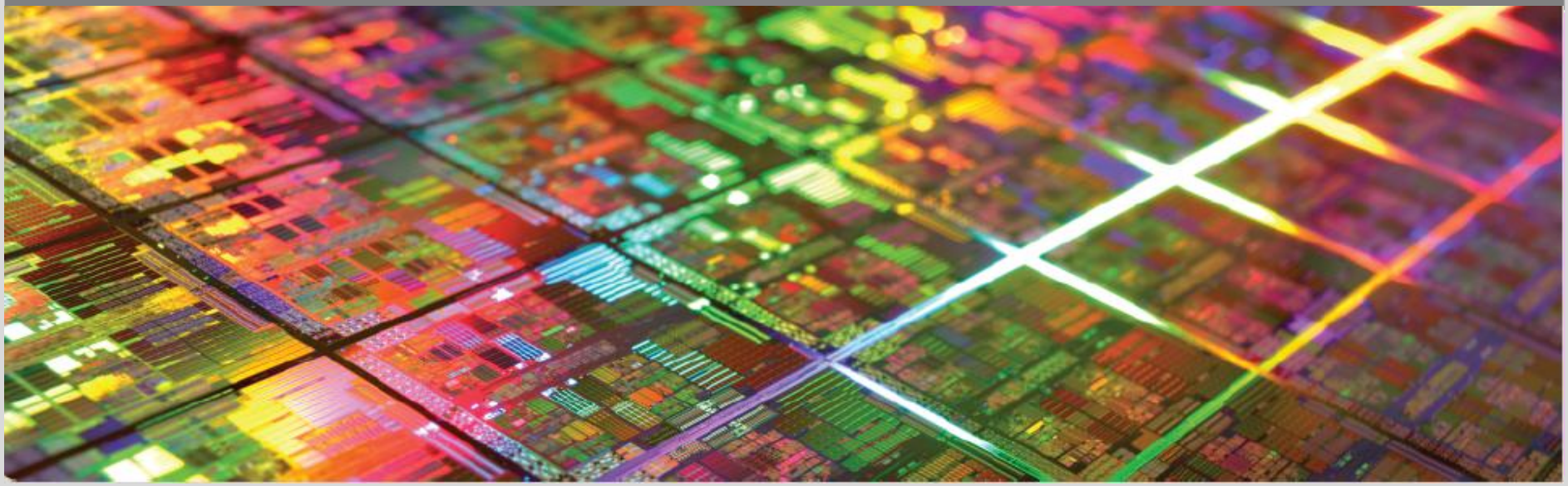
Karlsruher Institut für Technologie

Rechnerstrukturen

Vorlesung im Sommersemester 2011

Prof. Dr. Wolfgang Karl

Fakultät für Informatik – Lehrstuhl für Rechnerarchitektur und Parallelverarbeitung



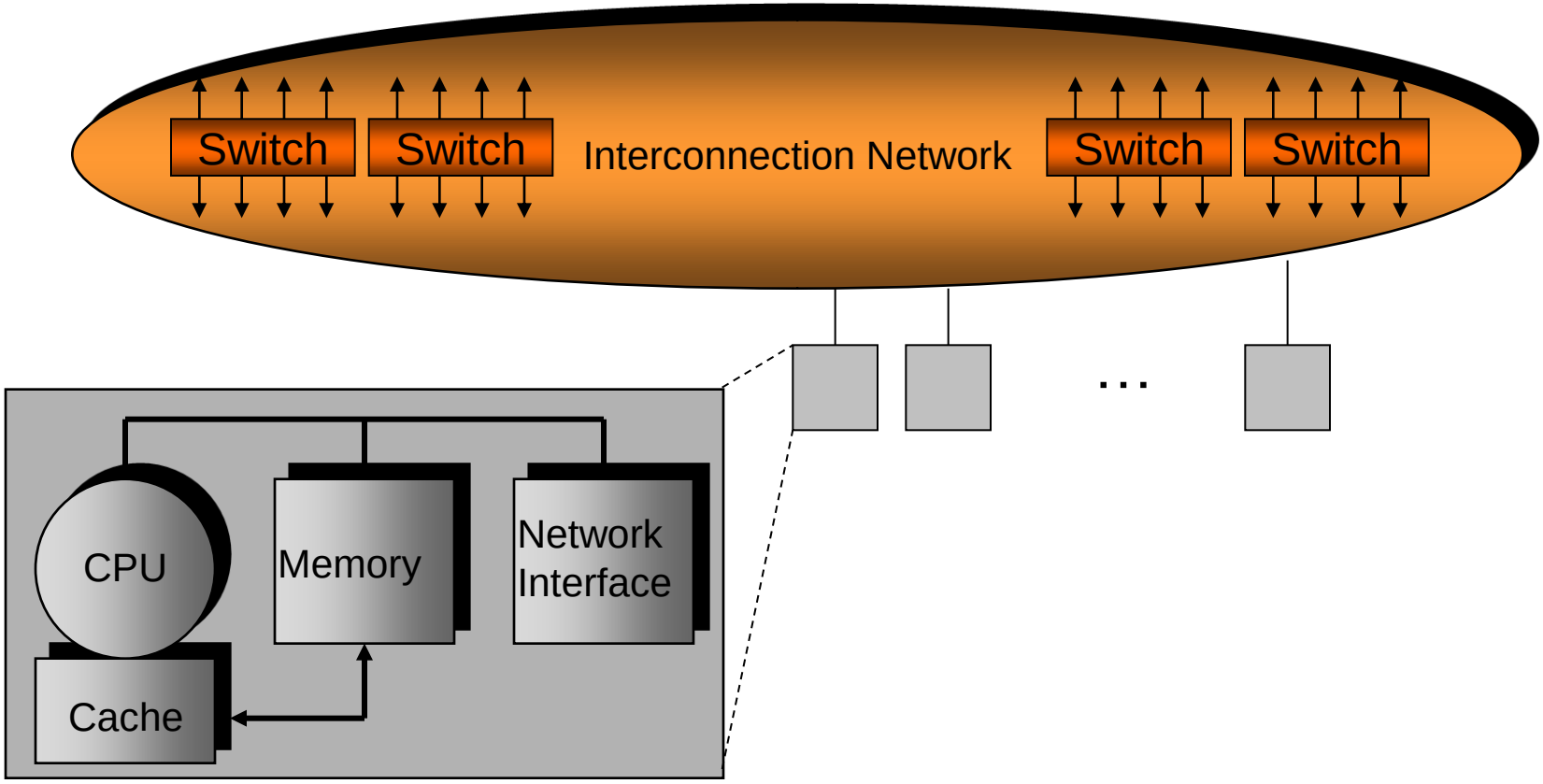
Vorlesung Rechnerstrukturen

Kapitel 3: Multiprozessoren – Parallelismus auf Prozess-/ Blockebene

- 3.1 Motivation
- 3.2 Allgemeine Grundlagen
- 3.3 Parallele Programmierung
- 3.4 Quantitative Maßzahlen
- 3.5 Verbindungsstrukturen
- 3.6 Multiprozessoren mit gemeinsamem Speicher
- 3.7 Multiprozessoren mit verteiltem Speicher

Multiprozessor mit verteiltem Speicher

- **Allgemeine Rechnerorganisation**



Multiprozessor mit verteiltem Speicher

- **Fallstudie IBM Blue Gene/L**
- Systemkomponenten
  - 65536 Knoten
    - in bis zu 64 Racks, die auch so organisiert werden können, als wären es verschiedene Systeme, wobei auf jedem ein eigenes Single Software Image läuft
  - Knoten
    - 2 BG/L Compute ASIC (BLC)
      - Dual Processor SoC ASIC
      - 9 Double data rate synchronous dynamic random access memory chips (DDR SDRAM chips) pro ASIC
  - Knoten über 5 Netzwerke verbunden
    - 64 x 32 x 32 3-D Torus
    - Global Collective Network
    - Global Barrier and Interrupt Network
    - I/O Network (Gigabit Ethernet)
    - Service Network

