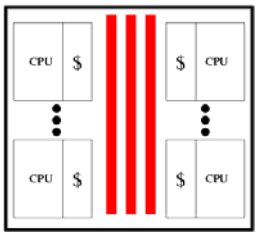
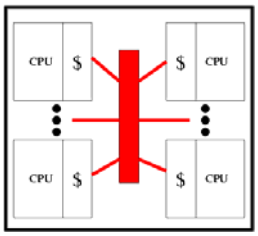
Token Coherence

Alternative Networks

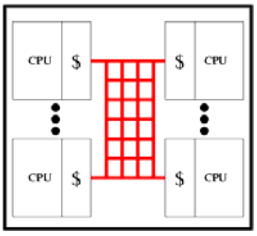
Bus



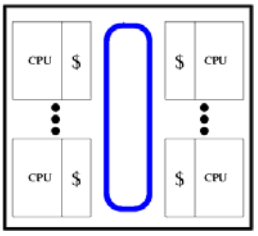
Crossbar



Packet-Switched



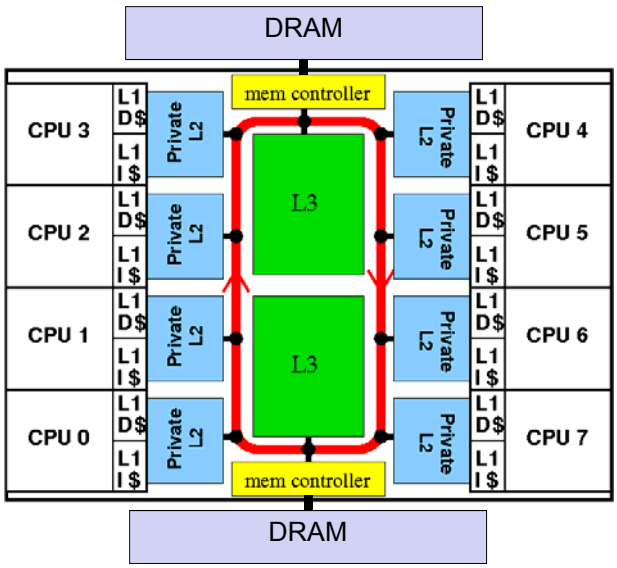
Ring?



Ring Interconnect

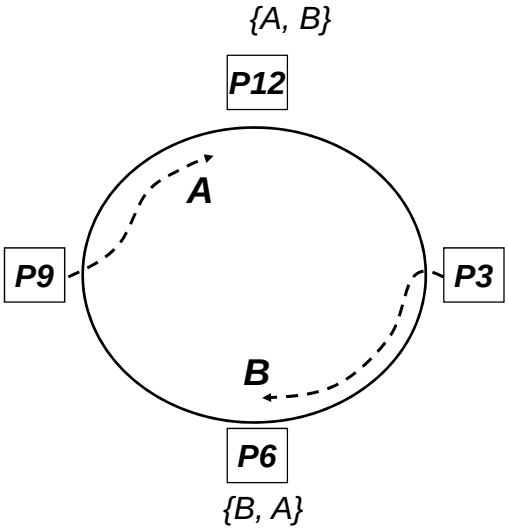
- Why to use rings?
  - Fast point-to-point links
  - Short links
  - Small constant number of ports
  - Less complex than packed-switched
  - Simple, distributed arbitration
  - Exploitable ordering for coherence

Example Architecture



Cache Coherence for a Ring

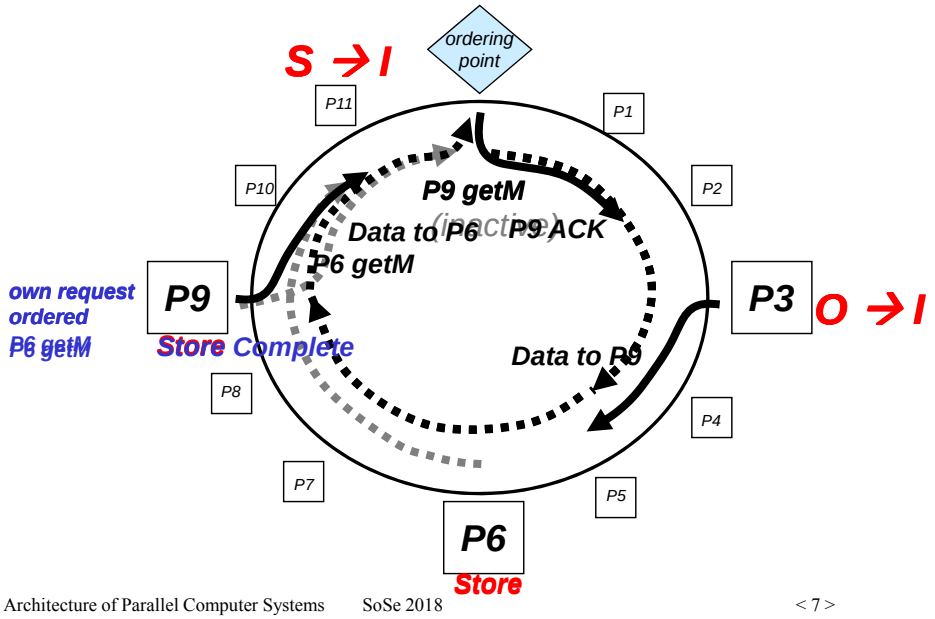
- Apply existing bus-based snooping protocols?
- Not that easy
  - Broadcast is simple
  - Order properties of a ring are different



Snooping Protocol for Rings

- Create a total bus order on a ring
- Assumptions
  - Unidirectional ring
  - Write-back, write-invalidate caches
  - Request forwarding

Example: Ordering-Point

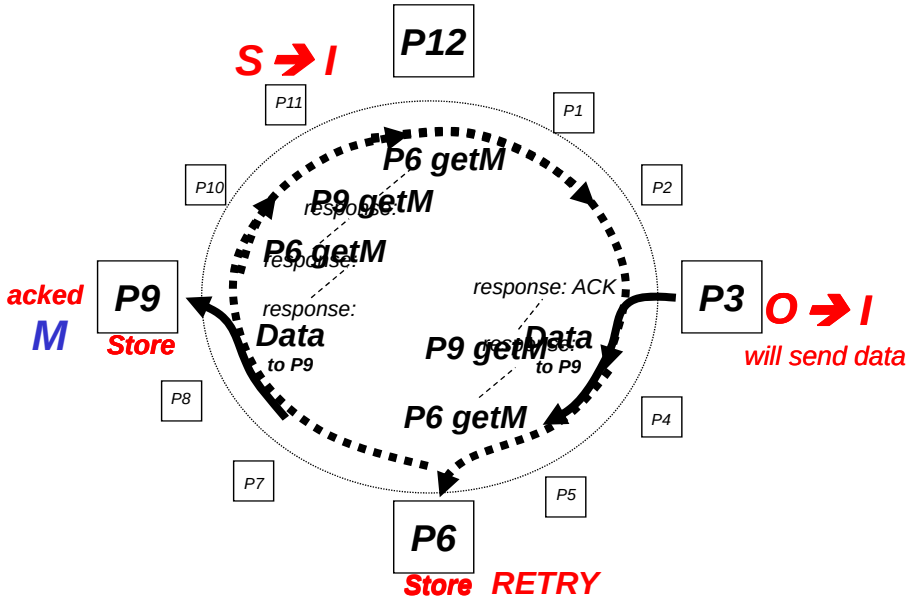


Results: Ordering-Point

- Requests totally ordered
  - Stable
  - Predictable performance
- Slow execution
  - Requests not immediately active
  - Control overhead
    - N+N/2 hops for request message
    - and N/2 hops for acknowledgement message

Can requests be immediately active?

