

Lecture: Networks, Disks, Datacenters, GPUs

- Topics: networks wrap-up, disks and reliability, datacenters, GPU intro (Sections 6.1-6.7, App D, Ch 4)

Packets/Flits

- A message is broken into multiple packets (each packet has header information that allows the receiver to re-construct the original message)
- A packet may itself be broken into flits – flits do not contain additional headers
- Two packets can follow different paths to the destination
Flits are always ordered and follow the same path
- Such an architecture allows the use of a large packet size (low header overhead) and yet allows fine-grained resource allocation on a per-flit basis

Flow Control

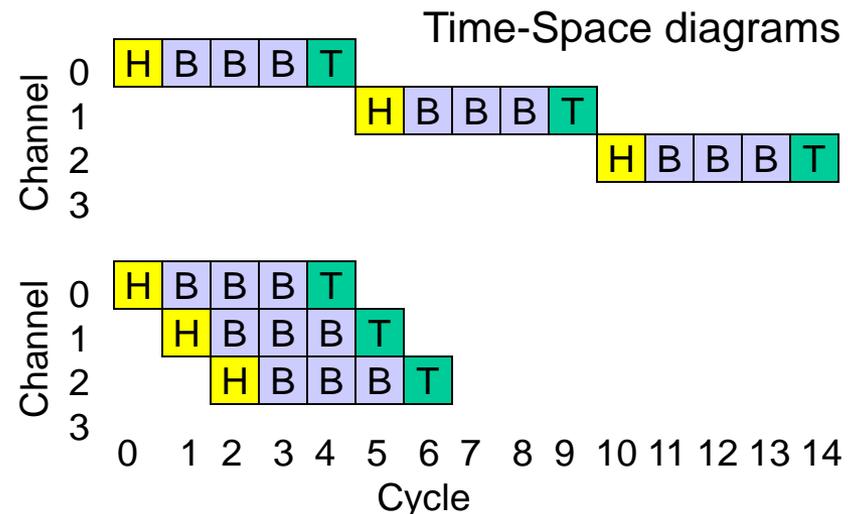
- The routing of a message requires allocation of various resources: the channel (or link), buffers, control state
- Bufferless: flits are dropped if there is contention for a link, NACKs are sent back, and the original sender has to re-transmit the packet
- Circuit switching: a request is first sent to reserve the channels, the request may be held at an intermediate router until the channel is available (hence, not truly bufferless), ACKs are sent back, and subsequent packets/flits are routed with little effort (good for bulk transfers)

Buffered Flow Control

- A buffer between two channels decouples the resource allocation for each channel
- Packet-buffer flow control: channels and buffers are allocated per packet

- Store-and-forward

- Cut-through



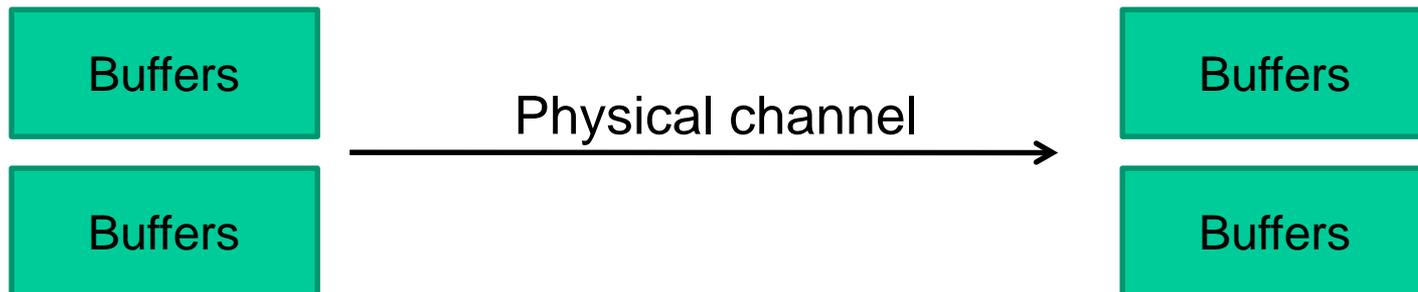
- Wormhole routing: same as cut-through, but buffers in each router are allocated on a per-flit basis, not per-packet

Virtual Channels



Flits do not carry headers. Once a packet starts going over a channel, another packet cannot cut in (else, the receiving buffer will confuse the flits of the two packets). If the packet is stalled, other packets can't use the channel.

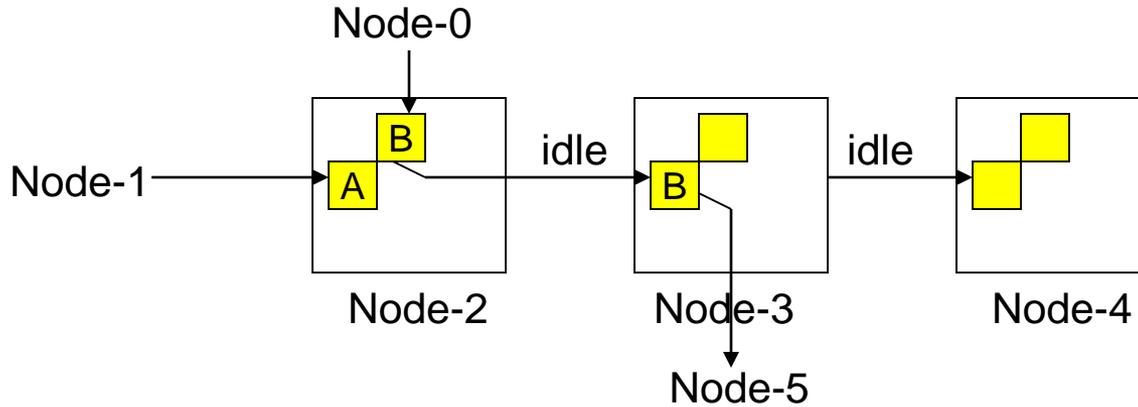
With virtual channels, the flit can be received into one of N buffers. This allows N packets to be in transit over a given physical channel. The packet must carry an ID to indicate its virtual channel.



Example

- **Wormhole:**

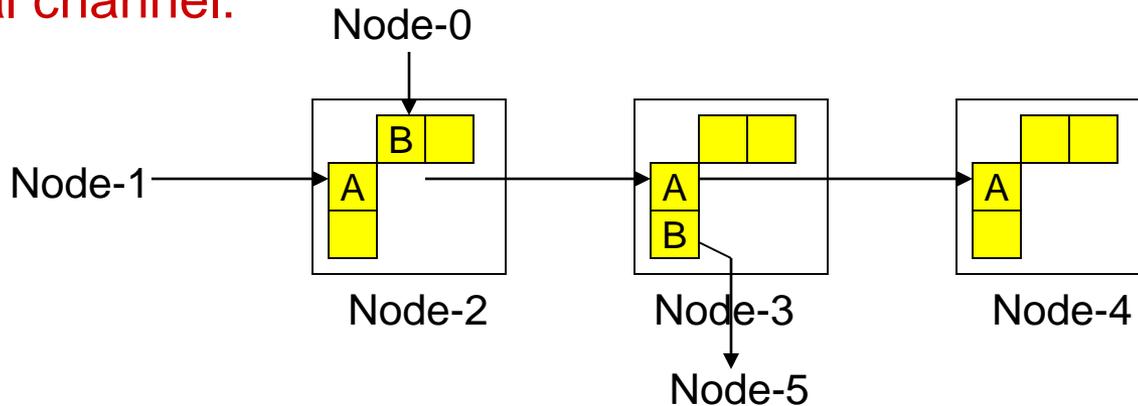
A is going from Node-1 to Node-4; B is going from Node-0 to Node-5



(blocked, no free VCs/buffers)