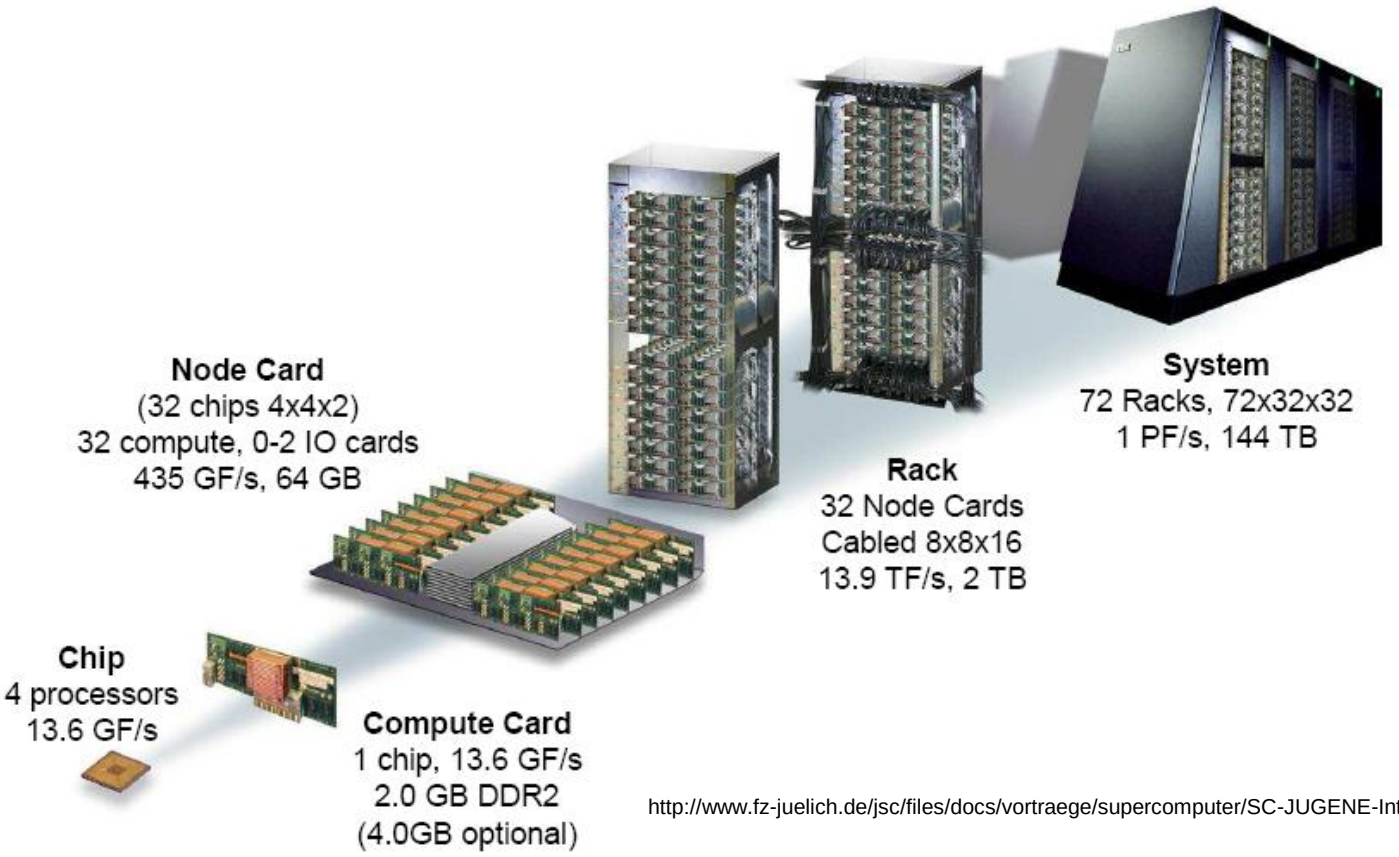
Multiprozessor mit verteiltem Speicher

- **Fallstudie JUGENE - Juelicher BlueGene/P**
  - Top500 – Juni 2010: Nr. 5
    - 825,5 TFLOPS
    - Cores: 294912

http://www.fz-juelich.de/jsc/files/docs/vortraege/supercomputer/SC-JUGENE-Introduction.pdf

Multiprozessor mit verteiltem Speicher

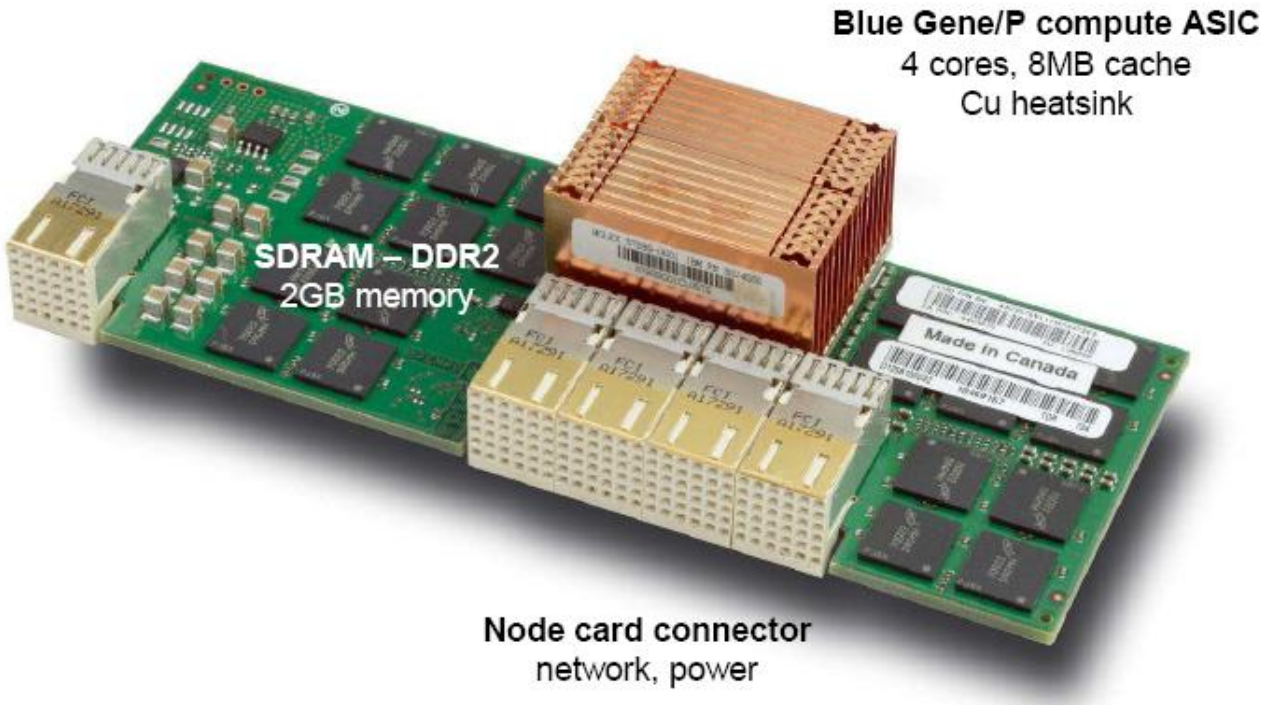
- **Fallstudie JUGENE - Juelicher BlueGene/P** Systemkomponenten



http://www.fz-juelich.de/jsc/files/docs/vortraege/supercomputer/SC-JUGENE-Introduction.pdf

Multiprozessor mit verteiltem Speicher

- **JUGENE - Juelicher BlueGene/P**
  - Compute Card



http://www.fz-juelich.de/jsc/files/docs/vortraege/supercomputer/SC-JUGENE-Introduction.pdf

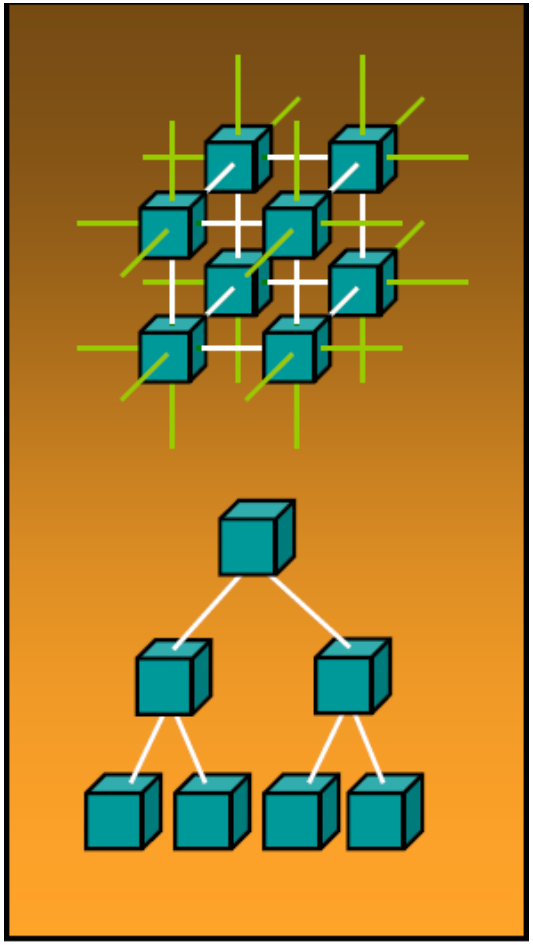
Multiprozessor mit verteiltem Speicher

- **JUGENE - Juelicher BlueGene/P**
  - Node Card

http://www.fz-juelich.de/jsc/files/docs/vortraege/supercomputer/SC-JUGENE-Introduction.pdf

Multiprozessor mit verteiltem Speicher

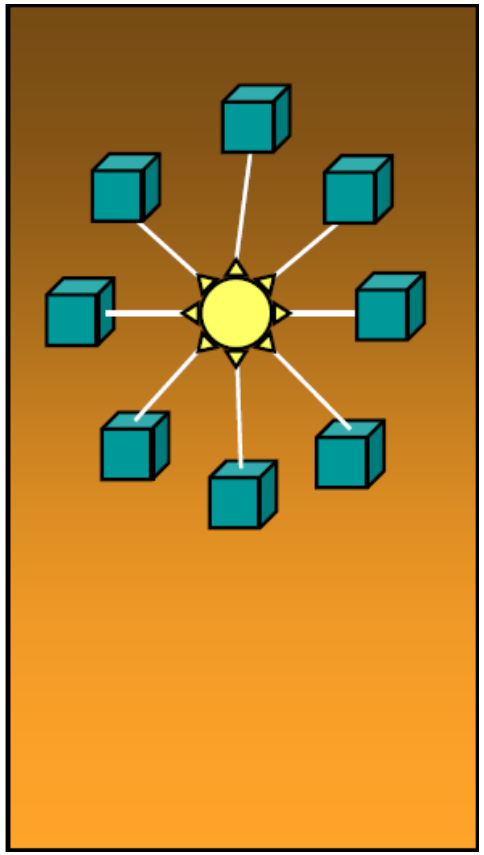
- **JUGENE - Juelicher BlueGene/P**
- **Verbindungsnetze**
- 3 Dimensional Torus
  - Interconnects all compute nodes (73,728)
  - Virtual cut-through hardware routing
  - 425 MB/s on all 12 node links (5.1 GB/s per node)
  - Communications backbone for computations
  - 188TB/s total bandwidth
- Collective Network
  - One-to-all broadcast functionality
  - Reduction operations functionality
  - 850 MB/s of bandwidth per link
  - Interconnects all compute and I/O nodes



http://www.fz-juelich.de/jsc/files/docs/vortraege/supercomputer/SC-JUGENE-Introduction.pdf