Example: Greedy-Order



Result: Greedy-Order

- Average case is fast
  - Requests are immediately active
- Unbounded retries
  - Starvation of writers

Ideal Solution
- Fast for average case
- Stable for worse-case (no tries)

Background

- „single-writer or many-readers“ cache coherence invariant
- Invalidation based Protocol is a distributed algorithm of this invariant
- Token counting can be used to enforce the invariant

- T tokens are assigned to each block (T >= # of caches)
- One token is designed as an owner token
- Tokens can be held in processor caches, memory modules, and coherence messages
- Initially, the block's home memory module holds all tokens for a block

Addition rules for handling tokens and data have to be defined

Token Counting Rules

Rule #1 (Conservation of Tokens)

After system initiation, tokens may not be created or destroyed.
One token for each block is the owner token. The owner token can be either clean or dirty. Whenever the memory receives the owner token, the memory sets the owner token to clean.

Rule #2 (Write Rule)

A component can write a block only if it holds all T tokens for that block and has valid data. After writing the block, the writer sets the owner token to dirty.

Rule # 3 (Read Rule)

A component can read a block only if it holds at least one token for that block and has valid data.

Rule #4 (Data Transfer Rule)

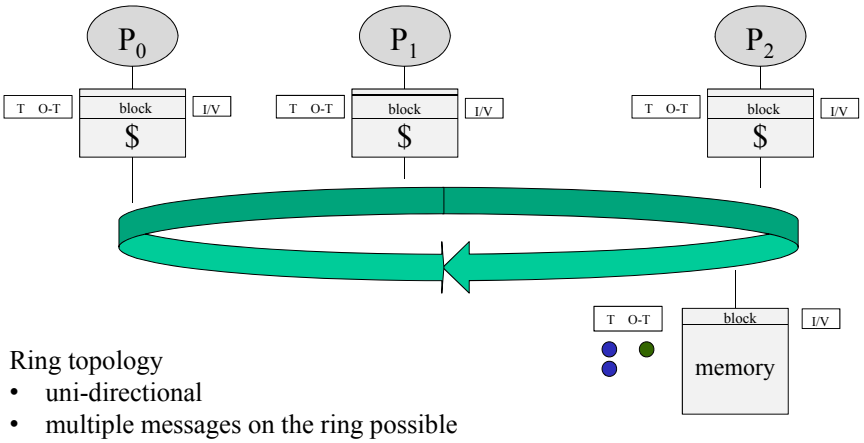
If a coherence message contains a dirty owner token, it must contain data.

Rule #5 (Valid-Data Bit Rule)

A component sets valid-data bit for a block when data and at least one token arrives.
Valid-data bit is cleaned when it no longer holds any tokens.
Home memory sets valid-data bit whenever it receives a clean owner token.

Example Architecture with Token Coherence

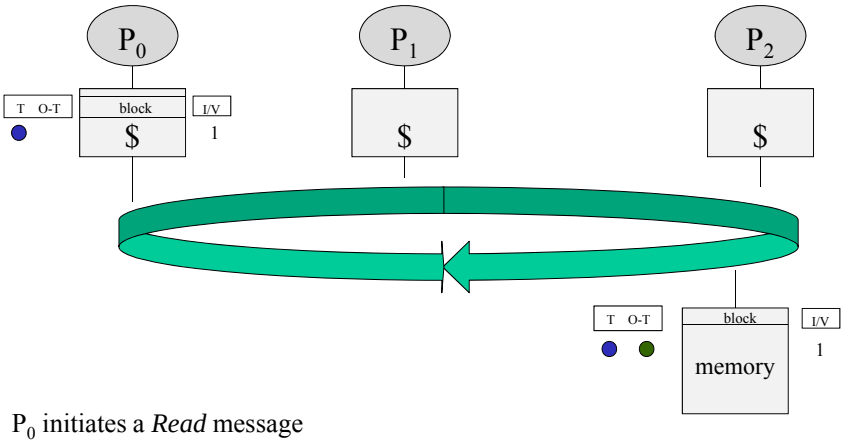
I/V =1 ⇔ block is valid



Ring topology
- uni-directional
- multiple messages on the ring possible
- max. one message per ring segment

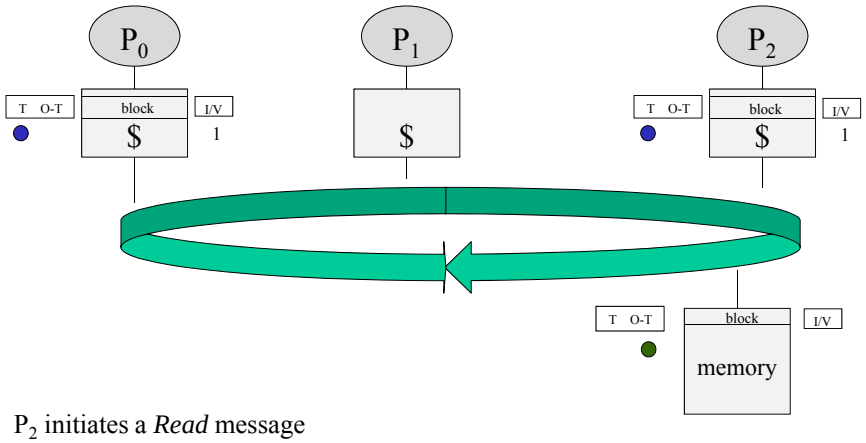
standard Token
O-Token (Clean/Dirty)

Example: Token Coherence (Read P_0)



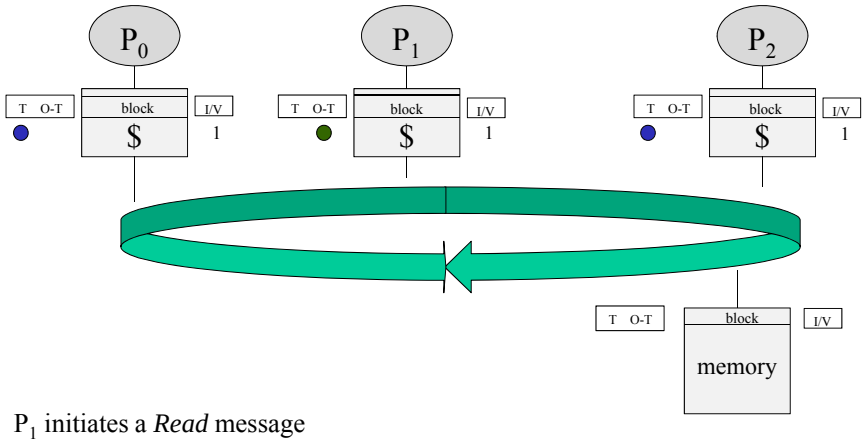
- P_0 initiates a *Read* message
- Memory puts standard token and the block in the message
- P_0 takes standard token and block; sets block to valid

Example: Token Coherence (Read P_2)