Traffic Analogy:
B is trying to make a left turn; A is trying to go straight; there is no left-only lane with wormhole, but there is one with VC

- **Virtual channel:**



(blocked, no free VCs/buffers)

Virtual Channel Flow Control

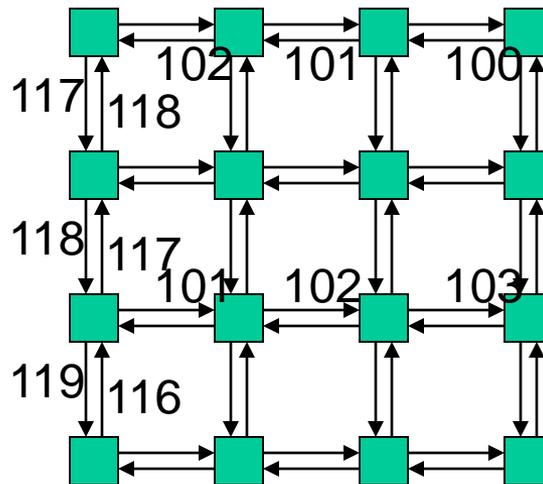
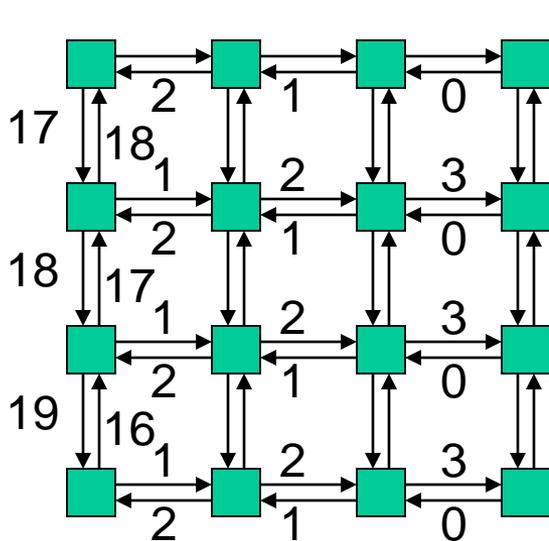
- Incoming flits are placed in buffers
- For this flit to jump to the next router, it must acquire three resources:
 - A free virtual channel on its intended hop
 - We know that a virtual channel is free when the tail flit goes through
 - Free buffer entries for that virtual channel
 - This is determined with credit or on/off management
 - A free cycle on the physical channel
 - Competition among the packets that share a physical channel

Buffer Management

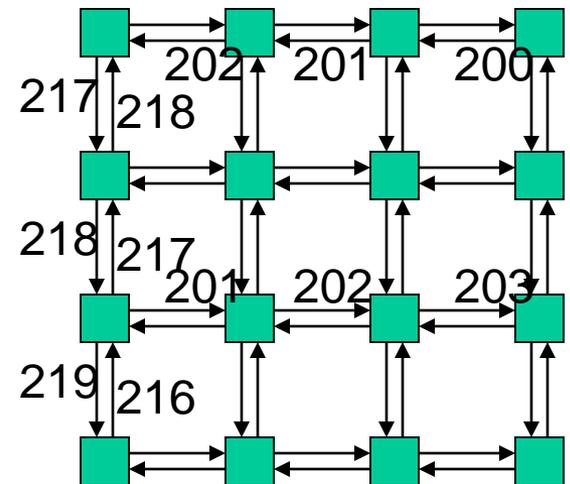
- Credit-based: keep track of the number of free buffers in the downstream node; the downstream node sends back signals to increment the count when a buffer is freed; need enough buffers to hide the round-trip latency
- On/Off: the upstream node sends back a signal when its buffers are close to being full – reduces upstream signaling and counters, but can waste buffer space

Deadlock Avoidance with VCs

- VCs provide another way to number the links such that a route always uses ascending link numbers



- Alternatively, use West-first routing on the 1st plane and cross over to the 2nd plane in case you need to go West again (the 2nd plane uses North-last, for example)



Router Functions

- Crossbar, buffer, arbiter, VC state and allocation, buffer management, ALUs, control logic, routing
- Typical on-chip network power breakdown:
 - 30% link
 - 30% buffers
 - 30% crossbar

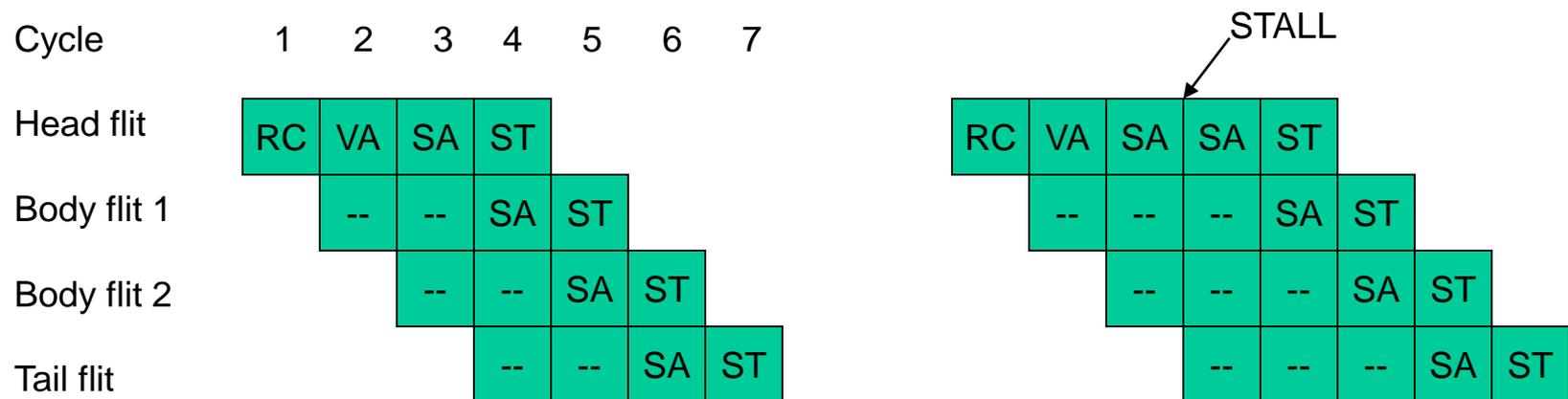
Router Pipeline

- Four typical stages:
 - RC routing computation: the head flit indicates the VC that it belongs to, the VC state is updated, the headers are examined and the next output channel is computed (note: this is done for all the head flits arriving on various input channels)
 - VA virtual-channel allocation: the head flits compete for the available virtual channels on their computed output channels
 - SA switch allocation: a flit competes for access to its output physical channel
 - ST switch traversal: the flit is transmitted on the output channel

A head flit goes through all four stages, the other flits do nothing in the first two stages (this is an in-order pipeline and flits can not jump ahead), a tail flit also de-allocates the VC

