)
- Bei der Verkettung von Pipelines fallen die Startzeiten der einzelnen Pipelines nicht mehr ins Gewicht

{Cray (front end), Chunk Chaining, Minimierung des Overheads} erstmal weggelassen. [...]

Feldrechner erstmal weggelassen [...]

1. Zuverlässigkeit von Rechnersystemen

Zuverlässigkeit (dependability): Fähigkeit eines Systems, während einer vorgegebenen Zeitdauer bei zulässigen Betriebsbedingungen die spezifizierte Funktion zu erbringen.

Fehlertoleranz (fault tolerance): Fähigkeit eines Systems, auch mit einer begrenzten Anzahl fehlerhafter Subsysteme die spezifizierte Funktion (bzw. den geforderten Dienst) zu erbringen.

Sicherheit (safety (nicht security!!)): das Nichtvorhandensein einer Gefahr für Mensch oder Sachwerte ... Rest klar

Wirkungskette: Fehlerursache à Fehlerzustand à Ausfall

Verschiedene Fehlerursachen:

- **Entwurfsfehler**
- **Herstellungsfehler**
- **Betriebsfehler**
- **Störungsbedingte Fehler**
- **Verschleißfehler**
- **Zufällige physikalische Fehler**
- **Bedienungsfehler**
- **Wartungsfehler**

Verschiedene Fehler nach Entstehungsort:

- **Hardware**
- **Software**

Verschiedene Fehler nach Dauer:

- **Permanente Fehler**
- **Temporäre Fehler**

Struktur-Funktions-Modell: gerichteter Graph, dessen Knoten die Komponenten und dessen Kanten die Funktionen eines Systems repräsentieren. Eine gerichtete Kante von der Komponente K_i zur Komponente K_j bedeutet, daß K_i eine Funktion erbringt, die von K_j benutzt werden kann.

Schichtenmodell: die Komponenten werden in disjunkte Schichten partitioniert, für die es

eine Totalordnung gibt. Funktionszuordnungen sind nur von den niedrigeren Schichten an höhere Schichten (eine Halbordnung) und innerhalb der Schichten zulässig.

Fehlermodell: (fault model): beschreibt die möglichen Fehlerzustände eines Systems, z.B. durch die Angabe der Komponentenmengen, die zugleich von einer Fehlerursache betroffen sein können (Einzelfehlerbereiche), und durch Angabe des möglichen fehlerhaften Verhaltens dieser Komponenten (Fehlfunktion).

Binäres Fehlermodell: logischer Ausdruck, jede "Variable" steht für ein Subsystem und kann die Zustände "wahr"=OK oder "falsch"= nicht OK annehmen.

Nichtredundantes Fehlermodell: nur fehlerfrei, wenn alle Komponenten fehlerfrei

Analog zum binären Fehlersystem, jedoch graphisch: Zuverlässigkeitsblockdiagramm und Fehlerbaum.

Fehlerbereich B: ein Fehlerbereich (B Teilmenge des Systems) ist eine Menge von Komponenten, die zugleich fehlerhaft sein können, ohne daß das System insgesamt fehlerhaft wird.

Einzelfehlerbereich: Menge aller Komponenten, die genau den gleichen Fehlerbereichen angehören.

Perfektionskern: Komplement der Vereinigung aller Fehlerbereiche (= muß immer funktionieren, damit System insgesamt fehlerfrei läuft)

Ausfälle:

- **Teilausfall: ein Teil geht nicht, Rest läuft fehlerfrei**
- **Unterlassungsausfall: keine oder richtige Ergebnisse**
- **Anhalteausfall: keine Ergebnisse mehr - Rechnerabsturz**
- **Haftausfall: ständig das gleiche Ergebnis**
- **Binärstellenausfall: verfälschte Binärstelle(n)**

Fail-stop-System: nur Anhalteausfälle

Fail-silent-System: nur Unterlassungsausfälle

Fail-safe-System: nur unkritische Ausfälle

Fehlereingrenzungen (erstmal weggelassen) [...]

Die Menge der zu tolerierenden Fehler gibt an, welche der vorgesehenen Fehler zu tolerieren sind.

Zeitredundanz t_R: Intervall, in dem kein zweiter Fehler auftauchen darf.

Fehlerbehandlungsdauer B: tatsächliche Zeit zur Fehlerbehandlung (muß kleiner als die Zeitredundanz sein)

Lebensdauer L; Sicherheitsdauer D

$f_L(t), f_B(t), f_D(t)$

Davon die Dichten:

$F_L(t), F_B(t), F_D(t)$

Die dazu korrespondierenden Verteilungsfunktionen:

$$F_X(t) := \int_0^t f_X(s)ds$$

d.h.:

für X=L, B und D

1. ***Zuverlässigkeitskenngrößen***
   1. **Zuverlässigkeit**

Fehlerwahrscheinlichkeit $F_L(t)$: Wahrscheinlichkeit, daß ein zu Beginn fehlerfreies System im Zeitintervall [0,t] fehlerhaft wird.

$$E(L) = \int_0^\infty R(t)dt$$

Überlebenswahrscheinlichkeit R(t): $R(t) = 1 - F_L(t)$

Mittlere Lebensdauer:

$$z(t) = \frac{f_L(t)}{R(t)}$$

Ausfallrate:

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

Verfügbarkeit:

2. $es\ gilt : F_D(t) \leq F_L(t)$ Sicherheit

Gefährdungswahrscheinlichkeit:

$$S(t) := 1 - F_D(t)$$

Sicherheitswahrscheinlichkeit:

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

Mittlere Sicherheitsdauer E(D):

$$Formel\ für\ X\ von Y\ System (X < Y): \sum_{i=X}^{Y} \binom{Y}{i} R_S(t)^i (1 - R_S(t))^{Y-i}$$ $$\mu(S) = \sum_{(K_1,\ldots,K_n) \in f^{-1}(wahr)} \mu\left(\bigcap_{i=1}^{n} K_i\right)$$ **Funktions**

j : S È {S} à [0,1]: Oberbegriff für Überlebenswahrscheinlichkeit R und Verfügbarkeit V;

$$\Phi_{S_1 \to S_2} = \frac{\mu(\neg S_1)}{\mu(\neg S_2)} = \frac{1 - \mu(S_1)}{1 - \mu(S_2)}$$

Zuverlässigkeitsverbesserung (Zuverlässigkeitsverbesserungsfaktor):

j (AvB) = j (A)+ j (B) - j (A) j (B)

Wenn mehr als 2 Komponenten è Untergruppen bilden: gerade mit MINUS, ungerade mit PLUS

(IMMER von E nach A gehen (!!!))

Redundanz: mehr technische Mittel, als nötig.

Dynamische Redundanz: Vorhandensein redundanter Mittel, die erst nach dem Auftreten eines Fehlers aktiv werden, um die Nutzfunktion zu erbringen.

Statische Redundanz: Vorhandensein redundanter Mittel, die während des gesamten Einsatzzeitraumes die gleiche Funktion erbringen.

Hamming und RAID ertsmal weggelassen [...]

Redundante verteilte Systeme:

- **Nichtredundantes Einfachsystem**
- **Einfachsystem mit ungenutzter Redundanz (fremdgenutzte Redundanz analog)**
- **Nichtredundantes Mehrfachsystem**
- **Mehrfachsystem aus verschiedenartigen Komponenten ungenutzter Redundanz**
- **(fremdgenutzte Redundanz analog)**
- **Mehrfachsysteme aus gleichartigen Komponenten mit gegenseitiger Redundanz**
- **Dynamsich redundantes System mit begrenzter Fehlererfassung**
- **Statisch redundantes System**

- **Statisch redundantes System mit Zuverlässigkeitsengpaß**

Verbesserung der Zuverlässigkeit: (3 Alternativen)

1. **Verbesserung der Komponenten**
2. **Zusätzliche Komponenten**
3. **Zusätzliche Subsysteme, insbesondere zusätzliche Rechner**

ALLGEMEINER NACHTRAG:

Bei Registerfenstern:

Anzahl ausrechnen: RF = (#Register (nicht globale!))/(#ins + # Outs)

Nur CALL und JMP zählen bei Bestimun der benötigten Registerfenster (bei BRA kein Fensterwechsel)

Wenn bei Vektorrechnern Leistungen im Bezug auf andere abstrakte Systeme verlangt weren: einfach mit min(a,b), max(a,b) etc. arbeiten (Sc=Sa+Sb), (Emaxc=Emaxa=Emaxb=1)

Stufen einer allg. Vektorpipeline:

1. **Exponent vergleichen**
2. **Mantisse verschieben**
3. **Exponent addieren**
4. **Ausger. Mantisse addieren**
5. **Mantisse multiplizieren 1.**
6. **Mantisse multiplizieren 2.**
7. **Ausger. Mantisse subtrahieren**
8. **Ergebnis normalisieren**

Bei Addition entfallen (3.,5.,6.,7.)

Bei Subtraktion entfallen (3.,4.,5.,6.)

Bei Multiplikation entfallen: (1.,2.,4.,7.)

Universelles Netzwerk mit k Kreuzschaltern und n Ein-/Ausgängen:

Anzahl der Zustände > Anzahl möglicher Permutationen

Zustände = 2^k

Mögliche Permutationen. n!

k^3 ld(n!)

Beispiel: n=3 à n!=6 à k=3 à 8 Kreuzschalter

Bei SUPERPIPELINING können Compiler einfacher und effizienter gestaltet werden

Physikalisch bzw. virtuell adressierter Cache:

Bei physikalisch adressiertem Cache sind nach einem Seitenfehler der Cache-Lines nicht mehr aktuell. Beim virtuell adressiertem Cache ist dies nicht der Fall.

Funktionswahrscheinlichkeiten:

Bei Ableitung von: $2^{-l\,t}$ Trick: $2^{-l\,t} = e^{-l\,t\,\ln 2}$ Abl.: e... mal Innere

Bei Stammfunktionsfindung von e... : e.... / Innere Ableitung

Bei Statischen Verbindungsnetzen:

Durchsatz: Was kann maximal zu einem Zeitpunkt verschickt weren (=#Verbindungen)

Restlichen Eigenschaften schon oben erwähnt.

Zuverlässigkeitsblockdiagramm:

***Fehlerbaum*: S negieren und aufzeichnen (mit negativen Komponenten) S oben nicht**

vergessen!!!

Bei X aus Y System: Alle Möglichkeiten in Zuverlässigkeitsblockdiagramm reinnehmen!!!

Ursache für "Von-Neumann-Flaschenhals":

Sequentieller Zugriff auf Befehle und Daten über 1 Register

Aufhebung: Harvard-Architektur, Zusatzregister, Befehlspipelining, Speicherhierarchien, Vektor-Pipelining, Vervielfachung der Datenprozessoren etc.

Bei RISC:

Pipeline aufmalen:

Bei Datenkonflikten: Op-holen nach Erg-speichern schieben = #NOOPs

Bei Kontrollkonflikten: 1. Stufe nach Erg-speichern schieben = #NOOPs