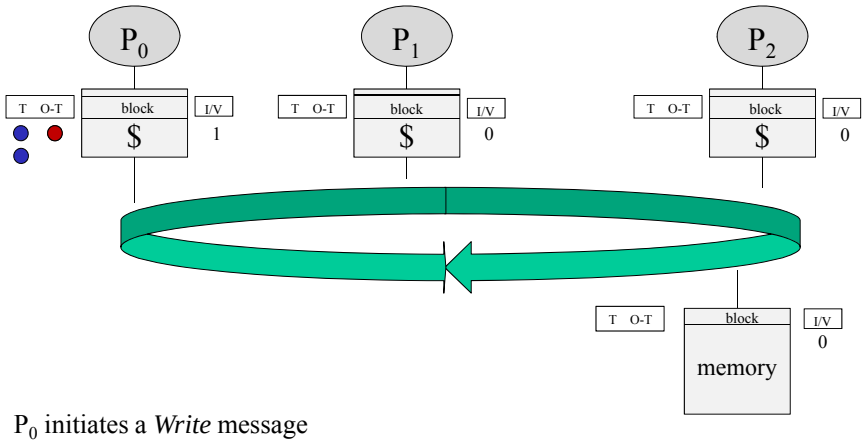
- P_2 initiates a *Read* message
- Memory puts standard token and the block in the message
- P_2 takes standard token and block; sets block to valid

Example: Token Coherence (Read P_1)



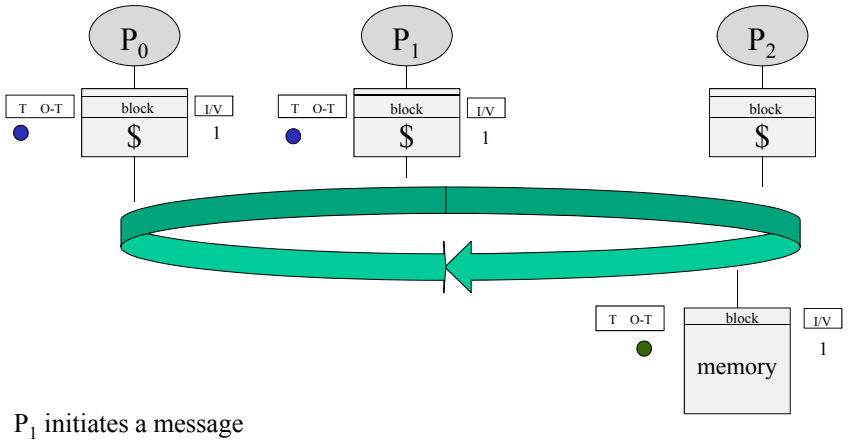
- P_1 initiates a *Read* message
- Memory puts last token (clean owner) and the block in the message
- P_1 takes owner token and block; sets block to valid

Example: Token Coherence (Write P_0)



- P_0 initiates a *Write* message
- P_1 puts owner token in the message, invalidates block
- P_2 puts standard token in the message, invalidates block
- P_0 takes all tokens; switches owner token to dirty

Example: Token Coherence (Read P_1)



- P_1 initiates a message
- P_0 puts owner token (clean), one standard token and the block in the message
- P_1 takes standard token and block
- Memory takes owner token and block; sets block to valid

Token Counting and Directory

- no broadcast and no ordered interconnect needed
- Uses unconstrained predictive direct requests

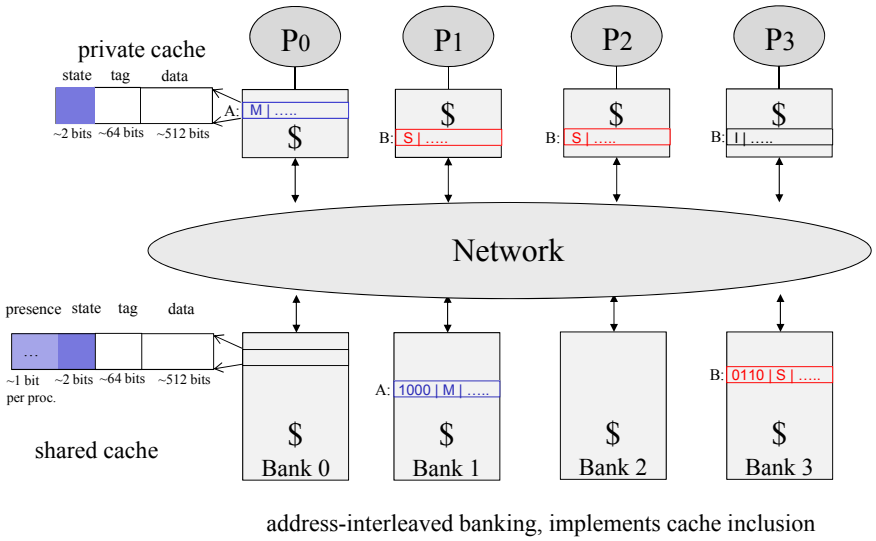
- Combining token counting and standard directory protocol
  - Token counting ensures coherency
  - Enables indirection-free sharing misses

Mapping of Token Counts to MOESI States

Tokens	Owner?	State
All	Dirty	M
Some	Dirty	O
All	Clean	E
Some	Clean	„F“
Some	Not held	S
None	Not held	I

Alternative Multilevel Cache Architectures

Distributed Shared Cache with Directory



address-interleaved banking, implements cache inclusion

Distributed Shared Cache with Directory

- On-chip cache coherence protocol used by Intel's Core i7
- Scales from 4 up to 16 cores
- 16-cores system adds 16 bits for each 64 byte cache line (~3% overhead)
- Miss to a block not cached in a private cache only introduces a small overhead in miss latency and consumed energy
- When blocks are shared, the traffic per miss is limited and independent of the number of cores

Scalability

- Some people forecast that the era of cache coherence nearing its end
- But State-of-the-art coherence mechanism are more scalable than we expect
- Scalability is also limited by
  - Memory technology
  - Interconnection networks
- Potential concerns when scaling on-chip coherence are:
  - Traffic on the on-chip interconnection network
  - Storage cost for tracking sharers
  - Inefficiencies caused by maintaining inclusion (if inclusion is needed)
  - Latency of cache misses
  - Energy overhead

Example: Xeon Phi

Xeon Phi Coprocessor Architecture

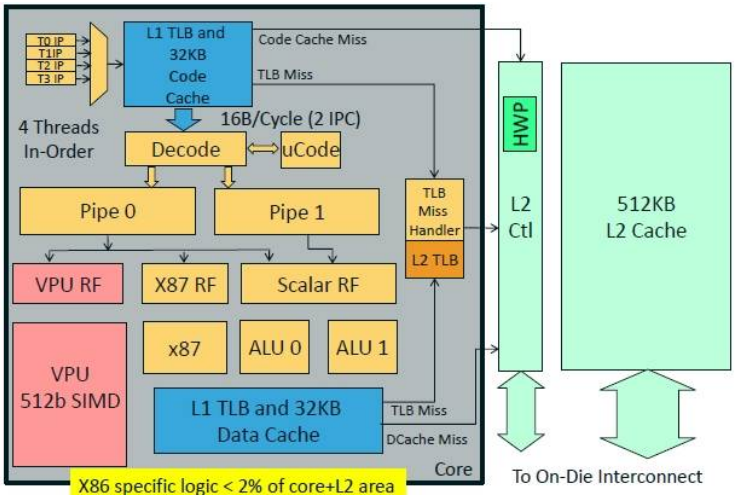
A single die with

- Up to 64 cores
- Dual-issue in-order execution pipelines
- Each core with
  - 512 bit wide vector processor units (VPU)
  - Core Ring Interface (CRI)
  - Interfaces to the core and the Ring interconnect
  - L2-Cache
  - Tag Directory (TD), part of the distributed duplicate tag directory infrastructure
  - Asynchronous Processor Interrupt Controller (APIC)
- Memory controllers (GBOX)
- A Gen2 PCI Express client logic (SBOX)
- The Ring Interconnect connecting all components