Router Pipeline

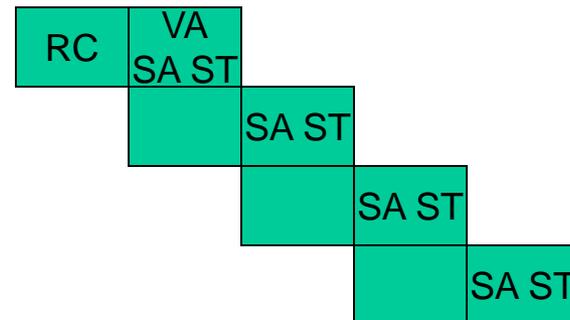
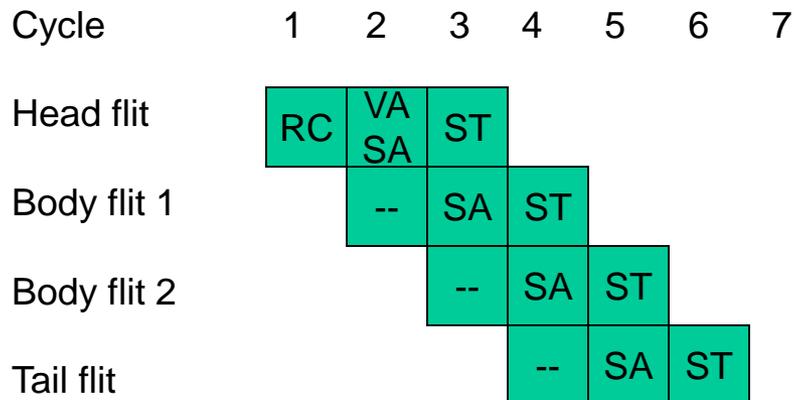
- Four typical stages:
 - RC routing computation: compute the output channel
 - VA virtual-channel allocation: allocate VC for the head flit
 - SA switch allocation: compete for output physical channel
 - ST switch traversal: transfer data on output physical channel



Speculative Pipelines

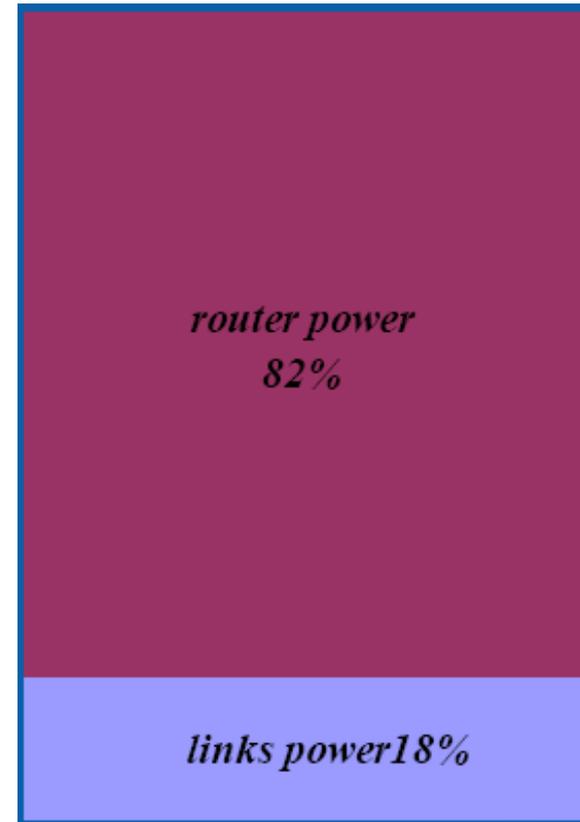
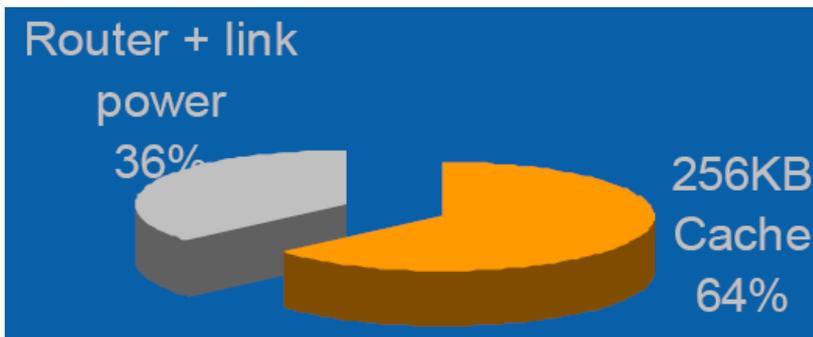
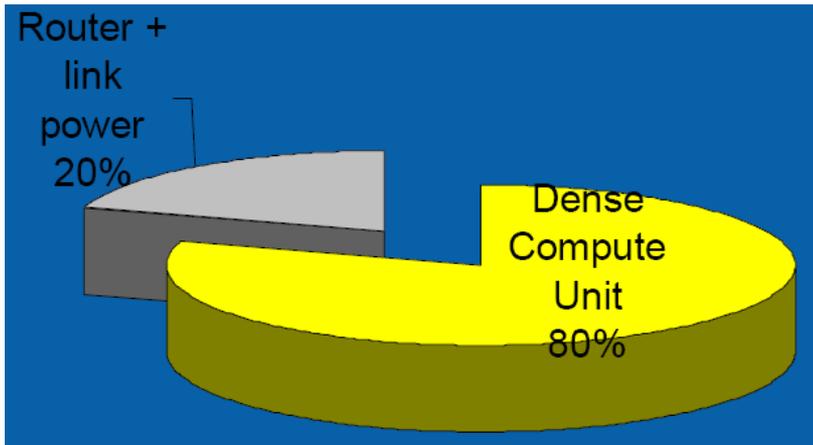
- Perform VA and SA in parallel
- Note that SA only requires knowledge of the output physical channel, not the VC
- If VA fails, the successfully allocated channel goes un-utilized

- Perform VA, SA, and ST in parallel (can cause collisions and re-tries)
- Typically, VA is the critical path – can possibly perform SA and ST sequentially



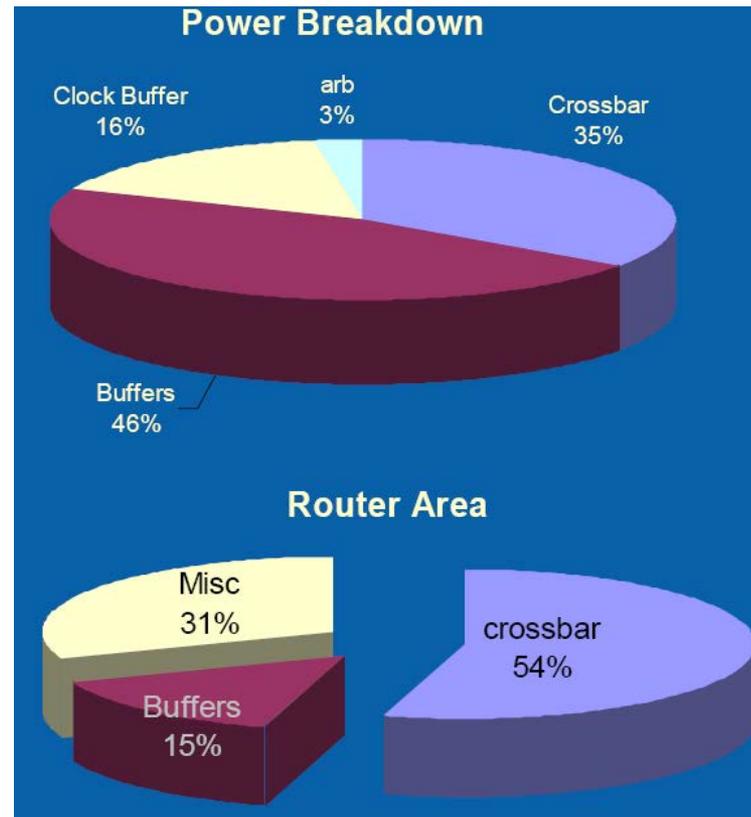
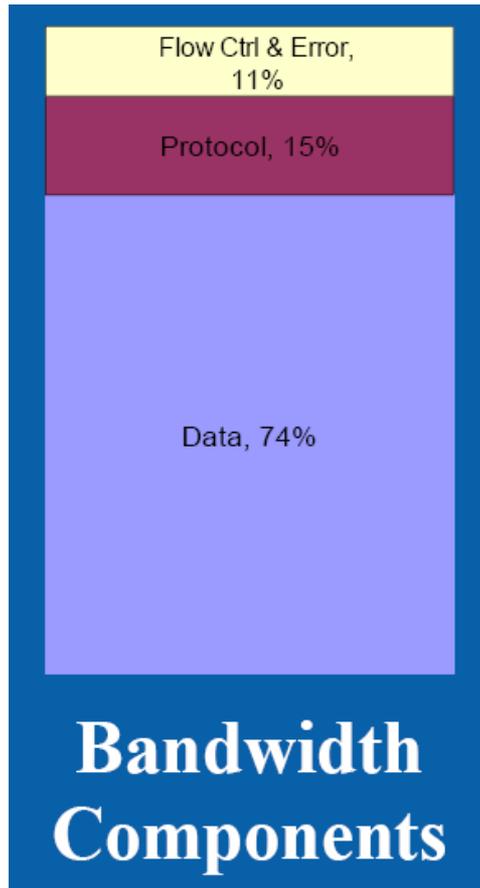
- Router pipeline latency is a greater bottleneck when there is little contention
- When there is little contention, speculation will likely work well!
- Single stage pipeline?