Multiprozessor mit verteiltem Speicher

- **JUGENE - Juelicher BlueGene/P**
- **Verbindungsnetze**
- Low Latency Global Barrier and Interrupt
  - Latency of one way to reach all 72K nodes 0.65 µs,
  - MPI 1.6 µs
  - External I/O Network
- 10 GBit Ethernet
  - Active in the I/O nodes
  - All external comm. (file I/O, control, user interaction)
- Control Network
  - 1 GBit Ethernet
  - Boot, monitoring and diagnostics



http://www.fz-juelich.de/jsc/files/docs/vortraege/supercomputer/SC-JUGENE-Introduction.pdf

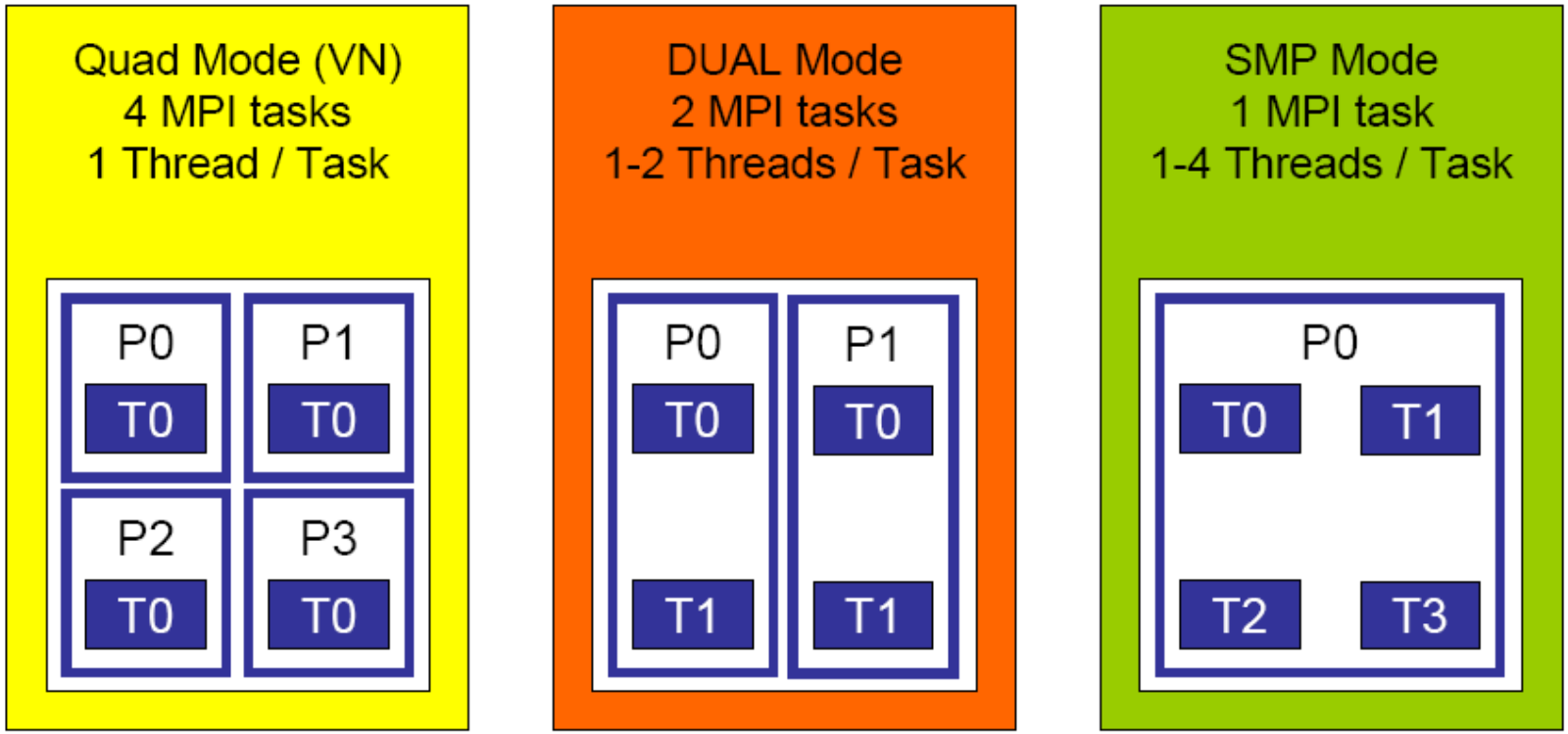
Multiprozessor mit verteiltem Speicher

- **Blue Gene/L ←→ Blue Gene/P**

Property		Blue Gene/L	Blue Gene/P
Node Properties	Node Processors Processor Frequency Coherency L3 Cache size (shared) Main Store Main Store Bandwidth (1:2 pclk) Peak Performance	2* 440 PowerPC® 0.7GHz Software managed 4MB 512MB/1GB 5.6GB/s 5.6GF/node	4* 450 PowerPC® 0.85GHz SMP 8MB 2GB/4GB 13.6 GB/s 13.9 GF/node
Torus Network	Bandwidth Hardware Latency (Nearest Neighbour) Hardware Latency (Worst Case)	6*2*175MB/s=2.1GB/s 200ns (32B packet) 1.6µs (256B packet) 6.4µs (64 hops)	6*2*425MB/s=5.1GB/s 100ns (32B packet) 800ns (256B packet) 3.2µs(64 hops)
Tree Network	Bandwidth Hardware Latency (worst case)	2*350MB/s=700MB/s 5.0µs	2*0.85GB/s=1.7GB/s 3.5µs
System Properties	Area (72k nodes) Peak Performance (72k nodes) Total Power	114m^2 410TF 1.7MW	160m^2 ~ 1PF ~2.3MW

Multiprozessor mit verteiltem Speicher

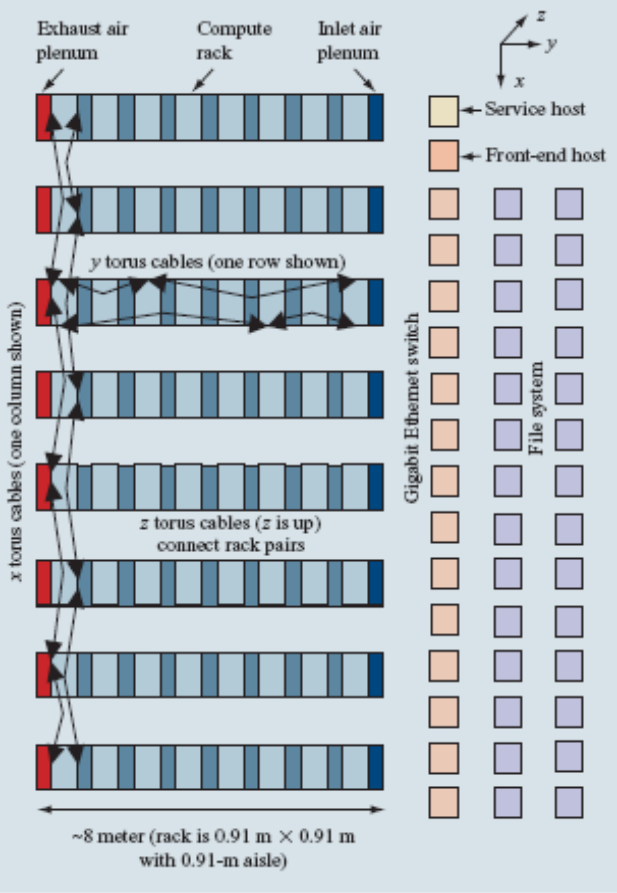
- **JUGENE - Juelicher BlueGene/P**
- Ausführungsmodi



http://www.fz-juelich.de/jsc/files/docs/vortraege/supercomputer/SC-JUGENE-Introduction.pdf