Xeon Phi 5110P with 60 cores

Intel Xeon Phi – First Generation

A Xeon Phi „Pentium“ core

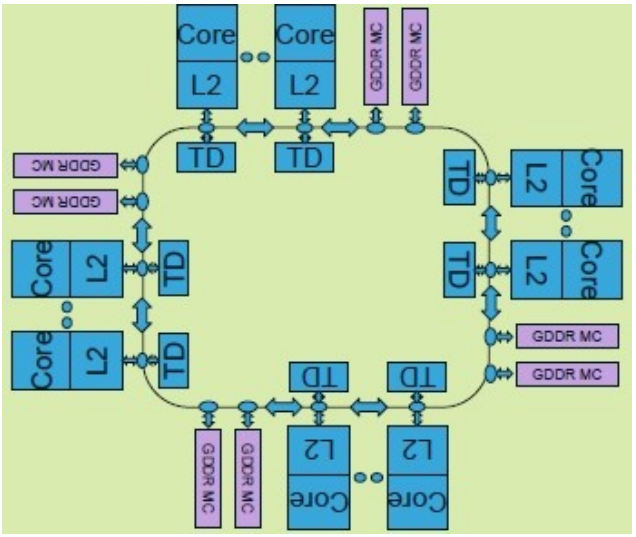


Intel MIC – many integrated cores

Xeon Phi Core Architecture

- 64-bit execution environment, x86-64 instructions
- No support for MMX, AVX and SSE instructions
- New vector instructions (AVX-512) utilize a dedicated 512-bit wide vector floating-point unit (VPU)
- Support of four hardware threads
- Core can execute two instructions per clock cycle (1 x U-pipe + 1 x V-pipe)
- vector register file with 32 registers of 512-bit wide per HW thread context
- VPU can execute 16 SP fp Ops, 16 int32 Ops, or 8 DP fp Ops per cycle
- Most vector instructions have 4 clock latency with a 1 clock throughput
- At least two hardware contexts or threads must be run on each core, otherwise core utilization is < 50%

MIC Architecture

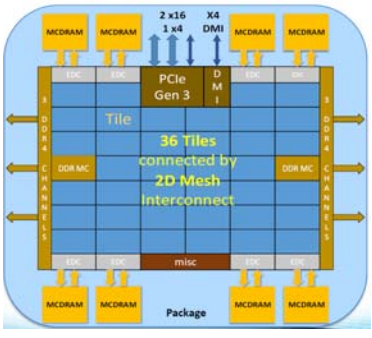


Ring topology

- Cores with private L1 caches
- One L2 cache per core
- Tag directories
- Memory controllers

Xeon Phi x200

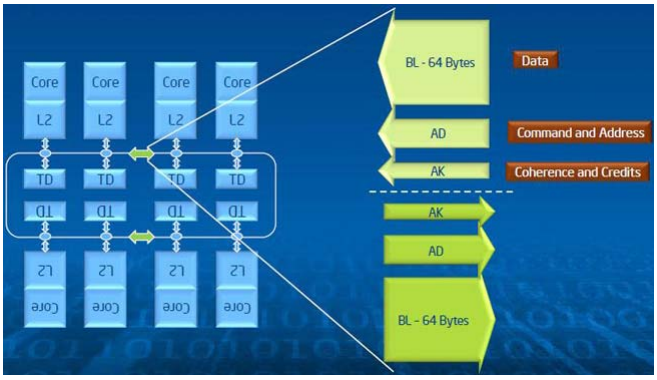
- > 8 billion transistors
- Out-of-order cores
- > 3 TFLOPS DP
- 8 banks with 16 GByte “near memory” and >400 GByte/s bw
- 6 DDR4 controllers for up to 384 GByte of “far memory” and ~90 GByte/s bw
- 2d mesh with tiles
  - 6x7 mesh
  - 36 tiles with 2 processors each
  - 2 extra tiles for spare processors
  - 2 tiles for memory controllers
  - 2 tiles for PCIe/DMI
- Processor core (modified Silvermont Atom core)
  - 1.2 – 1.3GHz, two AVX-512 vector units, and 32kB L1-caches
- Two cores of a tile share a L2-cache >30 MByte



x200 package including one Xeon Phi and 8 „near memory“ banks

The Interconnect

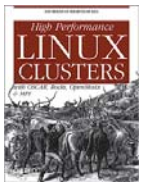
- Three independent bidirectional rings
  - **Data block** ring is 64 byte wide
  - **Address ring** is used to send read/write commands and memory addresses
  - **Acknowledgement ring** sends flow control and coherence messages



Bidirectional ring
- Max. distance N/2
- Average dist. N/4

Cluster-Computing

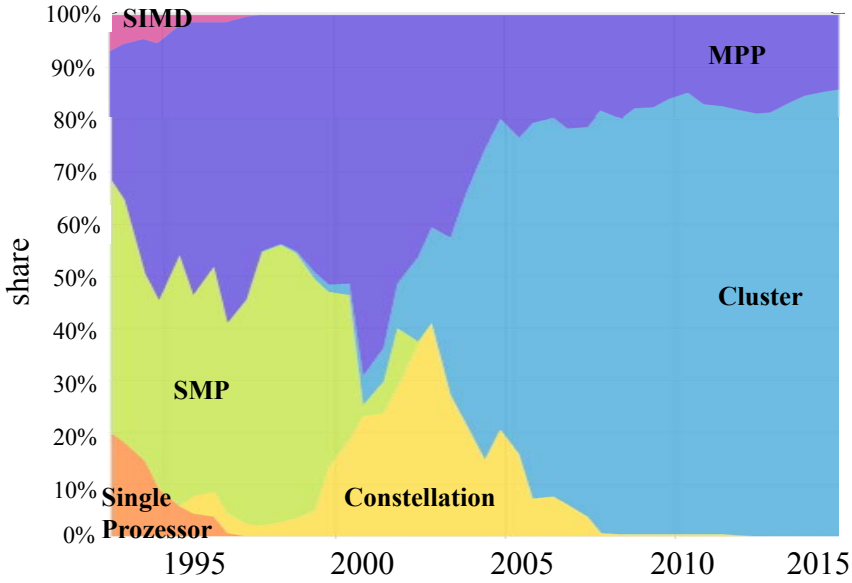
Literatur



- *High Performance Linux Clusters, Joseph D. Sloan, O´Reilly, 2005*

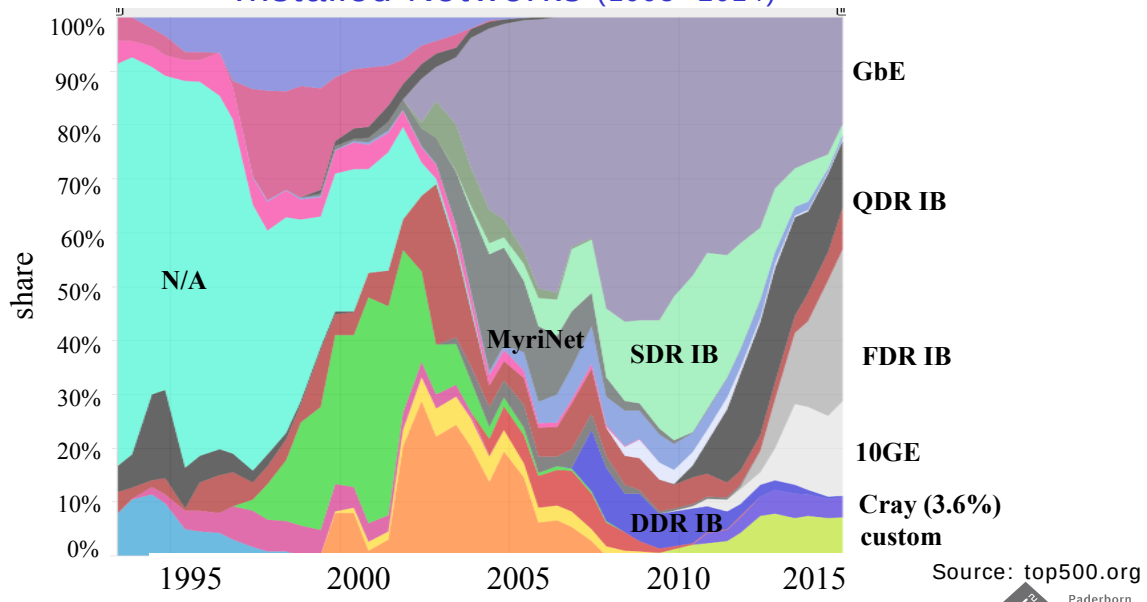
- *Aktuelle Bücher und zahlreiche Tutorials sind auch im Internet zu finden*

