Rechnerstrukturen

Vorlesung im Sommersemester 2007

Prof. Dr. Wolfgang Karl

Universität Karlsruhe (TH)

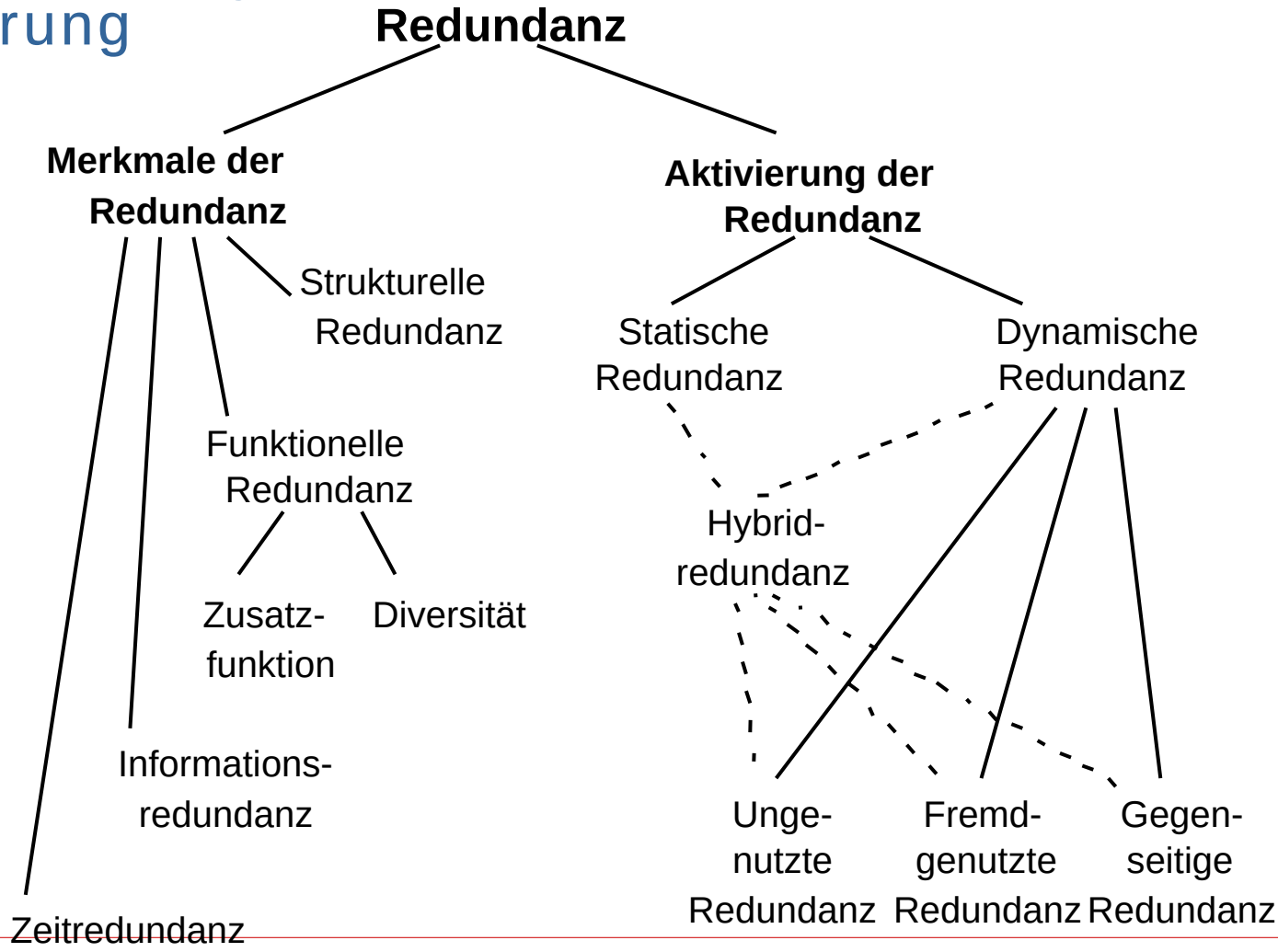
Fakultät für Informatik

Institut für Technische Informatik

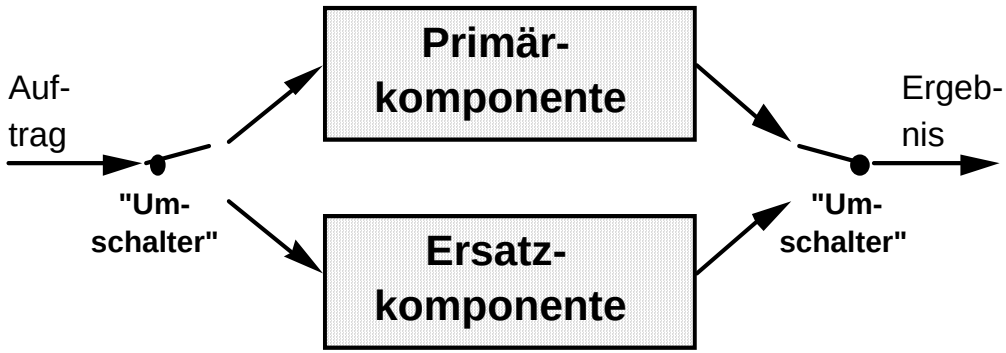
- **Kapitel 4: Fehlertoleranz, Zuverlässigkeit**