Multiprozessor mit verteiltem Speicher

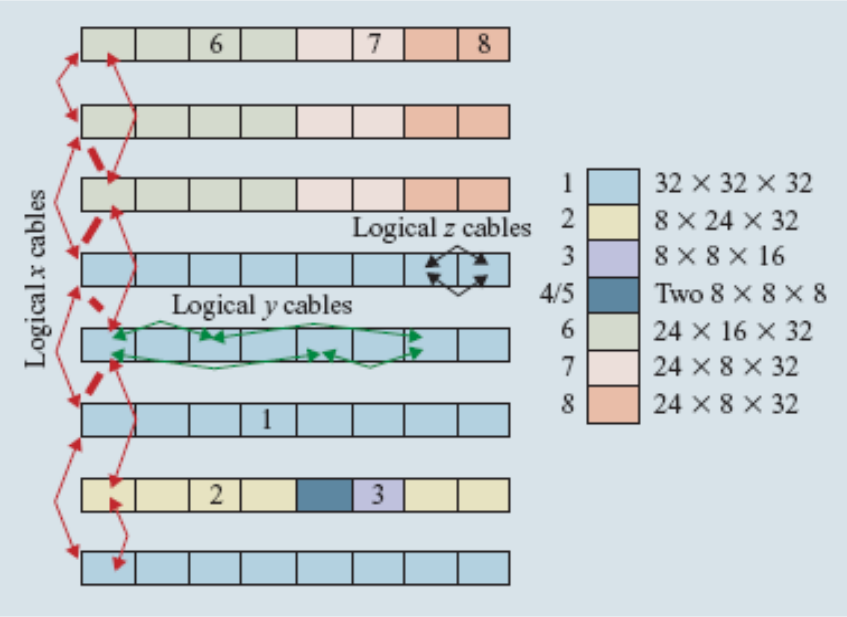
- **Fallstudie Blue Gene/L**
- Konfiguration



Gara et.al.: Overview of the Blue Gene/L system architecture. In: IBM Journal of Research and Development, Vol. 49, No. 2/3, 2005, pp.195-212

Multiprozessor mit verteiltem Speicher

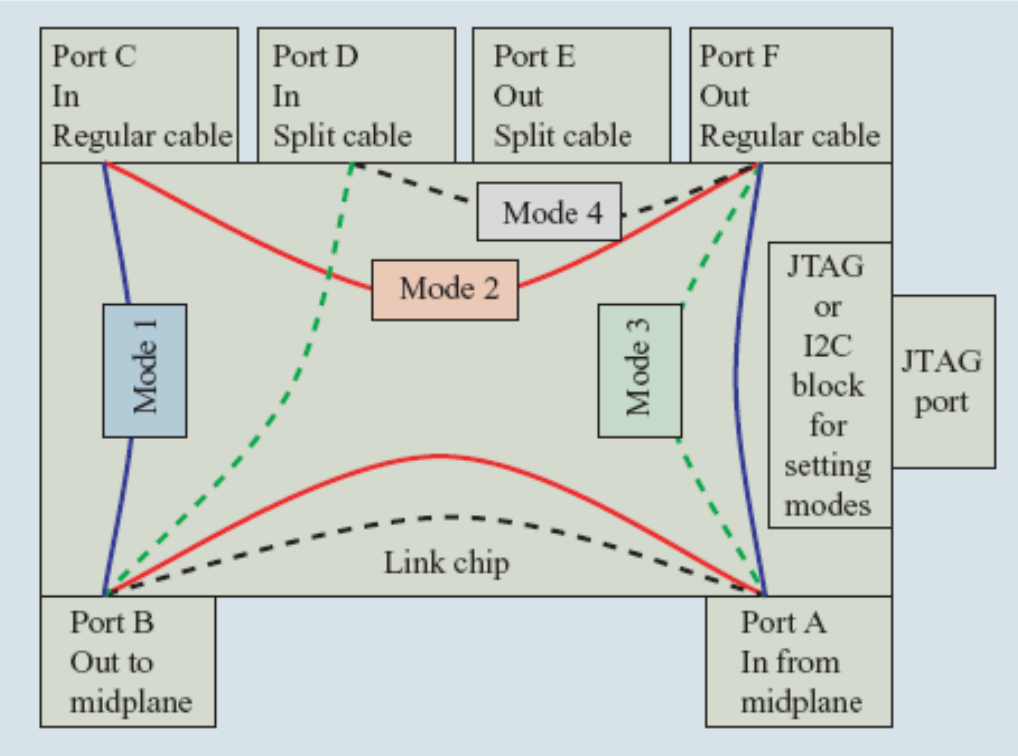
- **Fallstudie Blue Gene/L**
- Partitionierung
  - Beispiel: 8 Nutzer



Gara et.al.: Overview of the Blue Gene/L system architecture. In: IBM Journal of Research and Development, Vol. 49, No. 2/3, 2005, pp.195-212

Multiprozessor mit verteiltem Speicher

- **Fallstudie Blue Gene/L**
- BG/L Link Chip

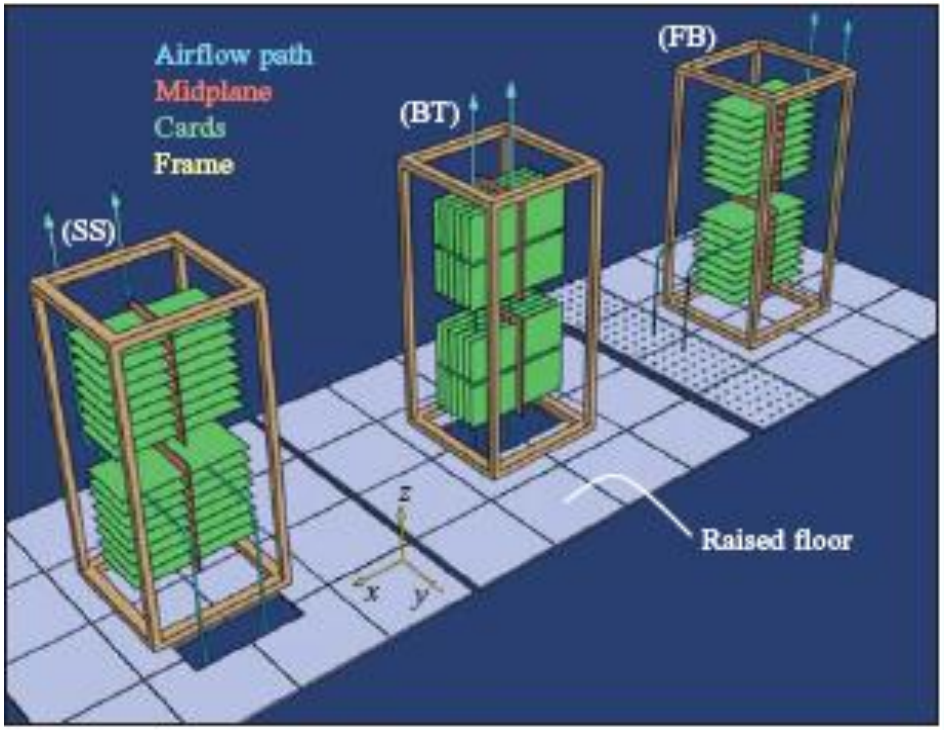


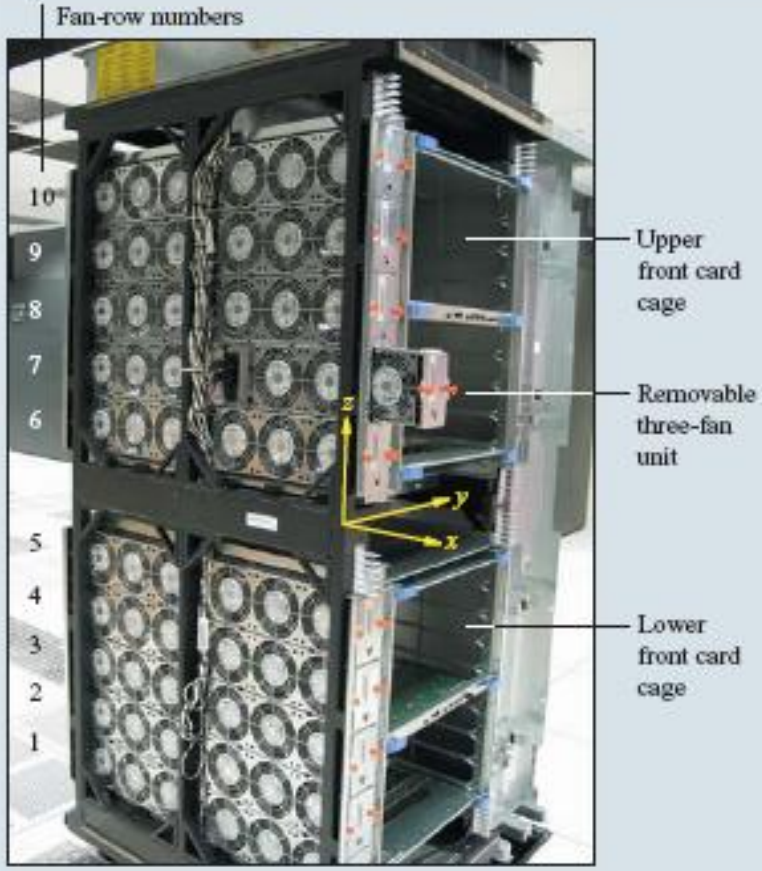
Gara et.al.: Overview of the Blue Gene/L system architecture. In: IBM Journal of Research and Development, Vol. 49, No. 2/3, 2005, pp.195-212