Architectural Trends (1993 -2014)



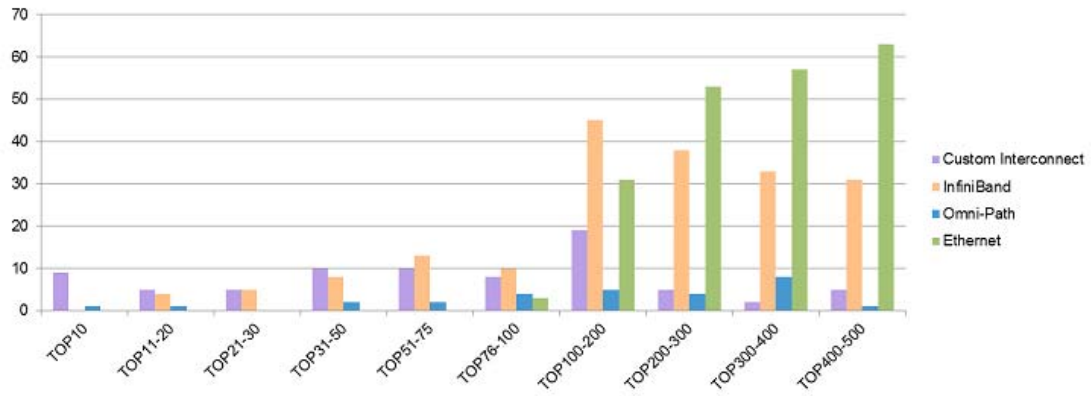
Source: top500.org

Installed Networks (1993 -2014)

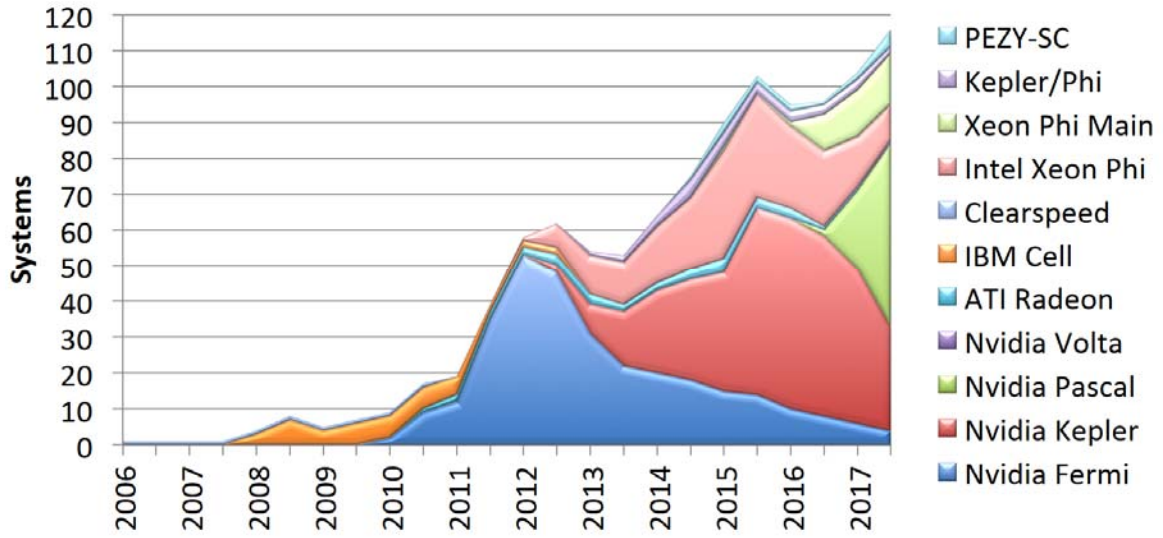


Source: top500.org

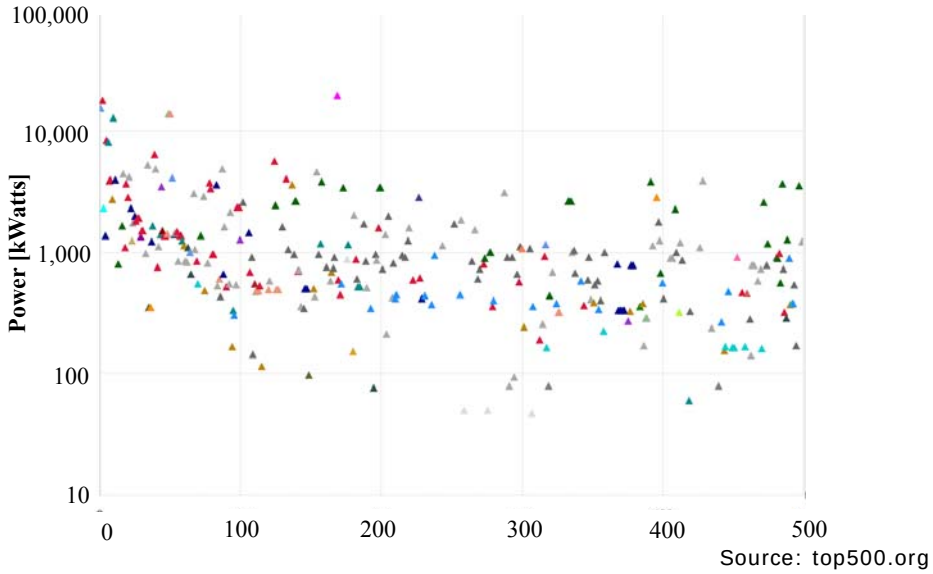
Network Type Distribution (11/2016)



Systems with Accelerators (11/2017)



Absolute Power Levels (11/2017)



Source: top500.org

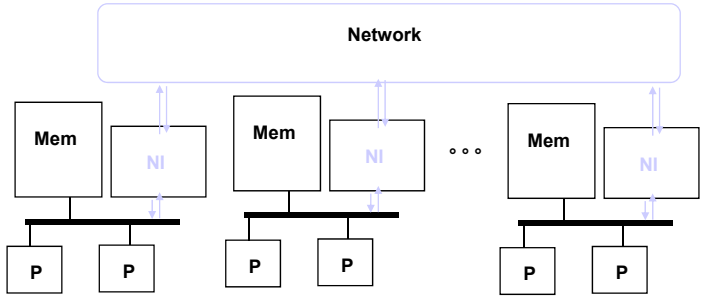
Bell‘s Law (1972)

- Gordon Bell beobachtete, dass IT-Entwicklungen in ungefähr 10-Jahreszyklen ablaufen
- Jede der Phasen benötigt ca. 10 Jahre
  - Erste Forschungsarbeiten
  - Erste Produkte und Verbesserungen
  - Allgemeine Nutzung
  - Auslaufen der Technologie
- Folgen HPC-Systeme ebenfalls diesem Gesetz?
  - Vector / SIMD
  - Custom Scalar
  - Commodity Cluster
  - Embedded / Accelerated
  - … was kommt als nächstes?