4.1: Grundlagen

- Redundanz
  - Unterscheidung nach ihren Merkmalen und ihrer Aktivierung

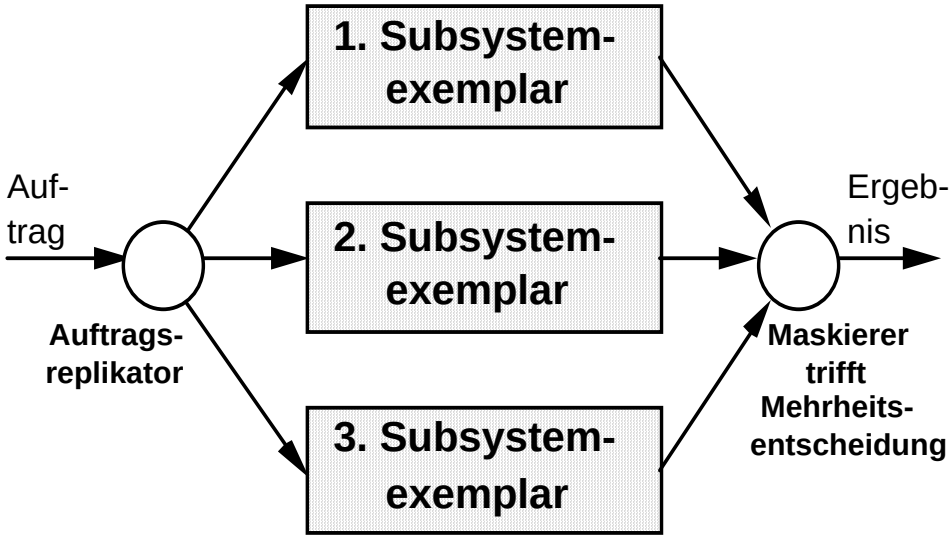


- Dynamische Redundanz (dynamic redundancy)
  - bezeichnet das Vorhandensein von redundanten Mitteln, die erst nach Auftreten eines Fehlers aktiviert werden, um eine ausgefallene Nutzfunktion zu erbringen.
  - Typisch für dynamische strukturelle Redundanz ist die Unterscheidung in Primär- und Ersatzkomponenten (bzw. Sekundär- oder Reservekomponenten).
  - Grundstruktur eines dynamisch strukturell redundanten Systems

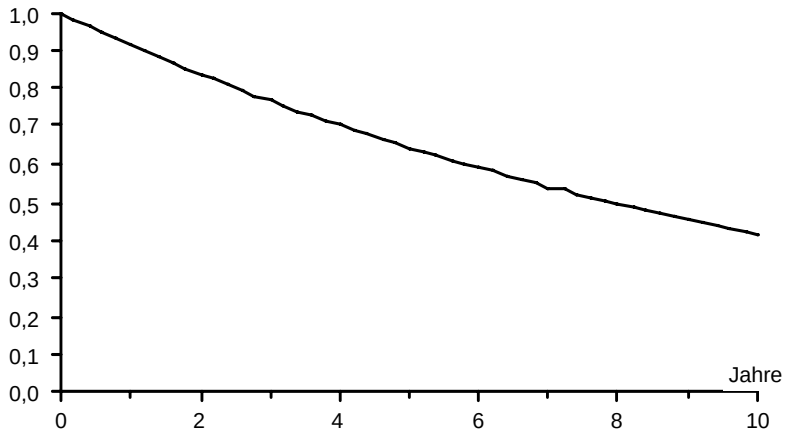


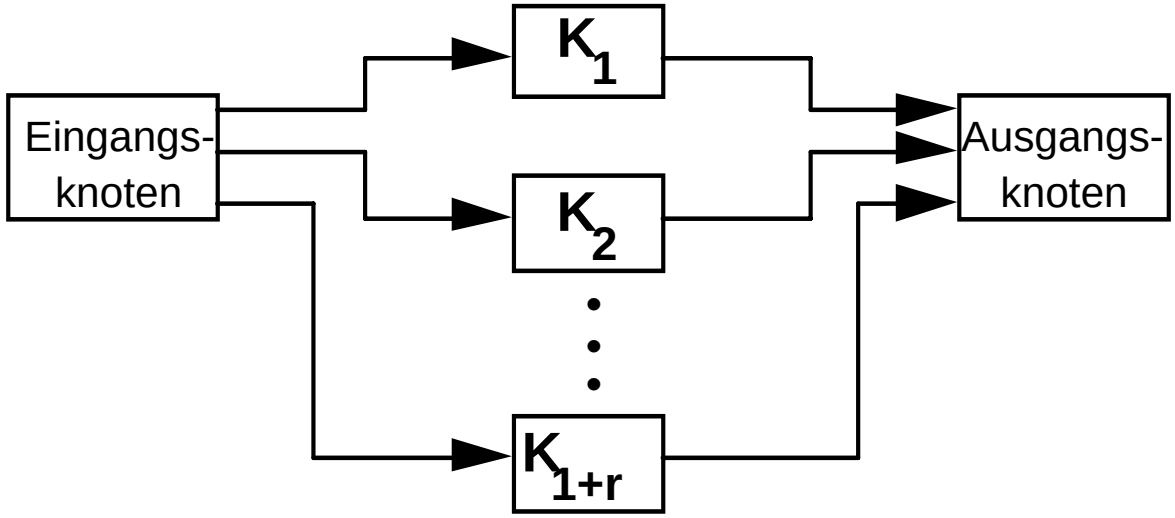
- Dynamische Redundanz (dynamic redundancy)
  - Bevor Ersatzkomponenten aktiviert werden, lassen diese sich auf eine der folgenden Arten verwenden:
  - Ungenutzte Redundanz
    - Ersatzkomponenten führen keine sonstigen Funktionen aus und bleiben bis zur fehlerbedingten Aktivierung passiv.
  - fremdgenutzte Redundanz:
    - Ersatzkomponenten erbringen nur Funktionen, die nicht zum betreffenden Subsystem gehören und im Fehlerfall bei niedrigerer Priorisierung ggf. verdrängt werden.
  - gegenseitige Redundanz:
    - Ersatzkomponenten erbringen die von einer anderen Komponente zu unterstützenden Funktionen, die Komponenten stehen sich gegenseitig als Reserve zur Verfügung.
      Dies ermöglicht einen abgestuften Leistungsabfall (graceful degradation).