Fallstudie

IBM Blue Gene/L Überblick

- Systemkomponenten
  - Kühlung



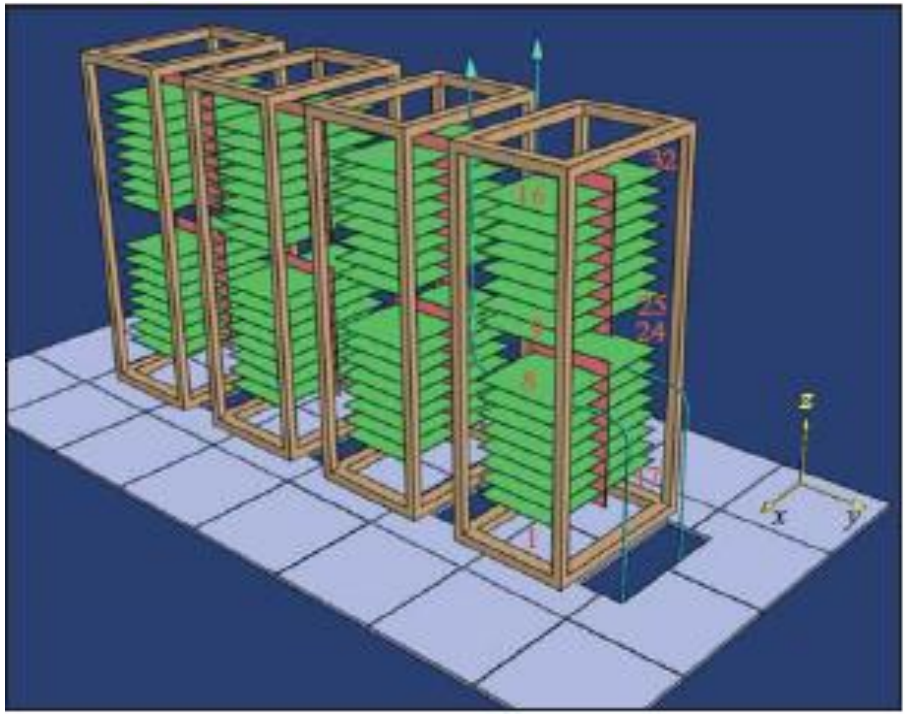


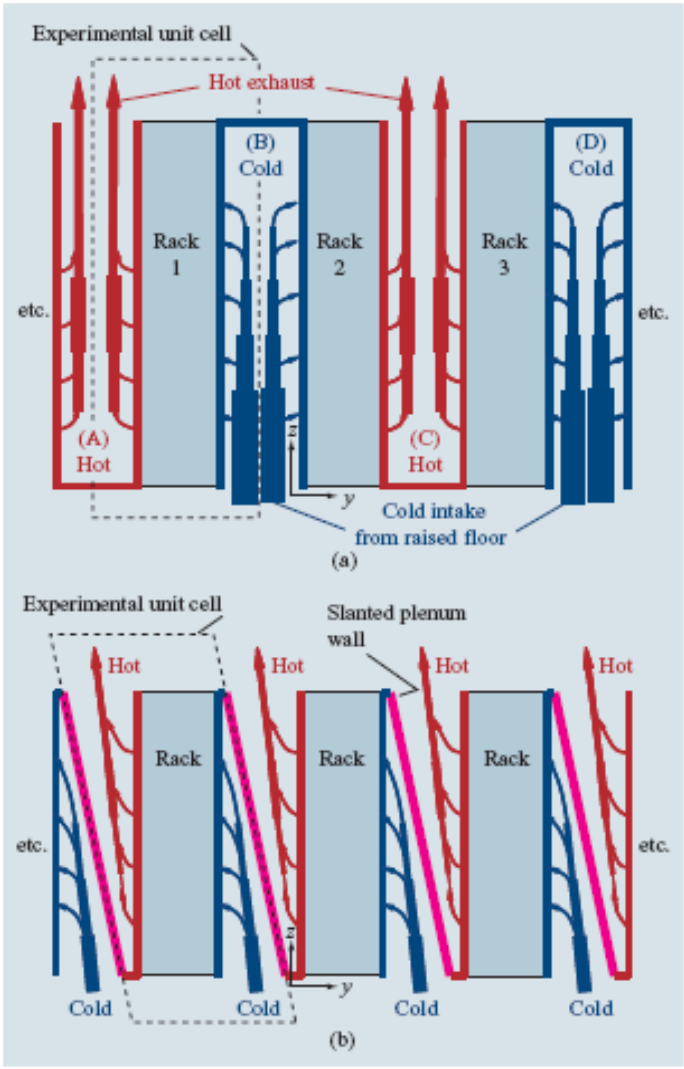
Quelle: P. Coteus, et.al.: Packaging the Blue Gene/L supercomputer. In: IBM Journal of Research and Development, Vol. 49, No. 2/3, 2005, pp.195-212

Fallstudie

IBM Blue Gene/L Überblick

- Systemaufbau





Quelle: P. Coteus, et.al.: Packaging the Blue Gene/L supercomputer. In: IBM Journal of Research and Development, Vol. 49, No. 2/3, 2005, pp.195-212

Fallstudie

- Literatur:
  - IBM Journal of Research and Development, Vol. 49, No. 2/3, 2005, Special Issue

Vorlesung Rechnerstrukturen

- **Kapitel 4: Vektorverarbeitung**
- 4.1 Grundlagen

Einführung

- **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]

Einführung

- **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
```

Einführung

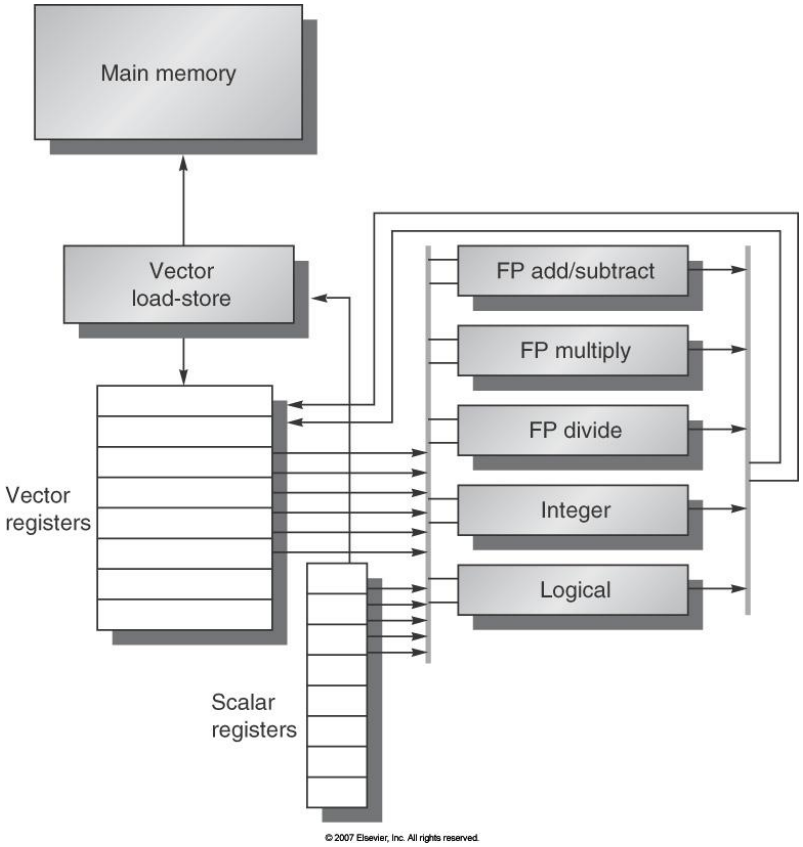
- **Wie arbeiten Vektorprozessoren?**
- Ein Beispiel: **Y = a * X + Y**,
- Analyse:
  - Die Schleife wird ungefähr 600 Mal durchlaufen.
  - Hoher Aufwand:
    - In einem Schleifendurchlauf werden jeweils die Elemente von X und Y addiert und nach der Berechnung wird das Ergebnis gespeichert. Die Adressen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

Einführung

- **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

Einführung

- **Wie arbeiten Vektorprozessoren?**
- Idee der Vektor-Prozessoren: SIMD-Verarbeitung



Beispielprogramm mit Vektoroperationen:

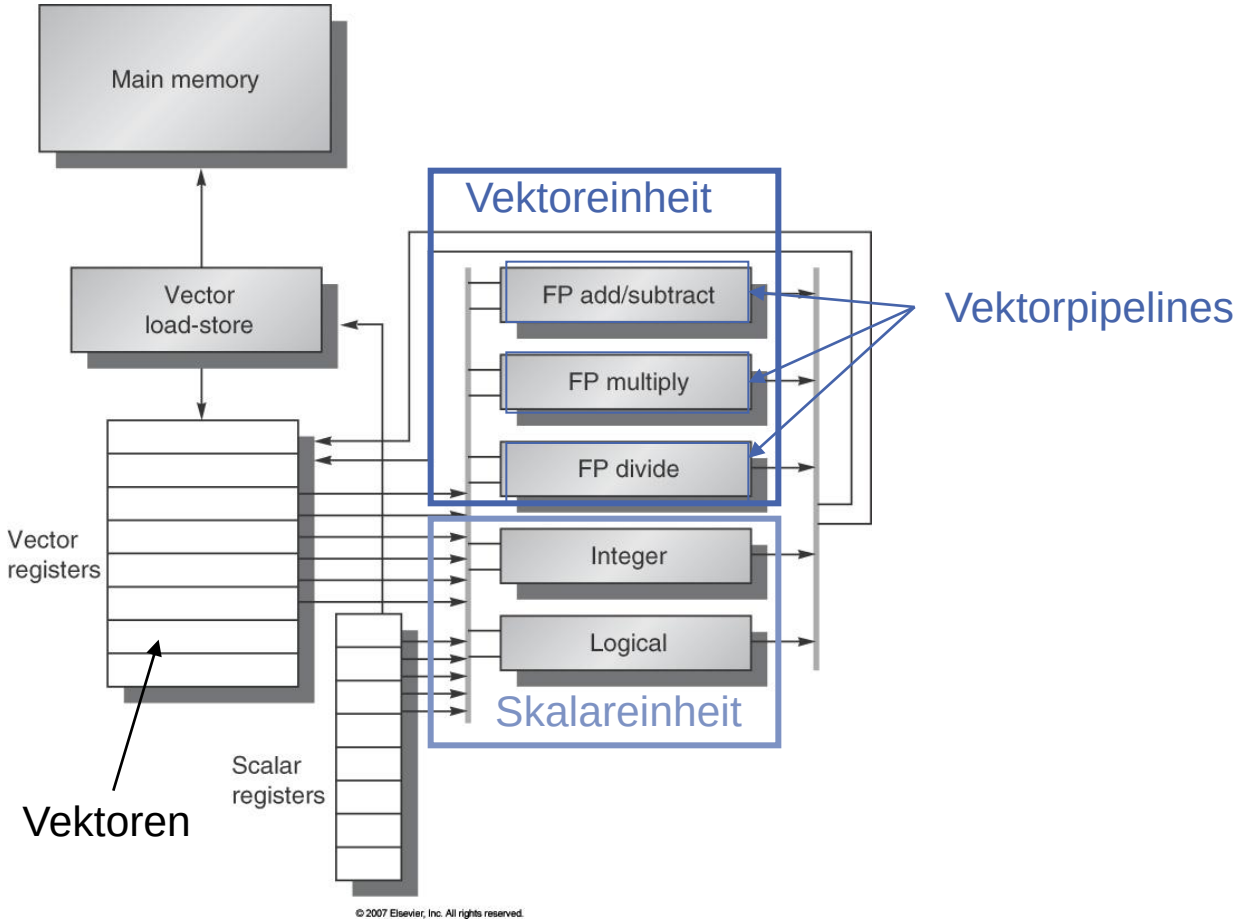
```
L.D     F0,a      ;load scalar a
LV      V1,Rx     ;load vector X
MULVS.D V2,V1,F0  ;vector-scalar multiply
LV      V3,Ry     ;load vector Y
ADDV.D  V4,V2,V3  ;add
SV      Ry,V4     ;store the result
```

Einführung

- **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. Dieses 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.

Einführung

- **Vektorprozessor:**



Einführung

- **Vektorprozessor:**
  - 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
```