Ausnutzung von Parallelität

- Parallelität im Prozessor
  - Wortbreite
  - Fließbandverarbeitung bei der Instruktionsabarbeitung
  - Superskalare Prozessoren mit dynamischem Inst.-Scheduling
  - Very-Long-Instruktion-Word mit statischem Inst.-Scheduling
- Parallelität im Rechenknoten
  - Multi-Socket , Multi-Chip und Multi-Core
  - Gemeinsamer, cache-kohärenter Speicher
- Parallelität im Cluster aus Rechenknoten
  - Rechenknoten mit lokalen Speichern
  - Explizites Nachrichtenaustauschen zwischen Knoten

Cluster-Architektur



besteht aus

- Knoten; handelsübliche, vollwertige PCs
  - Durch Massenproduktion effizient in Preis/Leistung
  - Stets aktuellste Technologie nutzbar
  - Flexibel erweiterbar, erneuerbar, aufteilbar
- ohne Monitor, Tastatur, Maus, ...
- oftmals
  - SMP-Knoten
  - Linux auf jedem Knoten
  - explizites Hochgeschwindigkeitskommunikationsnetzwerk für MPI

Gründe der Akzeptanz

- „Killer Micros“
  - Verdopplung der Leistung alle 18-24 Monate (Moore‘s Law)
  - PCs werden immer kostengünstiger („Aldi-PC“)
  - Standards ermöglichen modulare Bauweise
- „Killer Networks“
  - Immer bessere Kommunikationsnetzwerktechnik (GbE, IBA, ...)
  - Globale Netzwerke nutzbar (Cloud-Computing)
- „Killer Software“
  - Open-Source-Software-Projekte für Cluster-Systeme
  - Offene Kommunikationsprotokolle (MPI)
  - Portabilität der Anwendungen durch optimierte Compiler, Bibliotheken, ...

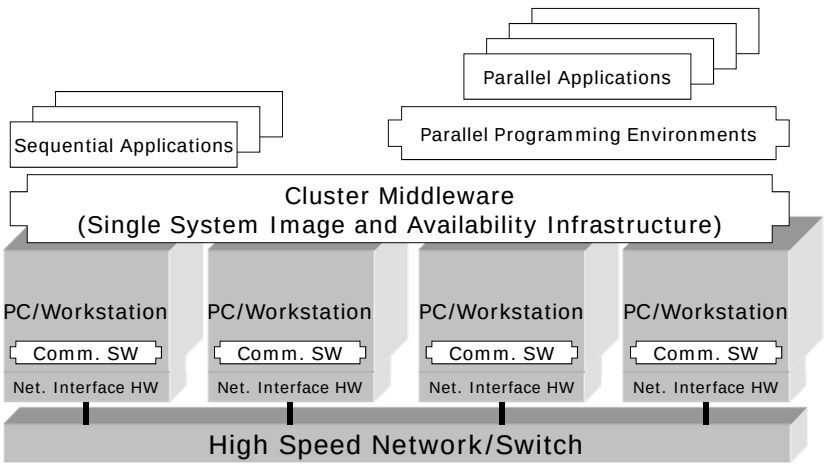
Einsatzbereiche

- Hochleistungsrechnen vs. Hochverfügbarkeit
- Kurze Antwortzeiten vs. Hoher Durchsatz
- Dedizierter Cluster vs. Ansammlung an PCs
- Homogene Knoten vs. heterogene Knoten
- Hochleistungskommunikationsnetzwerk vs. Standard Netzwerk
- Single-System-Image vs. einzeln administrierte PCs

Je nach Schwerpunkt, unterschiedliche Systemarchitektur notwendig.

Im weiteren Fokus sind die auf der linken Seite genannten Punkte.

Cluster-Architektur



Hochgeschwindigkeits-kommunikationsnetzwerke

Themen

- Netzwerktopologie
  - Anordnung der Leitungen und Schalter
  - Statisch vs. Dynamisch
- Vermittlungstechnik
  - Auswahl des Kommunikationswegs
  - Beschreibung von Kommunikationswegen
  - Eigenständige Schalter vs. Routen über Rechenknoten
- Netzwerkprotokolle
  - TCP/IP
  - MPI auf Hochgeschwindigkeitsnetzwerk
- Cluster-Kommunikationsnetzwerke
  - Beispiele: MyriNet, InfiniBand, ...

Bewertung von Netzwerken

- Netzwerk ist ein Graph G =(V,E)
  - V ist Menge der Knoten
  - E ist Menge an gerichtete/ungerichtete Verbindungen zwischen Knoten
- Kriterien eines Graphen
  - Durchmesser
  - Grad
  - Kantenkonnektivität
  - Bisektionsbreite
- Kriterien für ein Kommunikationsnetzwerk
  - Latenzzeit
  - Bandbreite
  - Verzögerung/Übertragungszeit
  - Durchsatz

Begriffe

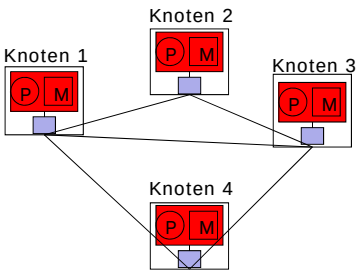
- Grad eines Knotens
  - Anzahl der Adjazentenkanten (ein- bzw. auslaufenden Kanten)
- Grad eines Netzwerks
  - Maximaler Knotengrad
- Distanz eines Knotenpaars
  - Länge des kürzesten Pfades zwischen beiden Knoten
  - Wie viele Knoten müssen mindestens passiert werden
- Durchmesser eines Graphen
  - Maximum der Distanzen aller Knotenpaare
- Kantenkonnektivität
  - Minimale Anzahl an Kanten, die aus dem Graphen entfernt werden müssen, um den Graphen in zwei Teilgraphen zu zerlegen
- Bisektionsbreite
  - Minimale Anzahl an Kanten, die aus dem Graphen entfernt werden müssen, um zwei Teilgraphen gleicher (±1) Größe zu zerlegen

Ziele

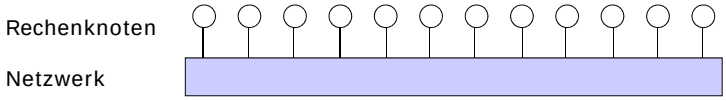
- Kleiner konstanter Grad
  - Kosten des Netzwerks proportional zu Anzahl Pins am Chip
  - Einfachere Skalierbarkeit
- Kleiner Durchmesser
  - Kurzer längster Weg
- Hohe Kantenkonnektivität
  - Mehrere unabhängige Pfade zwischen zwei Knoten
  - Alternative Wege zur Stauvermeidung und Ausfallsicherheit
- Hohe Bisektionsbreite
  - Ideal Knotenanzahl / 2

Direkte/Statische Verbindungsnetzwerke

- Knoten besteht aus Schalter und Prozessor
- jeder Knoten hat direkte Verbindung zu seinen benachbarten Knoten.
- Knoten ist Teil des Verbindungsnetzwerks
- Knoten entscheidet über Nutzung einer Verbindung.