- Statische Redundanz (*static redundancy*)
  - bezeichnet das Vorhandensein von redundanten Mitteln, die während des gesamten Einsatzzeitraums die gleiche Nutzfunktion erbringen.
  - Beispiel der statischen strukturellen Redundanz: *n*-von-*m*-System
  - 2-von-3-System:



- Verbesserung der Zuverlässigkeit durch Redundanz
  - Nichtredundantes Einfachsystem: S1=K1
  - Bei konstanter Ausfallrate beschreibt man die Zeitabhängigkeit der Funktionswahrscheinlichkeit φ (S_1, t) durch eine Exponentialverteilung
    - mit $z(t)=\lambda$, $\varphi(S_1, t) = e^{-\lambda \cdot t}$.
  - Beispiel:
    - Funktionswahrscheinlichkeit φ (S_1, t) mit $\lambda=10^{-5/h}$





Systemfunktion

$$S_{1+r} = K_1 \vee \ldots \vee K_{1+r}$$

Funktionswahrscheinlichkeit

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

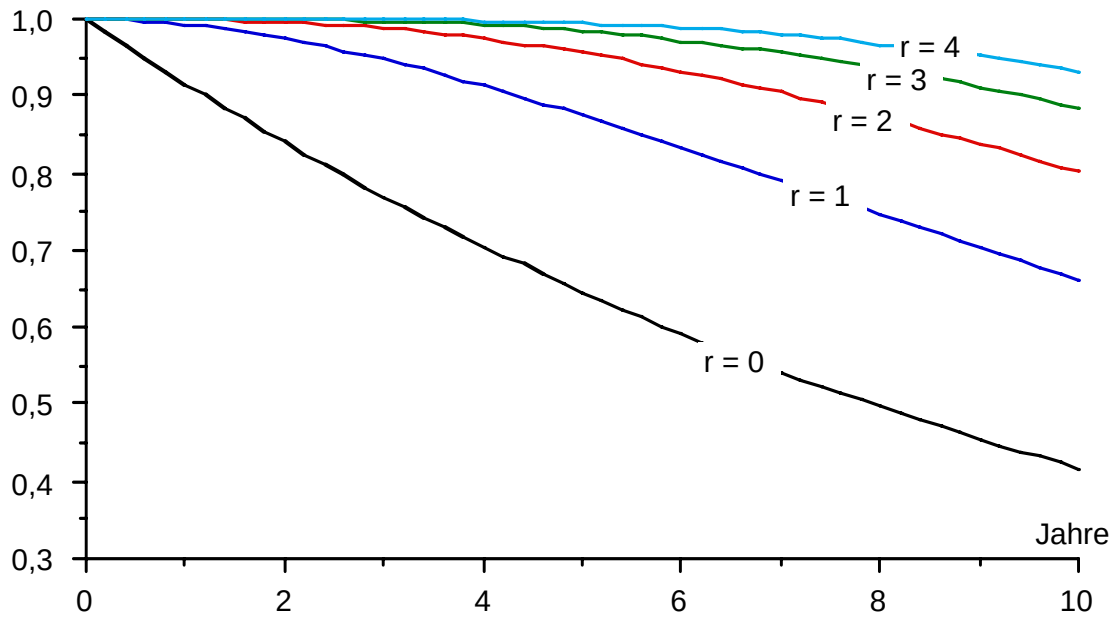
gleiche konstante Ausfallrate λ

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

Zuverlässigkeitsverbesserung

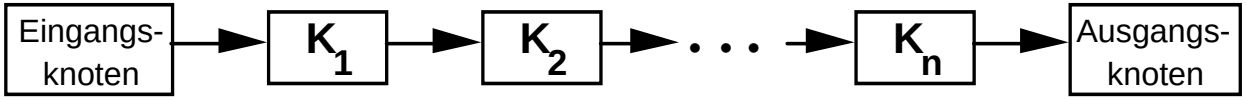
$$\Phi_{S1 \to S1+r} = (1 - e^{-\lambda \cdot t})^{-r}.$$

Funktionswahrscheinlichkeit für Parallelsystem



Annahme einer Komponentenausfallrate von $\lambda = 10^{-5}/\mathrm{h}$

Seriensystem (Nichtredundantes Mehrfachsystem)



Seriensystem

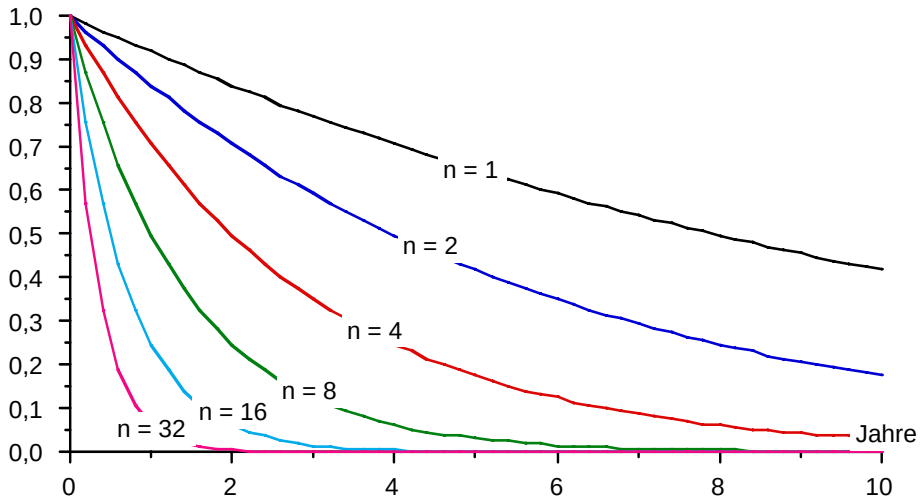
$$S_n = K_1 \wedge \ldots \wedge K_n$$

Zuverlässigkeit

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

Funktionswahrschein-lichkeit $\varphi(S_n, t)$

für $\lambda = 10^{-5}/h$



Ist die Fehlererfassung zu gering oder verbieten sich wiederholte Berechnungen wegen den geforderten maximalen Antwortzeiten, so kann statische Redundanz eingesetzt werden.