Einführung

- **Vektorprozessor:**
- 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.

Einführung

- **Vektorprozessor:**
- Vektorbefehle

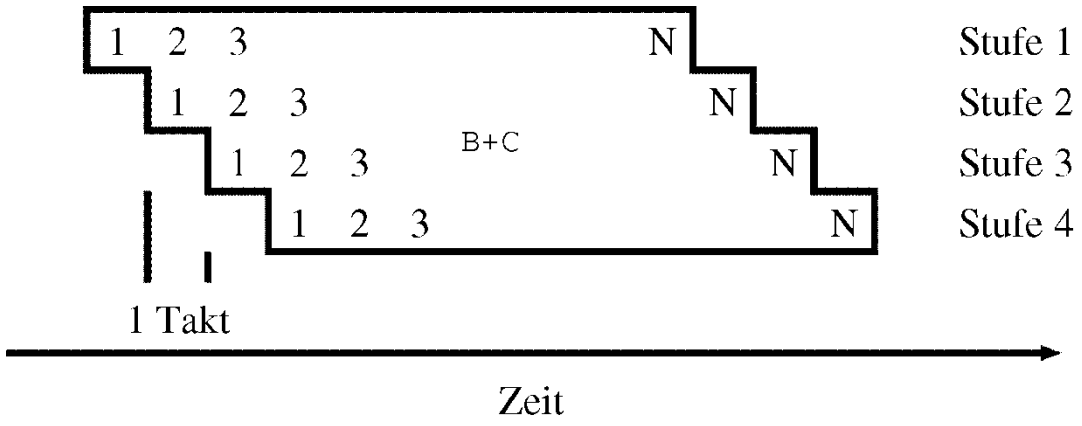
Instruktion	Operanden	Funktion
SVI	(R1+V2),V1	Store V1 to vector whose elements are at R1+V2(i), i.e., V2 is an index.
CVI	V1,R1	Create an index vector by storing the values 0, 1 × R1, 2 × R1,...,63 × R1 into V1.
S--V.D S--VS.D	V1,V2 V1,F0	Compare the elements (EQ, NE, GT, LT, GE, LE) in V1 and V2. If 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.

Einführung

- **Vektorprozessor:**
- 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

Einführung

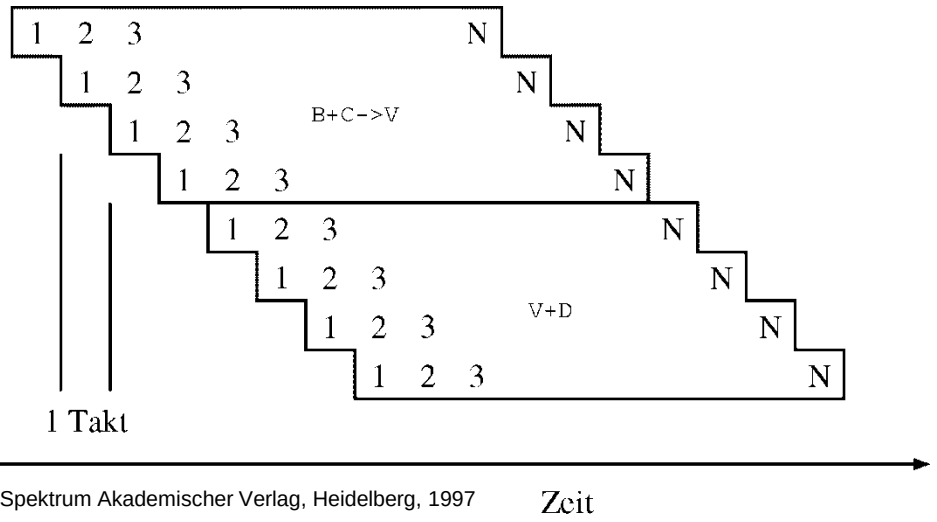
- **Vektorprozessor:**
- Pipelining
  - Beispiel: B[i] + C[i] mit i= 1,2,…,N



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

Einführung

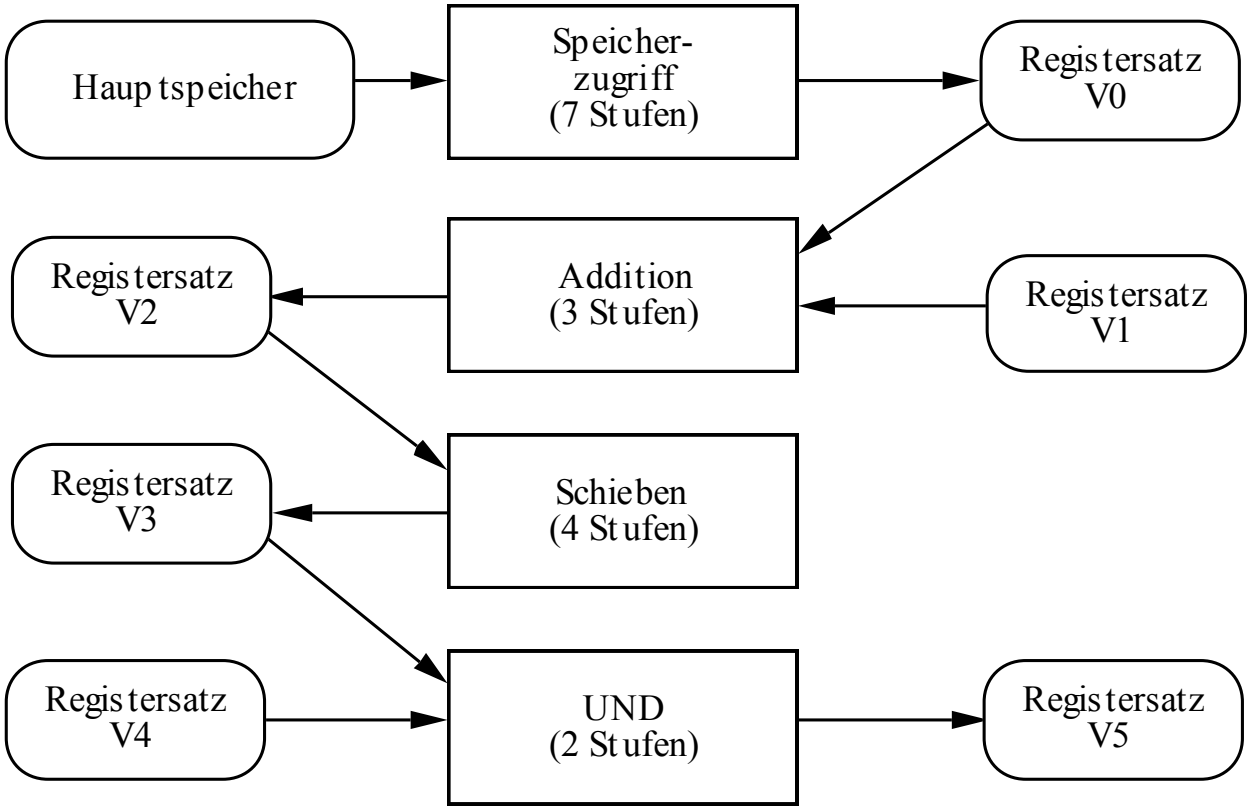
- **Vektorprozessor:**
- Verkettung
  - Das Pipeline-Prinzip kann auch auf eine Folge von Vektoroperationen erweitert werden.
  - Zu diesem Zweck werden die (spezialisierten) Pipelines miteinander verkettet,
  - d.h. die Ergebnisse einer Pipeline werden sofort der nächsten Pipeline zur Verfügung gestellt.
  - Beispiel:
    - B[i] * C[i] + D[i] mit i= 1,2,…,N



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

Einführung

- **Vektorprozessor:**
- Verkettung



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

Einführung

- **Vektorprozessor:**
- 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.