Example Intel Router



Source: Partha Kundu, "On-Die Interconnects for Next-Generation CMPs", talk at On-Chip Interconnection Networks Workshop, Dec 2006

Example Intel Router



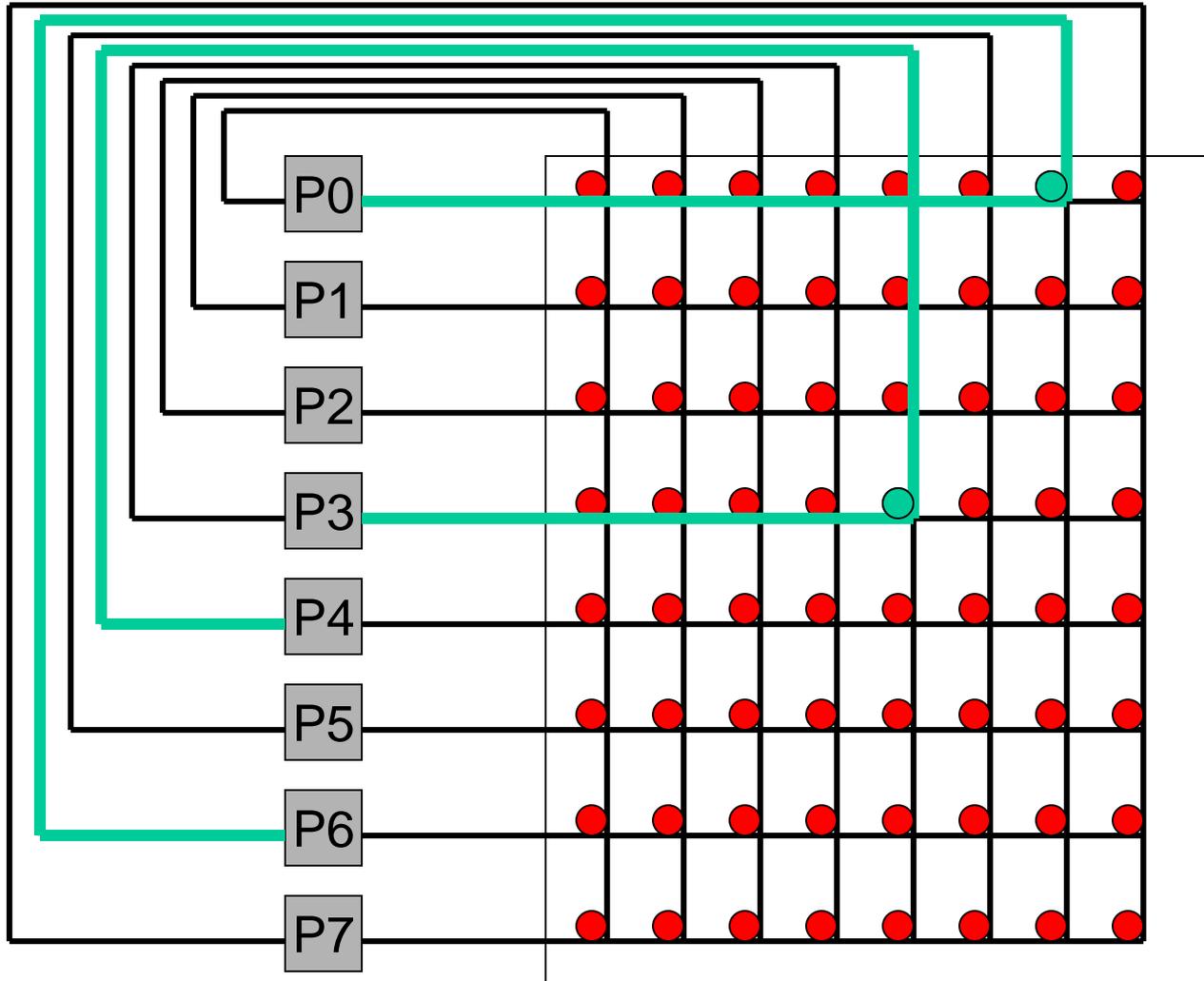
- Used for a 6x6 mesh
- 16 B, > 3 GHz
- Wormhole with VC flow control

Source: Partha Kundu, "On-Die Interconnects for Next-Generation CMPs", talk at On-Chip Interconnection Networks Workshop, Dec 2006

Current Trends

- Growing interest in eliminating the area/power overheads of router buffers; traffic levels are also relatively low, so virtual-channel buffered routed networks may be overkill
- Option 1: use a bus for short distances (16 cores) and use a hierarchy of buses to travel long distances
- Option 2: hot-potato or bufferless routing