Dabei führen mehrere Komponenten die gleiche Berechnung aus, um anschließend die errechneten Ergebnisse zu vergleichen und ein mehrheitliches auszuwählen.

Bis zu f fehlerhafte Komponenten können überstimmt werden, wenn mindestens n=f+1 fehlerfreie, insgesamt also m=2·f+1 Komponenten vorhanden sind.

$$S_{m\ von\ m} = \bigvee_{1 \le i_1 < \ldots < i_n \le m} K_{i_1} \wedge \ldots \wedge K_{i_n}$$

- **1. Alternative: Verbesserung der Komponenten**
  - **+einfacher Ansatz zur Verbesserung**
  - **+bis zu einer gegebenen Grenze kostengünstig**
  - **–ab dieser Grenze steigen die Kosten überproportional**
  - **–Lösung oft nicht leistungsfähig und zuverlässig zugleich**

- **2. Alternative: Zusätzliche Komponenten**
  - **–bei Verdoppelung oder Vervierfachung hoher Aufwand**
  - **+Prüfzeichen sind ein effizientes Mittel gegen spezielle Fehler**
  - **– Ansatz ist unflexibel, da meist voller Zusatzaufwand notwendig**

- **3. Alternative: Zusätzliche Subsysteme (zusätzliche Rechner)**
  - **+alle Rechner gleich, keine Spezialrechner**
  - **+flexible Lastverteilung (und Umverteilung bei Fehler) möglich**
  - **–überproportionale Kosten**
  - **–Aufwand zur Herstellung der aktuellen Zustandsinformation**
  - **–Aufwand zur Vermeidung von Inkonsistenzen**

- **Kapitel 5: Vektorprozessoren**

5.1: Grundlagen

- Wie arbeiten Vektorprozessoren?
  - Ein Beispiel:
    - Y = a * X + Y,
      - wobei X und Y Vektoren sind und a eine Konstante ist.
      - Bildet die innere Schleife des Linpack-Benchmarks und wird auch als SAXPY (Single Precision a x X plus Y) bzw. DAXPY (Double Precision a x X plus Y) bezeichnet.
      - Für alle i, i = Anzahl der Elemente des Vektors X und des Vektors Y, ergibt sich der neue Wert für Y[i] aus der Multiplikation von a mit X[i] und der Addition des Zwischenergebnisses mit Y[i]

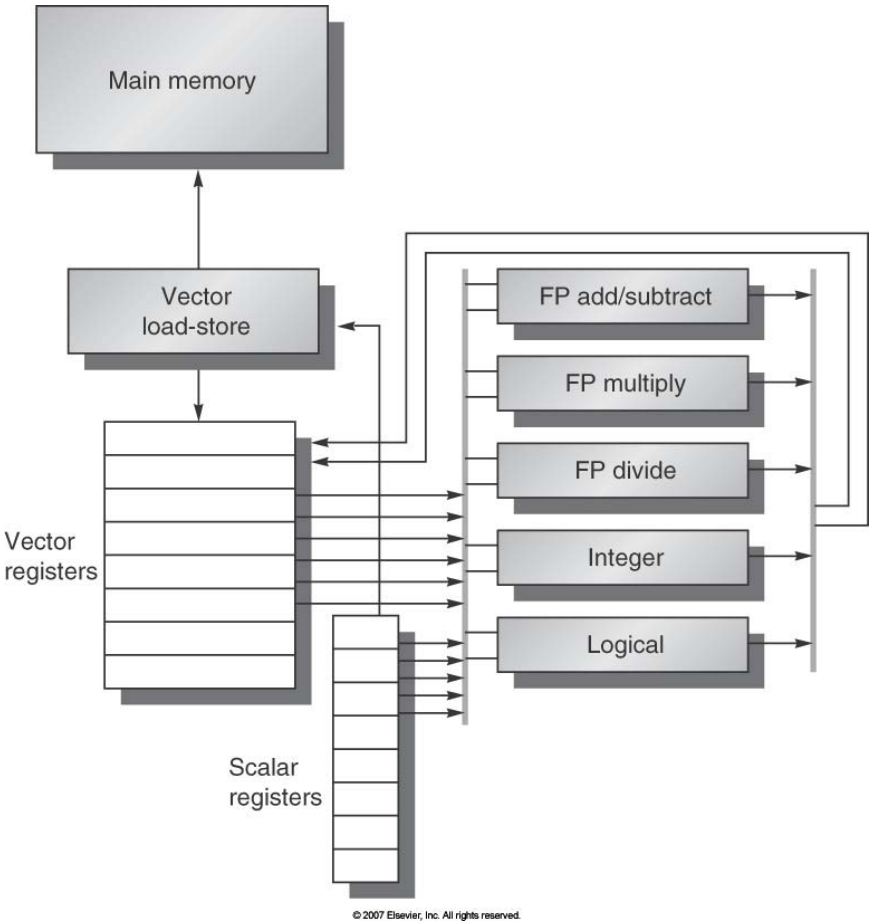
- Wie arbeiten Vektorprozessoren?
  - Ein Beispiel:
    - Y = a * X + Y,
  - In MIPS-Notation

```
                                          ;in Rx bzw. Ry Startadresse von X,Y
        L.D       F0,a                    ;load scalar a
        DADDIU    R4,Rx,#512              ;last address to load
Loop:   L.D       F2,0(Rx)                ;load X(i)
        MUL.D     F2,F2,F0                ;a × X(i)
        L.D       F4,0(Ry)                ;load Y(i)
        ADD.D     F4,F4,F2                ;a × X(i) + Y(i)
        S.D       0(Ry),F4                ;store into Y(i)
        DADDIU    Rx,Rx,#8                ;increment index to X
        DADDIU    Ry,Ry,#8                ;increment index to Y
        DSUBU     R20,R4,Rx               ;compute bound
        BNEZ      R20,Loop                ;check if done
```