Einführung

- **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.

Einführung

- **Parallelarbeit in einem Vektorrechner**
- 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

Einführung

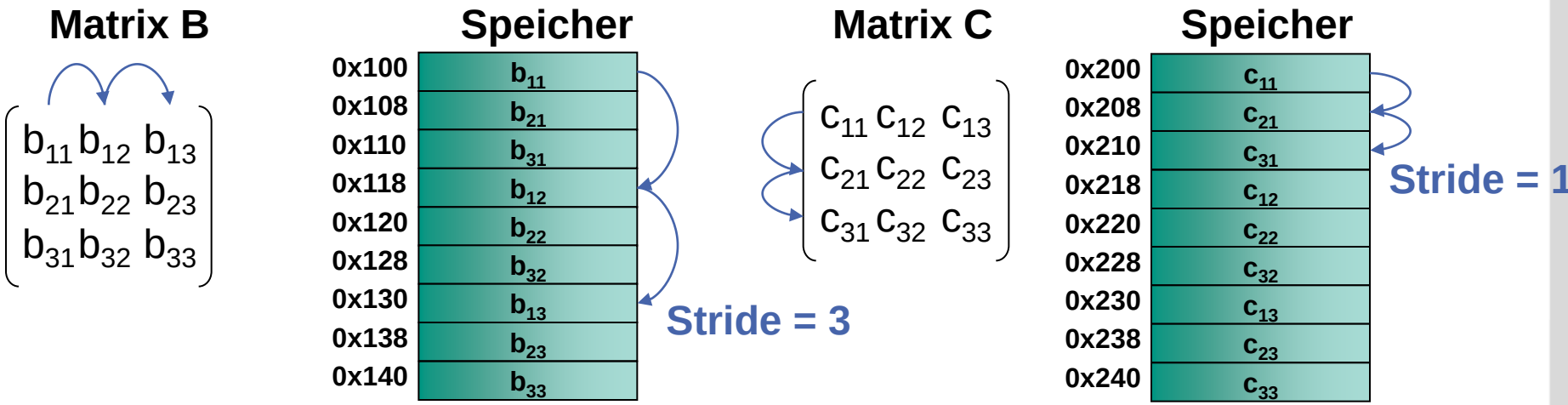
- **Parallelarbeit in einem Vektorrechner**
- Parallelitätsebenen in der Software
  - Vektor-Pipeline-Parallelität wird durch die Vektorisierung der innersten Schleife mittels eines vektorisierenden Compilers genutzt.
  - Mehrere Vektor-Pipelines in einer Vektoreinheit können durch Verkettung von Vektorbefehlen 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
  - 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.

Vorlesung Rechnerstrukturen

- **Kapitel 4: Vektorverarbeitung**
- 4.1 Eigenschaften

Vektorrechner

- **Vektor Stride**
  - Problem: die Elemente eines Vektors liegen nicht in aufeinander folgenden Speicherzellen
  - Beispiel: Matrix-Multiplikation (k=3)



Berechnung: $b_{11}* c_{11}+ b_{12}* c_{21}+b_{13}*c_{31}$ …

Vektorrechner

- **Vektor Stride**
  - Abstand zwischen Elementen, die in einem Register abgelegt werden müssen
  - Beispiel:
    - bei spaltenweiser Ablage der Daten im Speicher ist die der Stride für die Matrix C gleich 1 (oder 1 Doppelwort) und für die Matrix B gleich 3 (oder 3 Doppelwörter)
  - Vektorprozessoren mit Vektorregister können Strides größer 1 verarbeiten:
    - Vektorlade- und Vektrospeicherbefehle mit „Stride-Capability“
    - Zugriff auf nicht sequentielle Speicherzellen und Umformen in dichte Struktur

Vektorrechner

- **Vektor Stride**
- Problem:
  - Stride-Wert ist erst zur Laufzeit bekannt oder kann sich ändern
- Lösung
  - Ablegen des Stride-Wertes in ein Allzweckregister
  - Vektorspeicherzugriffsbefehle greifen auf den Wert zu
- Problem:
  - Zugriff auf eine Speicherbank erfolgt häufiger als es die Zugriffszeit der Bank erlaubt
  - Anhalten des Zugriffs, wenn:
- In heutigen Vektorrechnern:
  - Verteilen der Zugriffe von jedem Prozessor über mehrere Hundert von Speicherbänken

Vektorrechner

- **Bedingt ausgeführte Anweisungen**
- Problem:
  - Programme mit if-Anweisungen in Schleifen können nicht vektorisiert werden:
  - Kontrollflussabhängigkeiten!
  - Beispiel:

```
      do 100 i=1, 64
         if (A(i).ne. 0) then
             A(i) = A(i) –B(i)
         endif
  100 continue
```

- Lösung
  - Bedingt ausgeführte Anweisungen
  - Umwandlung von Kontrollflussabhängigkeiten in Datenabhängigkeiten

Vektorrechner

- **Bedingt ausgeführte Anweisungen**
- Vektor-Maskierungssteuerung
  - verwendet einen Boole‘schen Vektor der Länge der festgelegten MVL (maximale Vektorlänge), um die Ausführung eines Vektorbefehls zu steuern, in ähnlicher Weise wie bedingt ausgeführte Befehle eine Boole‘sche Bedingung verwenden, um zu bestimmen, ob eine Instruktion ein gültiges Ergebnis liefert oder nicht
- Vektor-Mask-Register
  - jede ausgeführte Vektorinstruktion arbeitet nur auf den Vektorelementen, deren Einträge eine 1 haben. Die Einträge im Zielvektorregister, die eine 0 im entsprechenden Feld des VM Registers haben, werden nicht verändert

Vektorrechner

- **Bedingt ausgeführte Anweisungen**
- Vektor-Maskierungssteuerung
  - Beispiel:

```
LV       V1,Ra     ;load vector A into V1
LV       V2,Rb     ;load vector B
L.D      F0,#0     ;load FP zero into F0
SNEVS.D  V1,F0     ;sets VM(i) to 1 if V1(i)!=F0
SUBV.D   V1,V1,V2  ;subtract under vector mask
CVM                ;set the vector mask to all 1s
SV       Ra,V1     ;store the result in A
```

Vektorrechner

- **Dünn besetzte Matrizen**
  - Elemente eines Vektors werden in einer komprimierten Form im Speicher abgelegt
  - Beispiel:

```
      do   100 i = 1,n
100           A(K(i)) = A(K(i)) + C(M(i))
```

  - implementiert die Summe der dünn besetzten Felder A und C mit Hilfe der Indexvektoren K und M. K und M zeigen jeweils die Elemente von A und C an, die nicht 0 sind.
  - Alternative Darstellung ist die Verwendung von Bit-Vektoren

Vektorrechner

- **Dünn besetzte Matrizen**
- SCATTER-GATHER Operationen mit Index-Vektoren
  - unterstützen den Transport zwischen der gepackten Darstellung und der normalen Darstellung dünn-besetzter Matrizen
  - GATHER-Operation
    - verwendet Index-Vektor und holt den Vektor, dessen Elemente an den Adressen liegen, die durch Addition einer Basisadresse und den Offsets im Index-Register berechnet werden
    - Nicht gepackte Darstellung im Vektorregister
  - SCATTER-Operation
    - Speichern der gepackten Darstellung

Vektorrechner

- **Dünn besetzte Matrizen**
- SCATTER-GATHER Operationen mit Index-Vektoren

```
LV   Vk,Rk            ;load K
LVI  Va,(Ra+Vk)       ;load A(K(I))
LV   Vm,Rm            ;load M
LVI  Vc,(Rc+Vm)       ;load C(M(I))
ADDV.D Va,Va,Vc       ;add them
SVI  (Ra+Vk),Va       ;store A(K(I))
```