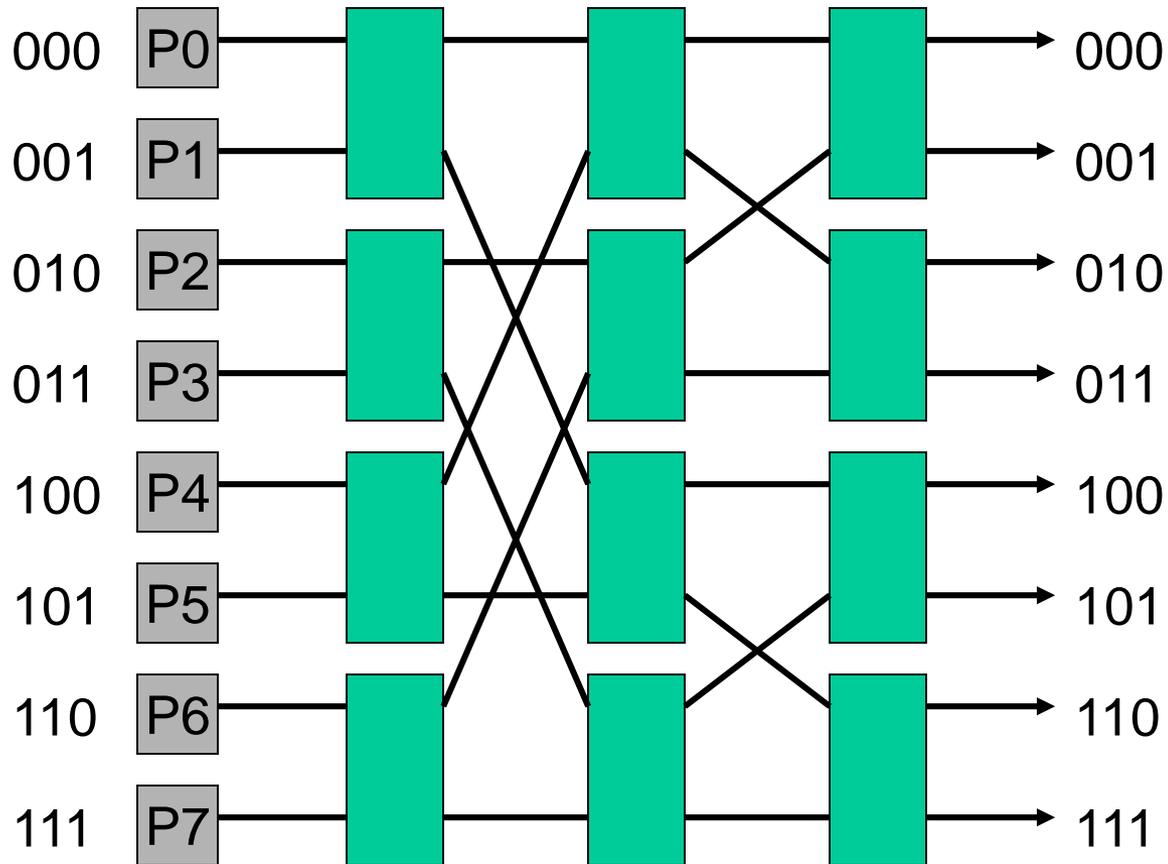
Centralized Crossbar Switch



Crossbar Properties

- Assuming each node has one input and one output, a crossbar can provide maximum bandwidth: N messages can be sent as long as there are N unique sources and N unique destinations
- Maximum overhead: WN^2 internal switches, where W is data width and N is number of nodes
- To reduce overhead, use smaller switches as building blocks – trade off overhead for lower effective bandwidth

Switch with Omega Network

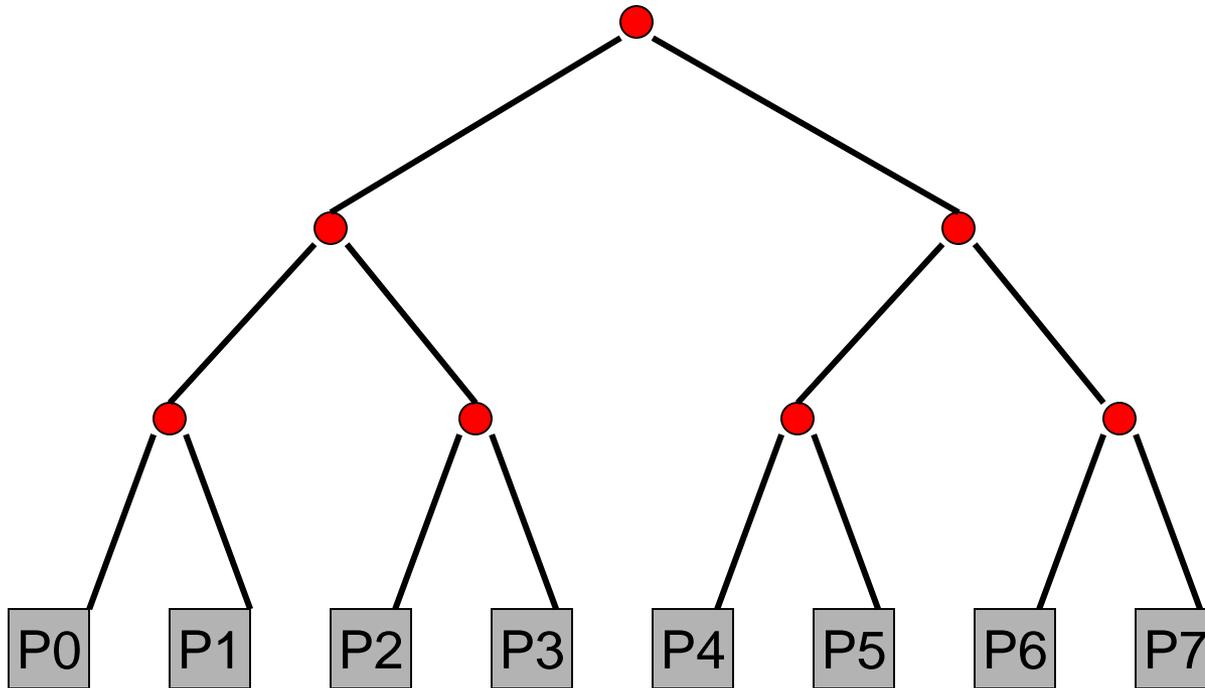


Omega Network Properties

- The switch complexity is now $O(N \log N)$
- Contention increases: $P_0 \rightarrow P_5$ and $P_1 \rightarrow P_7$ cannot happen concurrently (this was possible in a crossbar)
- To deal with contention, can increase the number of levels (redundant paths) – by mirroring the network, we can route from P_0 to P_5 via N intermediate nodes, while increasing complexity by a factor of 2

Tree Network

- Complexity is $O(N)$
- Can yield low latencies when communicating with neighbors
- Can build a fat tree by having multiple incoming and outgoing links

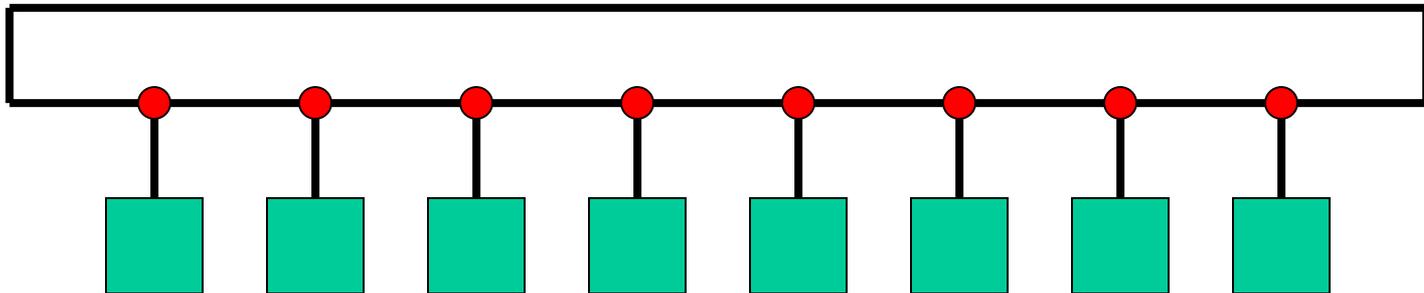


Bisection Bandwidth

- Split N nodes into two groups of $N/2$ nodes such that the bandwidth between these two groups is minimum: that is the bisection bandwidth
- Why is it relevant: if traffic is completely random, the probability of a message going across the two halves is $\frac{1}{2}$ – if all nodes send a message, the bisection bandwidth will have to be $N/2$
- The concept of bisection bandwidth confirms that the tree network is not suited for random traffic patterns, but for localized traffic patterns

Distributed Switches: Ring

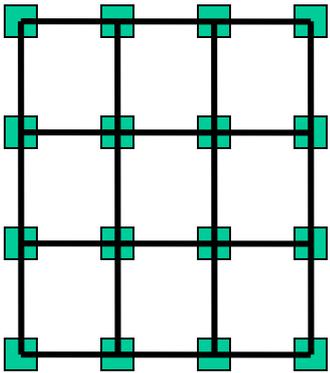
- Each node is connected to a 3x3 switch that routes messages between the node and its two neighbors
- Effectively a repeated bus: multiple messages in transit
- Disadvantage: bisection bandwidth of 2 and $N/2$ hops on average