- Wie arbeiten Vektorprozessoren?
  - Ein Beispiel:
    - Y = a * X + Y
  - Analyse:
    - Bei einer Länge der Vektoren von 64 Elementen wird die Schleife ungefähr 600 Mal durchlaufen.
    - Hoher Aufwand:
      - In einem Schleifendurchlauf werden jeweils die Elemente von X und Y addiert und nach der Berechnung wird das Ergebnis gespeichert. Die Adressen werden aktualisiert und das Abbruchkriterium wird geprüft.
      - Beachtung von Konflikten aufgrund von Datenabhängigkeiten
      - Beachtung von Multizyklus-Operationen
      - Pipeline-Stalls können durch Compiler-Optimierungen wie Loop-Unrolling und Software-Pipelining reduziert werden

- Wie arbeiten Vektorprozessoren?
  - Ein Beispiel:
    - Y = a * X + Y
  - Idee der Vektor-Prozessoren:
    - SIMD-Verarbeitung
    - Verarbeitung von Vektoren in einem Rechenwerk mit Pipeline-artig aufgebauten Funktionseinheiten
    - Bereitstellung von Vektoroperationen

- Wie arbeiten Vektorprozessoren?
  - Idee der Vektor-Prozessoren:

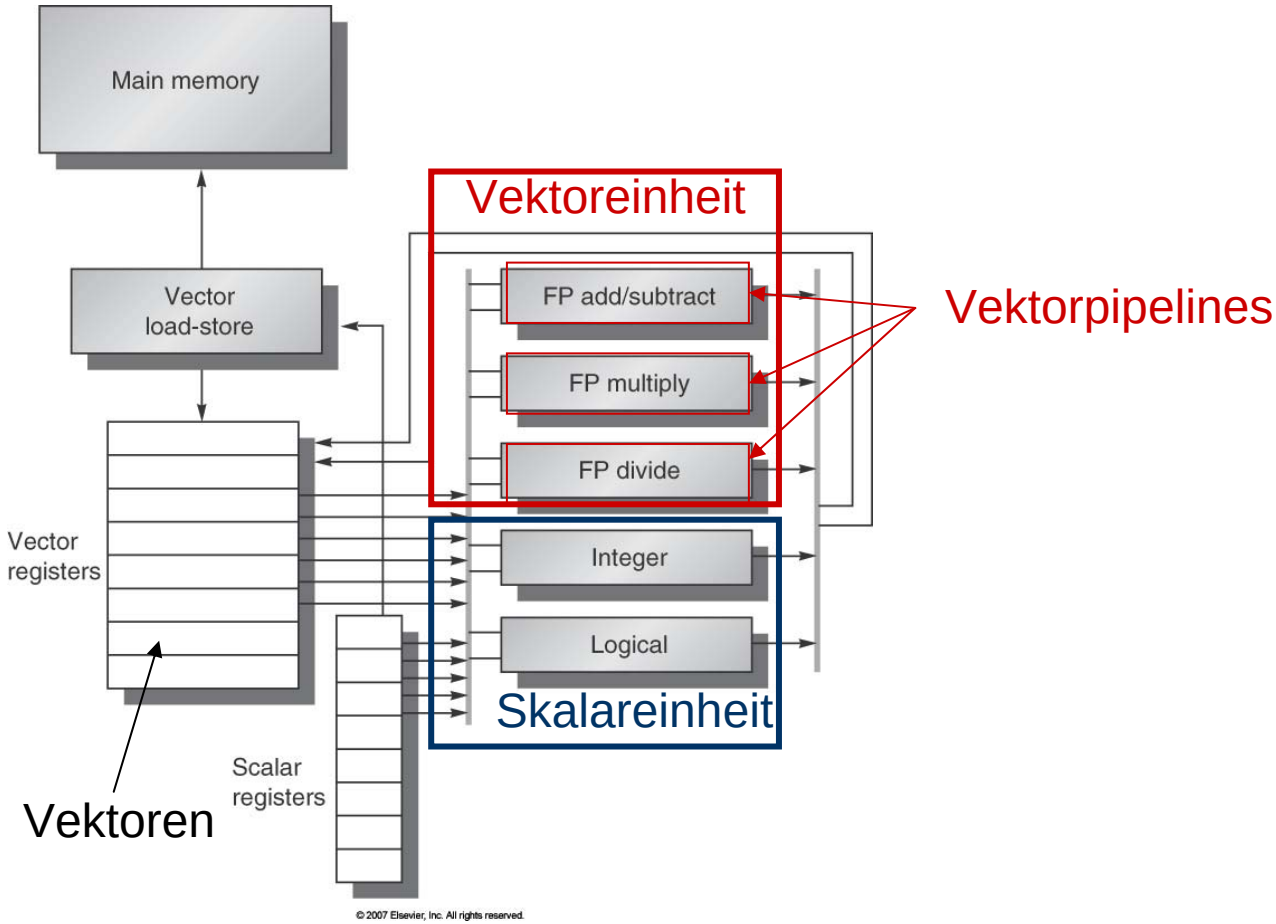


Beispielprogramm mit Vektoroperationen:

```
L.D      F0,a       ;load scalar a
LV       V1,Rx      ;load vector X
MULVS.D  V2,V1,F0   ;vector-scalar multiply
LV       V3,Ry      ;load vector Y
ADDV.D   V4,V2,V3   ;add
SV       Ry,V4      ;store the result
```

- Vektorprozessor:
  - Unter einem Vektorprozessor (Vektorrechner) versteht man einen Rechner mit pipelineartig aufgebautem/n Rechenwerk/en zur Verarbeitung von Arrays von Gleitpunktzahlen.
  - Vektor = Array (Feld) von Gleitpunktzahlen
  - Jeder Vektorrechner besitzt in seinem Rechenwerk einen Satz von Vektorpipelines. Dieser wird als Vektoreinheit bezeichnet.
  - Im Gegensatz zur Vektorverarbeitung wird die Verknüpfung einzelner Operanden als Skalarverarbeitung bezeichnet.
  - Ein Vektorrechner enthält neben der Vektoreinheit auch noch eine oder mehrere Skalareinheiten. Dort werden die skalaren Befehle ausgeführt, d.h. Befehle, die nicht auf ganze Vektoren angewendet werden sollen.
  - Die Vektoreinheit und die Skalareinheit(en) können parallel zueinander arbeiten, d.h. Vektorbefehle und Skalarbefehle können parallel ausgeführt werden.