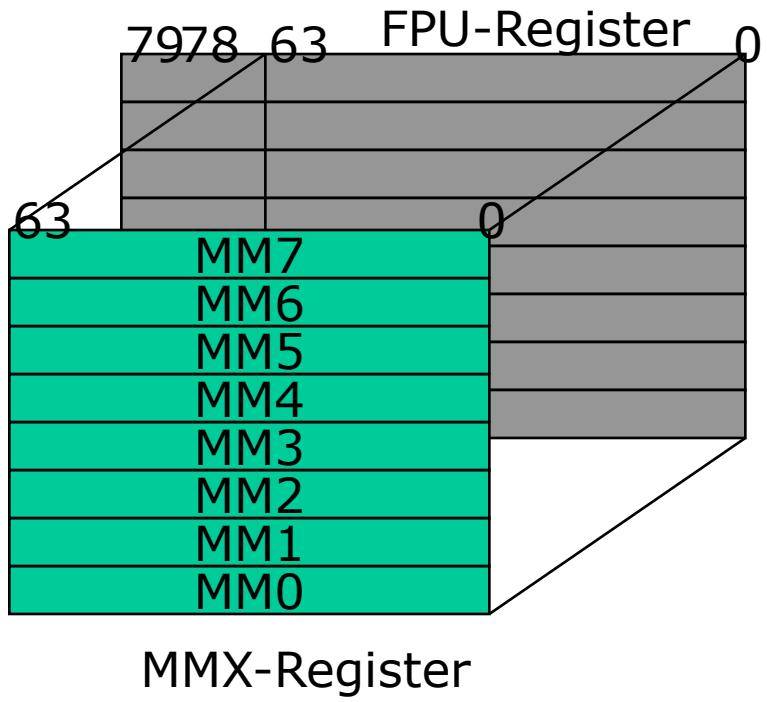
- Problem für vektorisierenden Compiler: konservative Annahmen wegen Speicherreferenzen
  - Verwendung einer Software-Hash Tabelle, ähnlich der ALAT im Intanium
  - erkennt, wenn zwei Elemente innerhalb einer Interation auf dieselbe Adresse zeigen

Vorlesung Rechnerstrukturen

- **Kapitel 4: Vektorverarbeitung**
- 4.3 SIMD-Verarbeitung in Mikroprozessoren

SIMD-Verarbeitung

- **Beispiel Intel MMX-Technologie**
  - MMX-Register, abgebildet auf FPU-Register

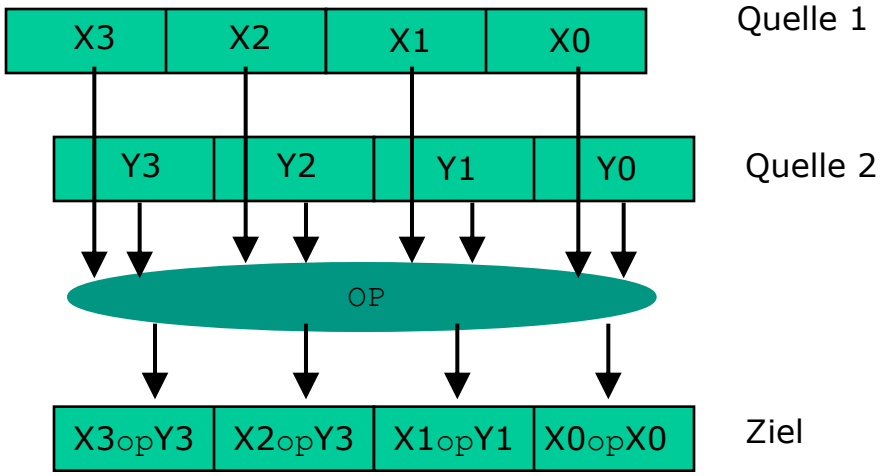


SIMD-Verarbeitung

- **Beispiel Intel MMX-Technologie**
  - MMX-Datentypen und Formate

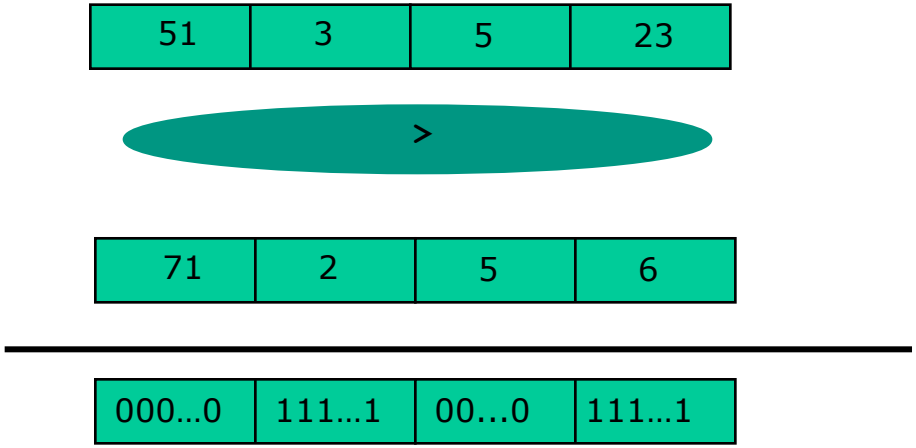
SIMD-Verarbeitung

- **Beispiel Intel MMX-Technologie**
  - SIMD-Verarbeitung



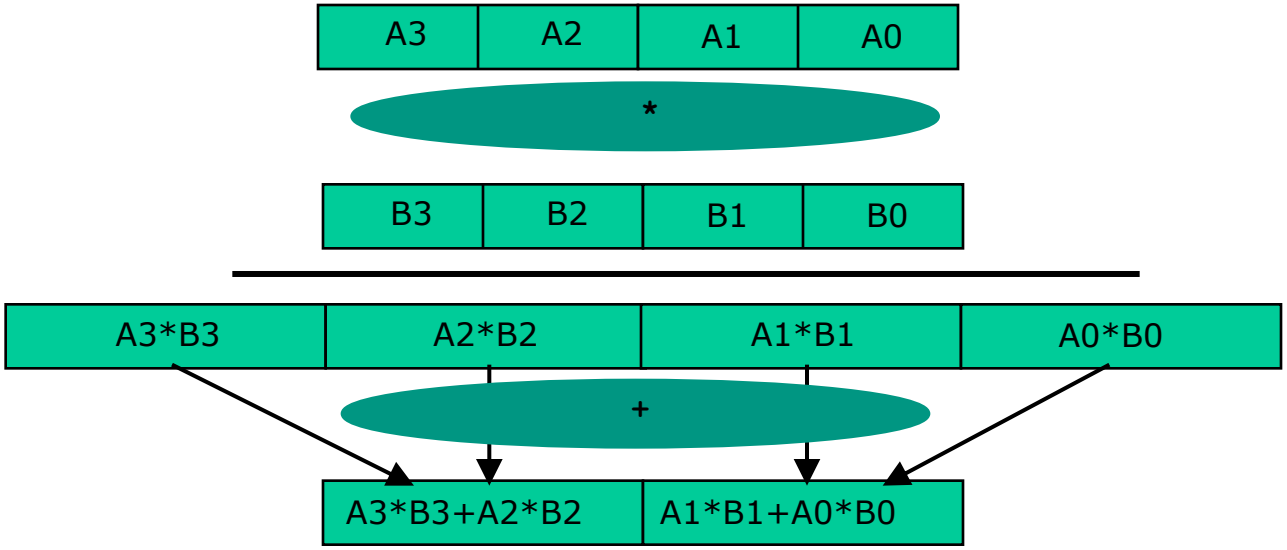
SIMD-Verarbeitung

- **Beispiel Intel MMX-Technologie**
  - SIMD-Verarbeitung
  - **`PCMPGTW`**-Befehl auf Wortoperanden (Vergleich größer als)



SIMD-Verarbeitung

- **Beispiel Intel MMX-Technologie**
  - SIMD-Verarbeitung
    - **`PMADDWD`**-Befehl auf Wortoperanden, Ergebnis im Doppelwort (Multiply and add)



SIMD-Verarbeitung

- **Beispiel Intel MMX-Technologie**
  - Saturation Arithmetik
    - Algorithmen in der graphischen Datenverarbeitung vermeiden Überlauf und Unterlauf bei der Addition und Subtraktion von nicht vorzeichenbehafteten Pixeln
    - Übergang zum größten oder kleinsten darstellbaren Wert

```
  10101010
+ 11001100
  11111111
```

    - Keine Überprüfung, ob Über- oder Unterlauf des Zahlenbereichs, keine Ausnahmeverarbeitung

SIMD-Verarbeitung

- **Beispiel Intel MMX-Technologie**
  - Beispiel: Chroma keying („bluebox“ Technik)
    - Überlagerung des Bildes einer Frau auf ein Bild mit einer Blume mit Hilfe eines blauen Hintergrundes
      - x: Bild der Frau auf dem blauen Hintergrund
      - y: Blumenbild
      - new_image: neues Bild

```
for (i=0; i<image_size; i++) {
    if (x[i]==blue) new_image[i]=y[i]
          else      new_image[i]=x[i]
}
```

SIMD-Verarbeitung

- **Beispiel Intel MMX-Technologie**
  - Beispiel: Chroma keying („bluebox“ Technik)
    - MMX Befehlsfolge

```
movq      mm3,mem1      #load 8 pixels from woman's image (addr. mem1)
movq      mm4,mem2      #load 8 pixels from flower's image (addr. mem2)
pcmpeqb   mm1,mm3       #gen. Selection bit mask into mm1 (orig. „blue“)
pand      mm4,mm1       #AND; use the bit mask for cond. select into mm4.
pandn     mm1,mm3       #(NOT mm1) AND mm3; -“- into mm1

por       mm4,mm1       #OR; result into mm4
```