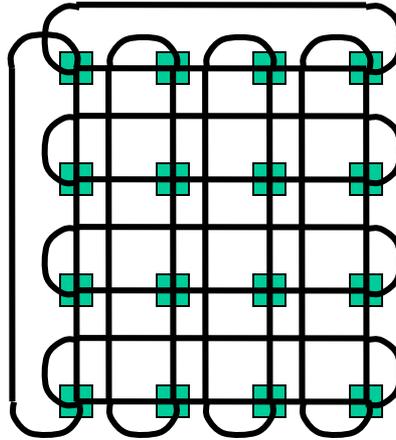
Distributed Switch Options

- Performance can be increased by throwing more hardware at the problem: fully-connected switches: every switch is connected to every other switch: N^2 wiring complexity, $N^2 / 4$ bisection bandwidth
- Most commercial designs adopt a point between the two extremes (ring and fully-connected):
 - Grid: each node connects with its N, E, W, S neighbors
 - Torus: connections wrap around
 - Hypercube: links between nodes whose binary names differ in a single bit

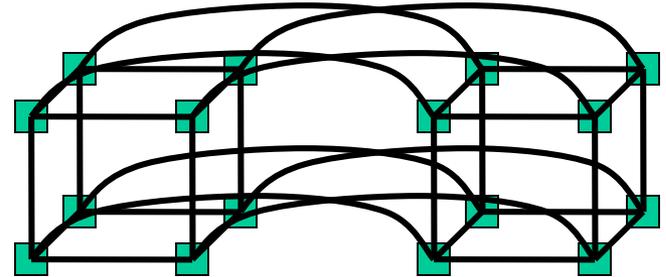
Topology Examples



Grid



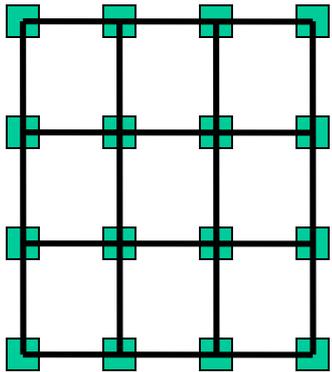
Torus



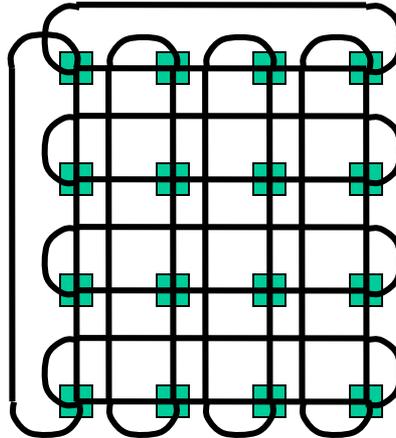
Hypercube

Criteria	Bus	Ring	2Dtorus	6-cube	Fully connected
64 nodes					
Performance Bisection bandwidth					
Cost Ports/switch Total links					

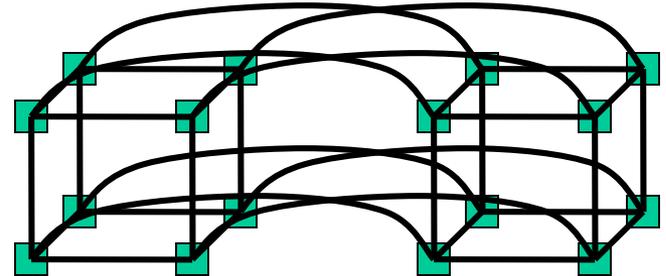
Topology Examples



Grid



Torus



Hypercube

Criteria	Bus	Ring	2Dtorus	6-cube	Fully connected
64 nodes					
Performance					
Diameter	1	32	8	6	1
Bisection BW	1	2	16	32	1024
Cost					
Ports/switch		3	5	7	64
Total links	1	64	128	192	2016

k-ary d-cube

- Consider a k-ary d-cube: a d-dimension array with k elements in each dimension, there are links between elements that differ in one dimension by 1 (mod k)
- Number of nodes $N = k^d$

Number of switches :

Switch degree :

Number of links :

Pins per node :

Avg. routing distance:

Diameter :

Bisection bandwidth :

Switch complexity :

Should we minimize or maximize dimension?

k-ary d-Cube

- Consider a k-ary d-cube: a d-dimension array with k elements in each dimension, there are links between elements that differ in one dimension by 1 (mod k)

- Number of nodes $N = k^d$

Number of switches	: N	Avg. routing distance:	$d(k-1)/4$
Switch degree	: $2d + 1$	Diameter	: $d(k-1)/2$
Number of links	: Nd	Bisection bandwidth	: $2wk^{d-1}$
Pins per node	: $2wd$	Switch complexity	: $(2d + 1)^2$

The switch degree, num links, pins per node, bisection bw for a hypercube are half of what is listed above (diam and avg routing distance are twice, switch complexity is $(d + 1)^2$) because unlike₂₈ the other cases, a hypercube does not have right and left neighbors.

Should we minimize or maximize dimension?

Warehouse-Scale Computer (WSC)

- 100K+ servers in one WSC
- ~\$150M overall cost
- Requests from millions of users (Google, Facebook, etc.)
- Cloud Computing: a model where users can rent compute and storage within a WSC, there's an associated service-level agreement (SLA)
- Datacenter: a collection of WSCs in a single building, possibly belonging to different clients and using different hardware/architecture

Workloads

- Typically, software developed in-house – MapReduce, BigTable, Memcached, etc.
- MapReduce: embarrassingly parallel operations performed on very large datasets, e.g., organize data into clusters, aggregate a count over several documents
- Hadoop is an open-source implementation of the MapReduce framework; makes it easy for users to write MapReduce programs without worrying about low-level task/data management

MapReduce

- Application-writer provides Map and Reduce functions that operate on key-value pairs
- Each map function operates on a collection of records; a record is (say) a webpage or a facebook user profile
- The records are in the file system and scattered across several servers; thousands of map functions are spawned to work on all records in parallel
- The Reduce function aggregates and sorts the results produced by the Mappers, also performed in parallel

MR Framework Duties

- Replicate data for fault tolerance
- Detect failed threads and re-start threads
- Handle variability in thread response times
- Use of MR within Google has been growing every year:
Aug'04 → Sep'09
 - Number of MR jobs has increased 100x+
 - Data being processed has increased 100x+
 - Number of servers per job has increased 3x

WSC Hierarchy

- A rack can hold 48 1U servers (1U is 1.75 inches high)
- A rack switch is used for communication within and out of a rack; an array switch connects an array of racks
- Latency grows if data is fetched from remote DRAM or disk (300us vs. 0.1us for DRAM and 12ms vs. 10ms for disk)
- Bandwidth within a rack is much higher than between arrays; hence, software must be aware of data placement and locality

PUE Metric and Power Breakdown

- PUE = Total facility power / IT equipment power
(power utilization effectiveness)
- It is greater than 1; ranges from 1.33 to 3.03, median of 1.69
- The cooling power is roughly half the power used by servers
- Within a server (circa 2007), the power distribution is as follows: Processors (33%), DRAM memory (30%), Disks (10%), Networking (5%), Miscellaneous (22%)

CapEx and OpEx

- Capital expenditure: infrastructure costs for the building, power delivery, cooling, and servers
- Operational expenditure: the monthly bill for energy, failures, personnel, etc.
- CapEx can be amortized into a monthly estimate by assuming that the facilities will last 10 years, server parts will last 3 years, and networking parts will last 4