Indirekte/Dynamische Verbindungsnetzwerke

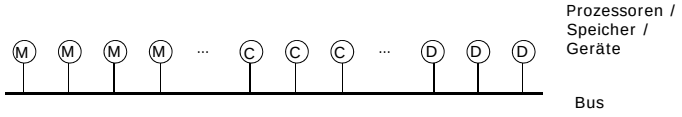


- Aufteilung in Rechenknoten und Netzwerkkomponenten
- Rechenknoten haben Verbindung zum gemeinsamen Netzwerk
- Netzwerk schaltet bei Bedarf Verbindungen

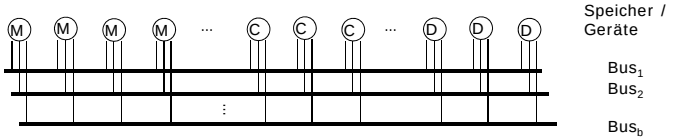
- Komponenten eines Schalters
  - Bus
  - Schaltmatrix
- Klassifizierung von Netzwerken
  - Einstufiges Netzwerk
  - Mehrstufiges Netzwerk

Bus

- Eigenschaften
  - blockierend, Shared-Medium
  - kostengünstig
  - sehr beschränkte Skalierbarkeit (kleine Anzahl an Knoten)
- Single-Bus



- Multi-Bus



Bewertung: Bus

- Bus
  - Grad = 1*
  - Durchmesser = 1
  - Kantenkonnektivität = 1*
  - Bisektionsbreite = 1
  - Beispiel: Ethernet
    - Dynamisches Netzwerk
    - CSMA/CD-Protokoll
    - Kein Routing innerhalb eines Segments
    - 1500 bis 9000 Byte Paketgrößen

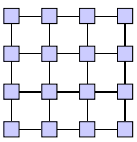
Bewertung: Clique

- Vollständiger Graph (Clique)
  - Grad = n-1
  - Durchmesser = 1
  - Kantenkonnektivität = n-1
  - Bisektionsbreite = $\lfloor n/2 \rfloor \lceil n/2 \rceil$
  - Eigenschaften:
    - Statisches Netzwerk
    - Jeder Knoten ist direkt mit jedem anderen verbunden
    - Kein Routing notwendig
    - Zu teuer für große Netzwerke

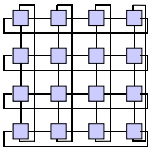
Bewertung: Ring

- Ring
  - Grad = 2
  - Durchmesser = $\lfloor n/2 \rfloor$
  - Kantenkonnektivität = 2
  - Bisektionsbreite = 2
  - Eigenschaften:
    - Statisches Netzwerk
    - Jeder Knoten i ist mit Knoten $(i-1)_{\text{mod } n}$ und $(i+1)_{\text{mod } n}$ verbunden
    - Einfaches Routing
    - Geringe Kosten für große Netzwerke
    - Schlechte Skalierung der Leistung

Gitterstrukturen



4 x 4 Gitter



4 x 4 Torus

Pro

$(\sqrt{n})$

- konstanter Knotengrad
- Erweiterbarkeit in kleinen Inkrementen
- gute Unterstützung von Algorithmen mit lokaler Kommunikationsstruktur (z.B. Modellierung physikalischer Prozesse)

Kontra

- große Netzwerke mit relativ hohem Durchmesser

Bewertung: Gitter

- d-dimensionales Gitter
  - Grad = 2d
  - Durchmesser = $d \cdot \left(\sqrt[d]{n} - 1\right)$
  - Kantenkonnektivität = d
  - Bisektionsbreite = $\left(\sqrt[d]{n}\right)^{d-1}$
  - Eigenschaften:
    - Statisches Netzwerk
    - Hohe Kosten für Netzwerke mit hoher Dimension
    - Relativ gute Skalierung der Leistung

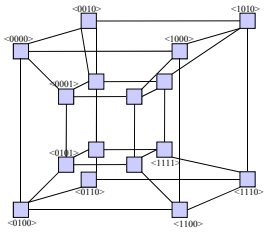
Bewertung: Torus

- d-dimensionales Torus
  - Grad = 2d
  - Durchmesser = $\frac{d}{2} \cdot \left(\sqrt[d]{n} - 1\right)$
  - Kantenkonnektivität = 2d
  - Bisektionsbreite = $2 \cdot \left(\sqrt[d]{n}\right)^{d-1}$
  - Eigenschaften:
    - Statisches Netzwerk
    - Hohe Kosten für Netzwerke mit hoher Dimension
    - Relativ gute Skalierung der Leistung

Hypercube

- Schalter <w> ist mit Schalter <w‘> verbunden, genau dann, wenn sich w‘ nur in einem Bit von w unterscheidet.
- Variabler Knotengrad
- Nur erweiterbar in Zweierpotenzen

- Beispiel: HQ(4)

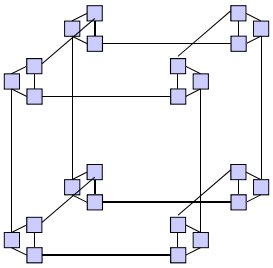


Bewertung: Hypercube

- Hyperwürfel (Hypercube)
  - Dimension k, $n = 2^k$
  - Grad = k
  - Durchmesser = k
  - Kantenkonnektivität = k
  - Bisektionsbreite = n/2
  - Eigenschaften:
    - Statisches Netzwerk
    - k unabängige Pfade zwischen zwei Knoten
    - HQ(k) besteht aus zwei HQ(k-1)

Würfelartig verbundene Ringe (CCC)

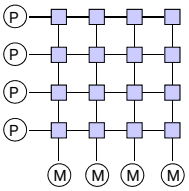
D-dimensionaler Hypercube, bei der jeder Knoten durch einen Ring der Länge d ersetzt wird. Jeder der d Knoten eines Rings besitzt neben den beiden Ringkanten eine Kante in Richtung einer der d-Hypecube-Dimensionen.



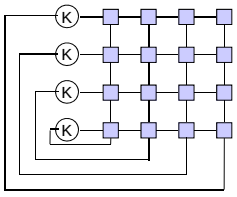
Cube Connected Cycles der Dimension 3 - CCC(3)

- Pro
  - konstanter Knotengrad von 3
  - nur in diskreten Schritten erweiter $\left(d \cdot 2^d\right)$
  - logarithmischer Durchmesser $\left(2d + \lfloor d/2 \rfloor\right)$

Kreuzschienenverteiler

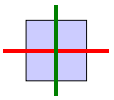


4 x 4 Schaltmatrix

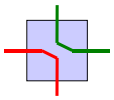


4 x 4 Schalter

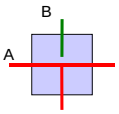
Schalterzustände:



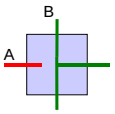
Durchschaltung



Kreuzschaltung



Fan-Out A

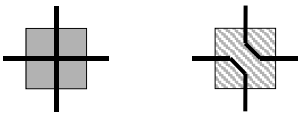


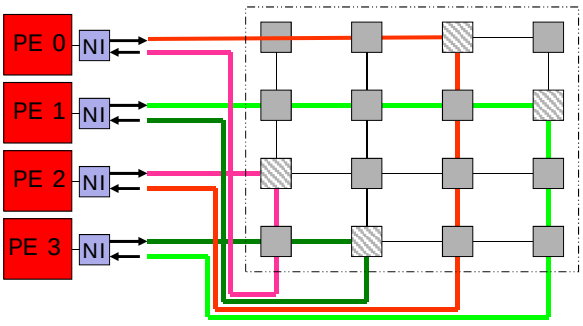
Fan-Out B

- n^2 interne Schalter
- Blockierungsfrei

Kreuzschienenverteiler

Basisschalter:





Bewertung: Crossbar

- Kreuzschienenverteiler (Crossbar)
  - Grad = 1*
  - Durchmesser = 1*
  - Kantenkonnektivität = ?
  - Bisektionsbreite = n/2
  - Eigenschaften:
    - Dynamisches Netzwerk, einstufig
    - Sehr hohe Leistung
    - Hohe Kosten

*als ein Schalter betrachtet. Intern konstanten Grad und mindestens O(log n) Durchmesser.