- Vektorprozessoren
  - Beispiel



- Vektorprozessoren
  - Die Pipeline-Verarbeitung wird mit einem Vektorbefehl für zwei Felder von Gleitpunktzahlen durchgeführt.
  - Die bei den Gleitpunkteinheiten skalarer Prozessoren nötigen Adressrechnungen entfallen.

Beispielprogramm mit Vektoroperationen:

```
L.D       F0,a      ;load scalar a
LV        V1,Rx     ;load vector X
MULVS.D   V2,V1,F0  ;vector-scalar multiply
LV        V3,Ry     ;load vector Y
ADDV.D    V4,V2,V3  ;add
SV        Ry,V4     ;store the result
```

- Vektorprozessoren
  - Vektorbefehle

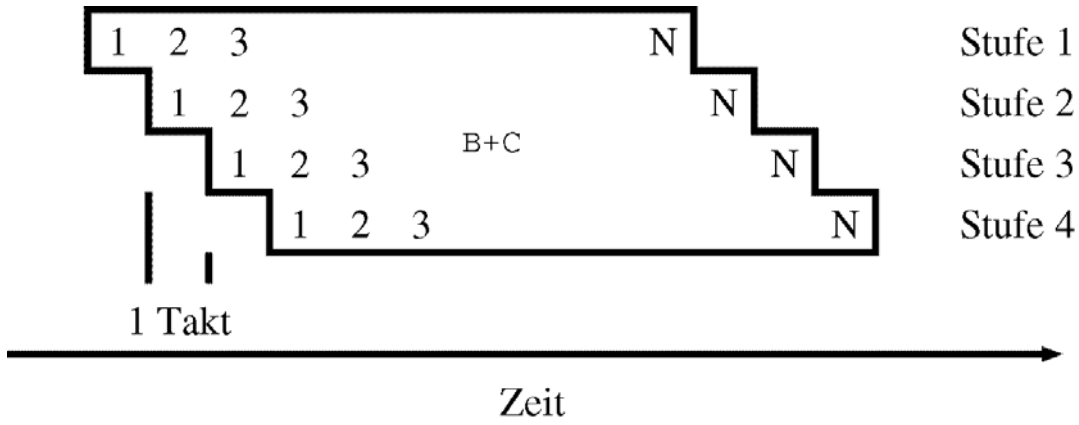
Instruktion	Operanden	Funktion
ADDV.D ADDVS.D	V1,V2,V3 V1,V2,F0	Add elements of V2 and V3, then put each result in V1. Add F0 to each element of V2 , then put each result in V1.
SUBV.D SUBVS.D SUBSV.D	V1,V2,V3 V1,V2,F0 V1,F0,V2	Subtract elements of V3 from V2, then put each result in V1. Subtract F0 from elements of V2, then put each result in V1. Subtract elements of V2 from F0, then put each result in V1.
MULV.D MULVS.D	V1,V2,V3 V1,V2,F0	Multiply elements of V2 and V3, then put each result in V1. Multiply each element of V2 by F0, then put each result in V1.
DIVV.D DIVVS.D DIVSV.D	V1,V2,V3 V1,V2,F0 V1,F0,V2	Divide elements of V2 by V3, then put each result in V1. Divide elements of V2 by F0, then put each result in V1. Divide F0 by elements of V2, then put each result in V1.
LV	V1,R1	Load vector register V1 from memory starting at address R1.
SV	R1,V1	Store vector register V1 into memory starting at address R1.
LVWS	V1,(R1,R2)	Load V1 from address at R1 with stride in R2, i.e., R1+i × R2.
SVWS	(R1,R2),V1	Store V1 from address at R1 with stride in R2, i.e., R1+i × R2.
LVI	V1,(R1+V2)	Load V1 with vector whose elements are at R1+V2(i), i.e., V2 is an index.

- Vektorprozessoren
  - Vektorbefehle

Instruktion	Operanden	Funktion
SVI	(R1+V2),V1	Store V1 to vector whose elements are at R1+V2(i), i.e., V2 is an index.
CVI	V1,R1	Create an index vector by storing the values 0, 1 × R1, 2 × R1,...,63 × R1 into V1.
S--V.D S--VS.D	V1,V2 V1,F0	Compare the elements (EQ, NE, GT, LT, GE, LE) in V1 and V2. If condition is true, put a 1 in the corresponding bit vector; otherwise put 0. Put resulting bit vector in vector-mask register (VM). The instruction S--VS.D performs the same compare but using a scalar value as one operand.
POP	R1,VM	Count the 1s in the vector-mask register and store count in R1.
CVM		Set the vector-mask register to all 1s.
MTC1 MFC1	VLR,R1 R1,VLR	Move contents of R1 to the vector-length register. Move the contents of the vector-length register to R1.
MVTM MVFM	VM,F0 F0,VM	Move contents of F0 to the vector-mask register. Move contents of vector-mask register to F0.