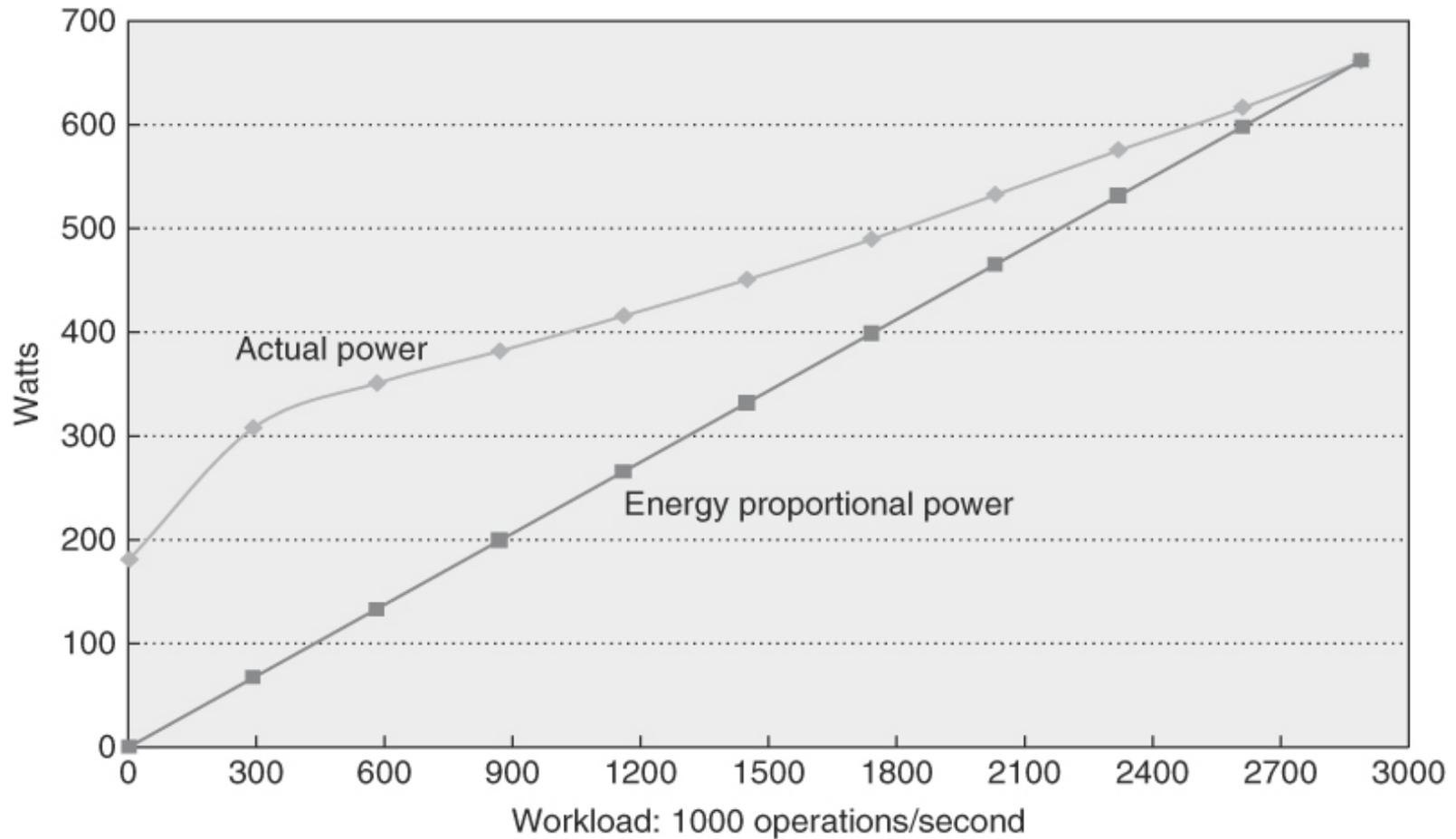
CapEx/OpEx Case Study

- 8 MW facility : facility cost: \$88M, server/networking cost: \$79M
- Monthly expense: \$3.8M. Breakdown:
 - Servers 53% (amortized CapEx)
 - Networking 8% (amortized CapEx)
 - Power/cooling infrastructure 20% (amortized CapEx)
 - Other infrastructure 4% (amortized CapEx)
 - Monthly power bill 13% (true OpEx)
 - Monthly personnel salaries 2% (true OpEx)

Improving Energy Efficiency

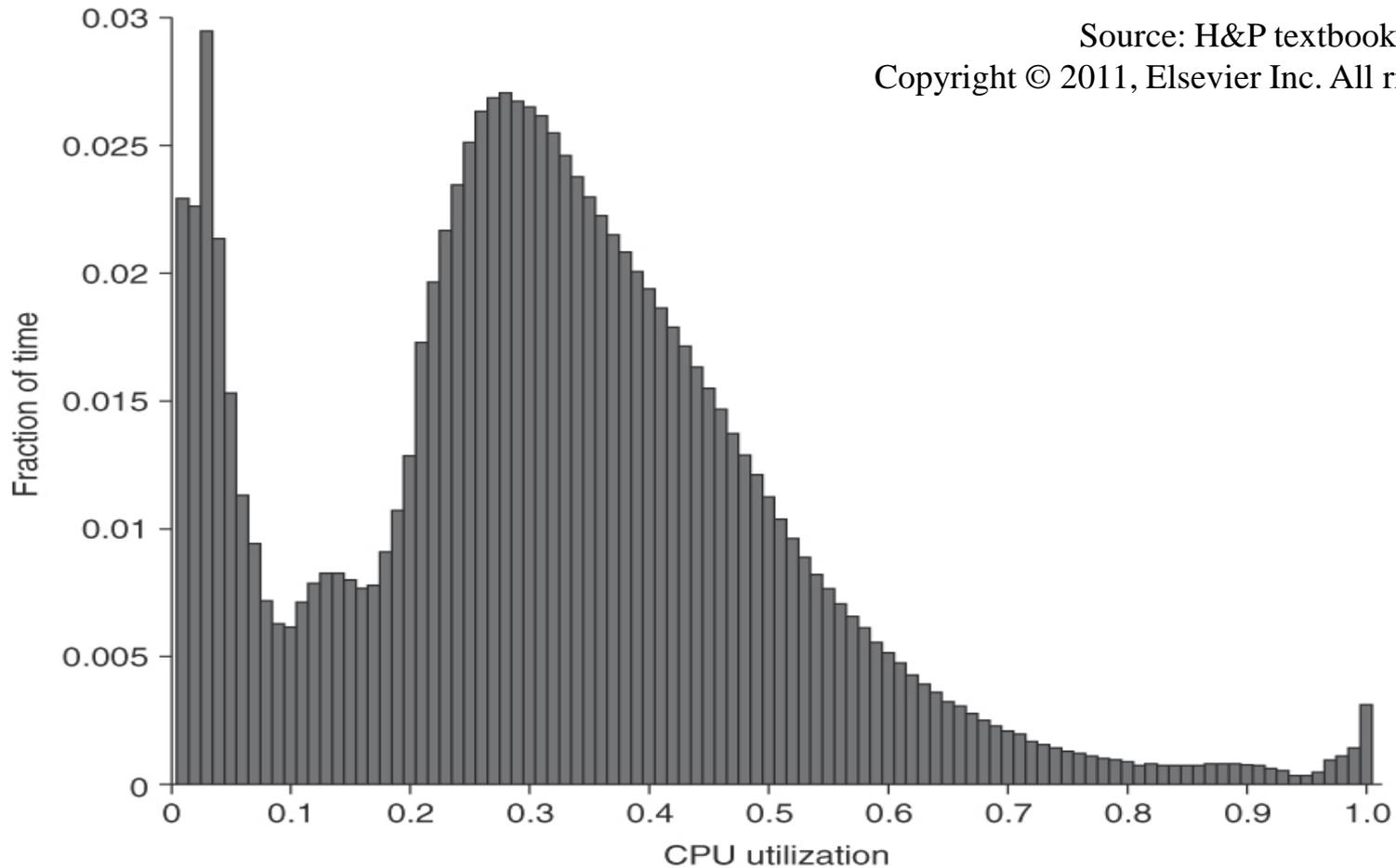
- An unloaded server dissipates a large amount of power
- Ideally, we want energy-proportional computing, but in reality, servers are not energy-proportional
- Can approach energy-proportionality by turning on a few servers that are heavily utilized
- See figures on next two slides for power/utilization profile of a server and a utilization profile of servers in a WSC

Power/Utilization Profile



Source: H&P textbook.
Copyright © 2011, Elsevier Inc. All rights Reserved.