Mehrstufige Netzwerke

- Schaltmatrizen als Basiselemente
- Mehrere Ebenen als Schalter bilden Netzwerk
- Eigenschaften
  - Skalierbare Netzwerke
  - Blockierungen im Netzwerk möglich

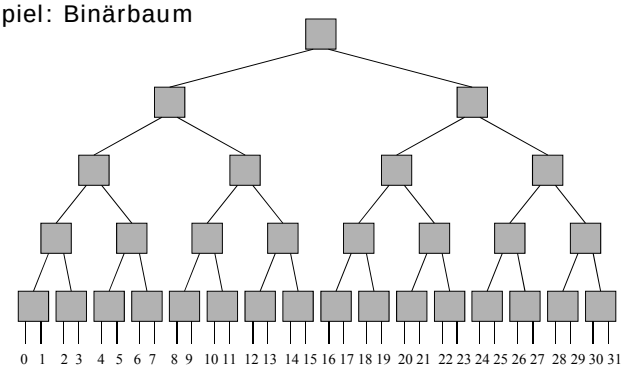
- Kleine Schaltmatrizen häufig verwendet (4x4, 8x8 bis 32x32)

Topologie: Baum

- k-närer Baum mit $\sum_{i=1}^{\log_k n} n/k^i$ Schaltern ((k+1)-wege)

- logarithmischer Durchmesser ($2*\log_k n - 1$)
- Wurzel ist ein Engpass (Bisektionsweite 1)

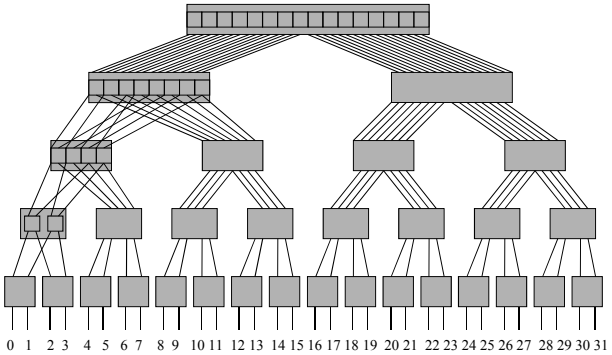
- Beispiel: Binärbaum



Topologie: Fetter Baum (Fat-Tree)

- n Knoten, **$\log_k n$ Level**
- Schalter mit Grad 2k: je k Verbindung in Level i-1 und in Level i+1
- Jeder Level mit **n / k Schaltern**

- Beispiel: k=2

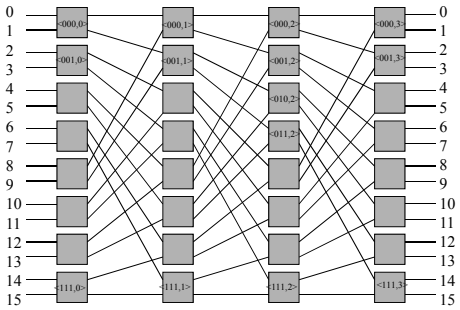


Bewertung: Fat-Tree

- Fat-Tree/Clos
  - „Baum“ mit n Knoten und n/2 $\log_k n$ Schaltern
  - Grad = 2k
  - Durchmesser = 2 $\log_k n$ - 1
  - Kantenkonnektivität = k
  - Bisektionsbreite = n/2
  - Eigenschaften:
    - Dynamisches Netzwerk, mehrstufig
    - Auch einstufige Variante mit $\log_k n$ Durchläufen möglich

Topologie: Omega-Netzwerk

- Schalter <w,l> ist mit Schalter <w_{lcs},l+1> und <w_{lcsbf},l+1> verbunden, wobei w_{lcs} durch *left cyclic shift* aus w entsteht und w_{lcsbf} aus w_{lcs} mit zusätzlichem *bit flip* des letzten Bits entsteht
- eindeutiger Pfad von jedem Eingang zu jedem Ausgang

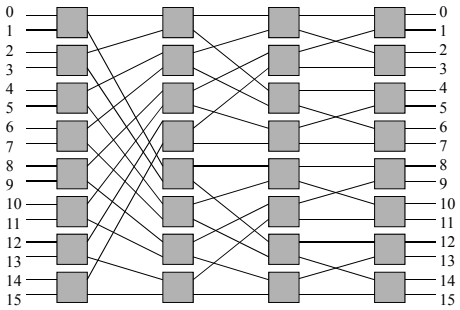


Bewertung: Omega-Netzwerk

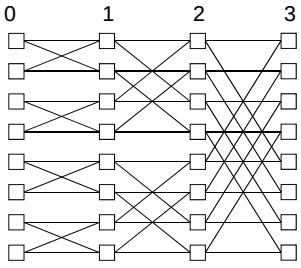
- Omega-Netzwerk
  - n Knoten, (n/2) $\log_2 n$ Crossbars
  - Grad = 2 (Crossbars mit Grad 4)
  - Durchmesser = $\log_2 n$
  - Kantenkonnektivität = 2
  - Bisektionsbreite = n/2
  - Eigenschaften:
    - Dynamisches Netzwerk, mehrstufig
    - Auch einstufige Variante mit $\log_2 n$ Durchläufen möglich

Topologie: Clos

- kleine Gruppen aus Knoten lokal vernetzbar mit kurzen Verbindungen
- gut im Raum platzierbar, wenige kreuzende Verbindungen



Butterfly



2-nary 4-fly

Butterfly der Dimension 3 - Bn(3)

Pro
- konstanter Knotengrad (= 4)
- logarithmischer Durchmesser

Cons
- Erweiterbarkeit nur in bestimmten Größen (d 2^d)

Zusammenfassung der Eigenschaften

	Anzahl Knoten	Anzahl Kanten	maximaler Knotengrad	Durch-messer	Bisektions-weite
Clique	n	$\frac{n\cdot(n-1)}{2}$	$n-1$	1	$\frac{n\cdot(n-1)}{4}$
Ring	n	n	2	$\lfloor n/2 \rfloor$	2
Gitter $G(a_1 x\ a_2 \ldots x\ a_d)$	$\prod_{k=1}^{d} a_k$	$\sum_{k=1}^{d}(a_k-1)\prod_{i\neq k} a_i$	$2d$	$\sum_{k=1}^{d}(a_k-1)$	$\min_{k=1}^{d} a_k$
Torus $T(a_1 x\ a_2 \ldots x\ a_d)$	$\prod_{k=1}^{d} a_k$	$d\sum_{k=1}^{d} a_k$	$2d$	$\sum_{k=1}^{d}\lfloor a_k/2 \rfloor$	$2\cdot\min_{k=1}^{d} a_k$
Fat-Tree level l, switch 2k	k^l	$l\cdot k^l$	$2k$	$2l-1$	k^{l-1}
Hypecube HQ(d)	2^d	$d\cdot 2^{d-1}$	d	d	2^{d-1}
Omega(d)	$d\cdot 2^d$	$d\cdot 2^{d+1}$	4	d	2^d
CCC(d)	$d\cdot 2^d$	$3\cdot d\cdot 2^{d-1}$	3	$2d+\lfloor d/2 \rfloor$	2^{d-1}
Butterfly Bn(d)	$d\cdot 2^d$	$d\cdot 2^{d+1}$	4	d	2^d