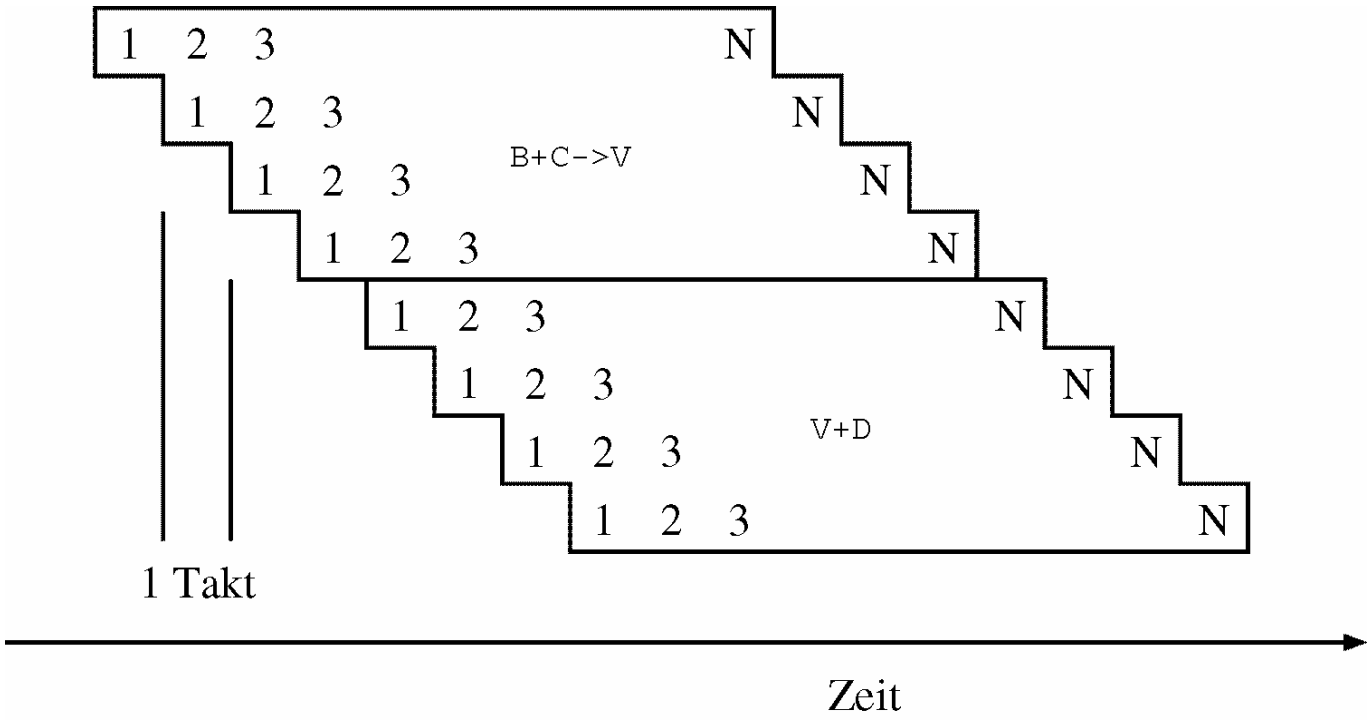
- Vektorprozessoren
  - Pipelining
    - Bei ununterbrochener Arbeit in der Pipeline kann man nach einer gewissen Einschwingzeit bzw. Füllzeit, die man braucht, um die Pipeline zu füllen, mit jedem Pipeline-Takt ein Ergebnis erwarten.
    - Dabei ist die Taktdauer durch die Dauer der längsten Teilverarbeitungszeit zuzüglich der Stufentransferzeit gegeben.
    - Beispiel Gleitkomma-Operationen:
      - Laden eines Paars von Gleitpunktzahlen aus Vektorregister
      - Vergleichen der Exponenten und Verschieben einer Mantisse
      - Addition der ausgerichteten Mantissen
      - Normalisieren des Ergebnisses und Schreiben in Zielregister

- Vektorprozessoren
  - Pipelining
    - Beispiel:
      - B[i] + C[i] mit i= 1,2,...,N