Server Utilization Profile



Source: H&P textbook.
Copyright © 2011, Elsevier Inc. All rights Reserved.

Figure 6.3 Average CPU utilization of more than 5000 servers during a 6-month period at Google. Servers are rarely completely idle or fully utilized, in-stead operating most of the time at between 10% and 50% of their maximum utilization. (From Figure 1 in Barroso and Hölzle [2007].) The column the third from the right in Figure 6.4 calculates percentages plus or minus 5% to come up with the weightings; thus, 1.2% for the 90% row means that 1.2% of servers were between 85% and 95% utilized.

Problem 1

Assume that a server consumes 100W at peak utilization and 50W at zero utilization. Assume a linear relationship between utilization and power. The server is capable of executing many threads in parallel. Assume that a single thread utilizes 25% of all server resources (functional units, caches, memory capacity, memory bandwidth, etc.). What is the total power dissipation when executing 99 threads on a collection of these servers, such that performance and energy are close to optimal?

Problem 1

Assume that a server consumes 100W at peak utilization and 50W at zero utilization. Assume a linear relationship between utilization and power. The server is capable of executing many threads in parallel. Assume that a single thread utilizes 25% of all server resources (functional units, caches, memory capacity, memory bandwidth, etc.). What is the total power dissipation when executing 99 threads on a collection of these servers, such that performance and energy are close to optimal?

For near-optimal performance and energy, use 25 servers. 24 servers at 100% utilization, executing 96 threads, consuming 2400W. The 25th server will run the last 3 threads and consume 87.5~W.

Other Metrics

- Performance does matter, both latency and throughput
- An analysis of the Bing search engine shows that if a 200ms delay is introduced in the response, the next click by the user is delayed by 500ms; so a poor response time amplifies the user's non-productivity
- Reliability (MTTF) and Availability ($MTTF / (MTTF + MTTR)$) are very important, given the large scale
- A server with MTTF of 25 years (amazing!) : 50K servers would lead to 5 server failures a day; Similarly, annual disk failure rate is 2-10% → 1 disk failure every hour

Important Problems

- Reducing power in power-down states
- Maximizing utilization
- Reducing cost with virtualization
- Reducing data movement
- Building a low-power low-cost processor
- Building a low-power low-cost hi-bw memory
- Low-power low-cost on-demand reliability

Magnetic Disks

- A magnetic disk consists of 1-12 *platters* (metal or glass disk covered with magnetic recording material on both sides), with diameters between 1-3.5 inches
- Each platter is comprised of concentric *tracks* (5-30K) and each track is divided into *sectors* (100 – 500 per track, each about 512 bytes)
- A movable arm holds the read/write heads for each disk surface and moves them all in tandem – a *cylinder* of data is accessible at a time

Disk Latency

- To read/write data, the arm has to be placed on the correct track – this *seek time* usually takes 5 to 12 ms on average – can take less if there is spatial locality
- *Rotational latency* is the time taken to rotate the correct sector under the head – average is typically more than 2 ms (15,000 RPM)
- *Transfer time* is the time taken to transfer a block of bits out of the disk and is typically 3 – 65 MB/second
- A disk controller maintains a disk cache (spatial locality can be exploited) and sets up the transfer on the bus (*controller overhead*)

RAID

- Reliability and availability are important metrics for disks
- RAID: redundant array of inexpensive (independent) disks
- Redundancy can deal with one or more failures
- Each sector of a disk records check information that allows it to determine if the disk has an error or not (in other words, redundancy already exists within a disk)
- When the disk read flags an error, we turn elsewhere for correct data

RAID 0 and RAID 1

- RAID 0 has no additional redundancy (misnomer) – it uses an array of disks and stripes (interleaves) data across the arrays to improve parallelism and throughput
- RAID 1 mirrors or shadows every disk – every write happens to two disks
- Reads to the mirror may happen only when the primary disk fails – or, you may try to read both together and the quicker response is accepted
- Expensive solution: high reliability at twice the cost

RAID 3

- Data is bit-interleaved across several disks and a separate disk maintains parity information for a set of bits
- For example: with 8 disks, bit 0 is in disk-0, bit 1 is in disk-1, ..., bit 7 is in disk-7; disk-8 maintains parity for all 8 bits
- For any read, 8 disks must be accessed (as we usually read more than a byte at a time) and for any write, 9 disks must be accessed as parity has to be re-calculated
- High throughput for a single request, low cost for redundancy (overhead: 12.5%), low task-level parallelism

RAID 4 and RAID 5

- Data is block interleaved – this allows us to get all our data from a single disk on a read – in case of a disk error, read all 9 disks
- Block interleaving reduces thrupt for a single request (as only a single disk drive is servicing the request), but improves task-level parallelism as other disk drives are free to service other requests
- On a write, we access the disk that stores the data and the parity disk – parity information can be updated simply by checking if the new data differs from the old data

RAID 5

- If we have a single disk for parity, multiple writes can not happen in parallel (as all writes must update parity info)
- RAID 5 distributes the parity block to allow simultaneous writes

Other Reliability Approaches

- High reliability is also expected of memory systems; many memory systems offer SEC-DED support – single error correct, double error detect; implemented with an 8-bit code for every 64-bit data word on ECC DIMMs
- Some memory systems offer chipkill support – the ability to recover from complete failure in one memory chip – many implementations exist, some resembling RAID designs
- Caches are typically protected with SEC-DED codes
- Some cores implement various forms of redundancy, e.g., DMR or TMR – dual or triple modular redundancy

SIMD Processors

- Single instruction, multiple data
- Such processors offer energy efficiency because a single instruction fetch can trigger many data operations
- Such data parallelism may be useful for many image/sound and numerical applications

GPUs

- Initially developed as graphics accelerators; now viewed as one of the densest compute engines available
- Many on-going efforts to run non-graphics workloads on GPUs, i.e., use them as general-purpose GPUs or GPGPUs
- C/C++ based programming platforms enable wider use of GPGPUs – CUDA from NVidia and OpenCL from an industry consortium
- A heterogeneous system has a regular host CPU and a GPU that handles (say) CUDA code (they can both be on the same chip)

The GPU Architecture

- SIMT – single instruction, multiple thread; a GPU has many SIMT cores
- A large data-parallel operation is partitioned into many thread blocks (one per SIMT core); a thread block is partitioned into many warps (one warp running at a time in the SIMT core); a warp is partitioned across many in-order pipelines (each is called a SIMD lane)
- A SIMT core can have multiple active warps at a time, i.e., the SIMT core stores the registers for each warp; warps can be context-switched at low cost; a warp scheduler keeps track of runnable warps and schedules a new warp if the currently running warp stalls

Architecture Features

- Simple in-order pipelines that rely on thread-level parallelism to hide long latencies
- Many registers (~1K) per in-order pipeline (lane) to support many active warps
- When a branch is encountered, some of the lanes proceed along the “then” case depending on their data values; later, the other lanes evaluate the “else” case; a branch cuts the data-level parallelism by half (branch divergence)
- When a load/store is encountered, the requests from all lanes are coalesced into a few 128B cache line requests; each request may return at a different time (mem divergence)

GPU Memory Hierarchy

- Each SIMT core has a private L1 cache (shared by the warps on that core)
- A large L2 is shared by all SIMT cores; each L2 bank services a subset of all addresses
- Each L2 partition is connected to its own memory controller and memory channel
- The GDDR5 memory system runs at higher frequencies, and uses chips with more banks, wide IO, and better power delivery networks
- A portion of GDDR5 memory is private to the GPU and the rest is accessible to the host CPU (the GPU performs copies)

Title

- Bullet