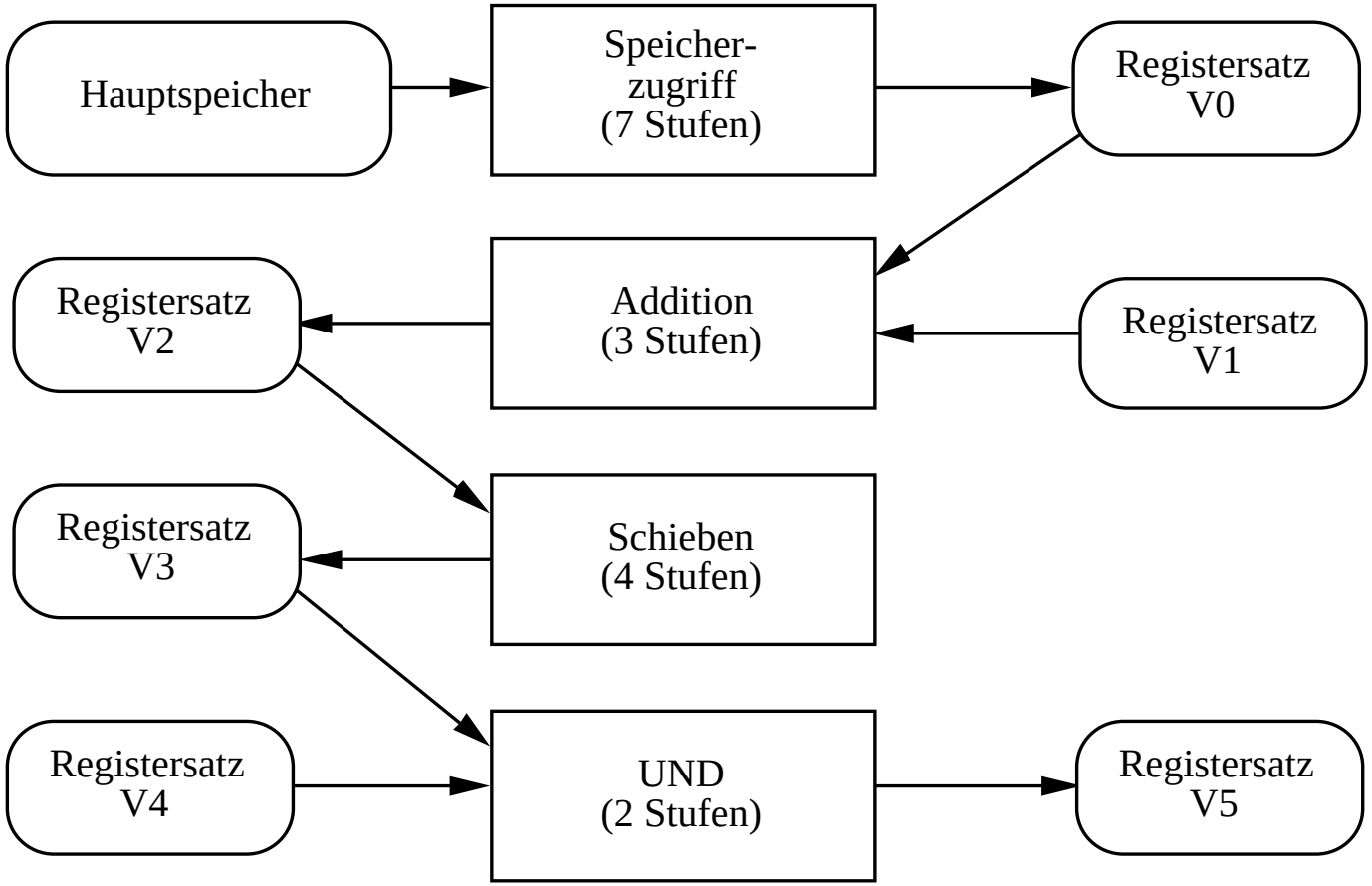


- Vektroprozessoren
  - Verkettung
    - Das Pipeline-Prinzip kann auch auf eine Folge von Vektoroperationen erweitert werden.
    - Zu diesem Zweck werden die (spezialisierten) Pipelines miteinander verkettet,
    - d.h. die Ergebnisse einer Pipeline werden sofort der nächsten Pipeline zur Verfügung gestellt.

- Vektroprozessoren
  - Verkettung
    - Beispiel:
      - B[i] * C[i] + D[i] mit i= 1,2,...,N



- Vektroprozessoren
  - Verkettung von 4 Pipelines



Aus: T. Ungerer: Parallelrechner und parallele Programmierung. Spektrum Akademischer-Verlag, Heidelberg, 1997

- Vektroprozessoren
  - Multifunktions- oder spezialisierte Pipelines
    - Zur Realisierung der arithmetisch-logischen Verknüpfung von Vektoren verwendet man entweder so genannte Multifunktions-Pipelines oder eine Anzahl von spezialisierten Pipelines.
    - Spezialisierte Pipelines werden zur Durchführung von speziellen Funktionen benutzt.
    - Hardware und Steuerung sind relativ einfach.
    - Man benötigt mehrere unabhängige Pipelines, um alle wichtigen Verknüpfungen durchführen zu können.

- Vektroprozessoren
  - Multifunktions- oder spezialisierte Pipelines
    - Zur Realisierung der arithmetisch-logischen Verknüpfung von Vektoren verwendet man entweder so genannte Multifunktions-Pipelines oder eine Anzahl von spezialisierten Pipelines.
    - Multifunktions-Pipelines
      - Der Aufbau einer Multifunktions-Pipeline erfordert eine höhere Stufenzahl, als sie zur Durchführung einer Verknüpfungsoperation notwendig wäre. Für die gerade aktuelle Operation werden alle nicht benötigten Stufen der Pipeline übersprungen.
    - Spezialisierte Pipelines
      - Durchführung von speziellen Funktionen benutzt.
      - Hardware und Steuerung sind relativ einfach.
      - Man benötigt mehrere unabhängige Pipelines, um alle wichtigen Verknüpfungen durchführen zu können.

- Vektroprozessoren
  - Parallelarbeit in einem Vektorrechner
    - Vektor-Pipeline-Parallelität:
      - durch die Stufenzahl der betrachteten Vektor-Pipeline gegeben
    - Mehrere Vektor-Pipelines in einer Vektoreinheit:
      - Vorhandensein mehrerer, meist funktional verschiedener Vektor-Pipelines in einer Vektoreinheit, durch Verkettung hintereinander geschaltet
    - Vervielfachung der Pipelines:
      - Vektor-Pipeline vervielfachen, so dass, bei Ausführung eines Vektorbefehl pro Takt nicht nur ein Paar von Operanden in eine Pipeline, sondern jeweils ein Operandenpaar in zwei oder mehr parallel arbeitende gleichartige Pipelines eingespeist werden.
    - mehrere Vektoreinheiten,
      - die parallel zueinander nach Art eines speichergekoppelten Multiprozessors arbeiten.

- Vektorprozessoren
  - Parallelitätsebenen in der Software
    - Vektor-Pipeline-Parallelität wird durch die Vektorisierung der innersten Schleife mittels eines vektorisierenden Compilers genutzt.
    - Mehrere Vektor-Pipelines in einer Vektoreinheit können durch Verkettung von Vektorbefehlen oder durch einen Vektor-Verbundbefehl (beispielsweise Vector-Multiply-Add) genutzt werden.
    - Bei der Vervielfachung der Pipelines (beispielsweise vier Vektoradditions-Pipelines, die mittels eines VADD-Befehls gleichzeitig aktiviert werden) geschieht die Vektorisierung wieder über die innerste Schleife

- Vektorprozessoren
  - Parallelitätsebenen in der Software
    - Das Vorhandensein mehrerer Vektoreinheiten wird durch ähnliche Parallelisierungsmechanismen wie für speichergekoppelte Multiprozessoren genutzt
    - Beispiel:
      - Aufteilung der Iterationen einer äußeren Schleife auf verschiedene Vektoreinheiten, welche die innere Schleife vektorisiert.