Lecture 28
Parallel Processors II

School of Computer Science and Engineering
Soongsil University

6. Parallel Processors from Client to Cloud

6.6 Introduction to Graphics Processing Units

- **Early video cards**
  - Frame buffer memory with address generation for video output
- **3D graphics processing**
  - Originally high-end computers (e.g., SGI)
  - Moore's Law $\Rightarrow$ lower cost, higher density
  - 3D graphics cards for PCs and game consoles
- **Graphics Processing Units**
  - Processors oriented to 3D graphics tasks
  - Vertex/pixel processing, shading, texture mapping, rasterization

CPU versus GPU

- **CPU**
  - Must be good at everything, parallel or not
  - Big on-chip caches
  - Sophisticated control logic
- **GPU**
  - Do not need be able to perform all the tasks of a CPU
    - Can dedicate all their resources to graphics
  - Small problems ... typically hundreds of megabytes to gigabytes
  - Do not rely on multilevel caches => instead hardware multithreading
  - GPU memory is oriented toward bandwidth rather than latency
  - Many parallel processors (MIMD) + many threads
    - Hence, each GPU processor is more highly multithreaded than a typical CPU, plus they have more processors.

NVIDIA Tesla Architecture of GeForce 8800

- Texture/processor cluster (TPC)
  - A geometry controller, an SMC, two SMs, and a texture unit
  - 7 TPCs
- Streaming multiprocessor (SM)
  - Eight SP thread processor cores, two special function units (SFUs), a multithreaded instruction fetch and issue unit (MT issue), an instruction cache, a read-only constant cache, and a 16 KB read/write shared memory
  - 2 SMs/TPC => 14 SMs
- Streaming processor (SP) core
  - Each SP is fine-grained multithreaded
  - Single-precision FP and integer units
  - 1024 scalar 32-bit registers
  - 8 SPs/SM => 112 SP cores

Basic Tesla Architecture of an NVIDIA GeForce 8800

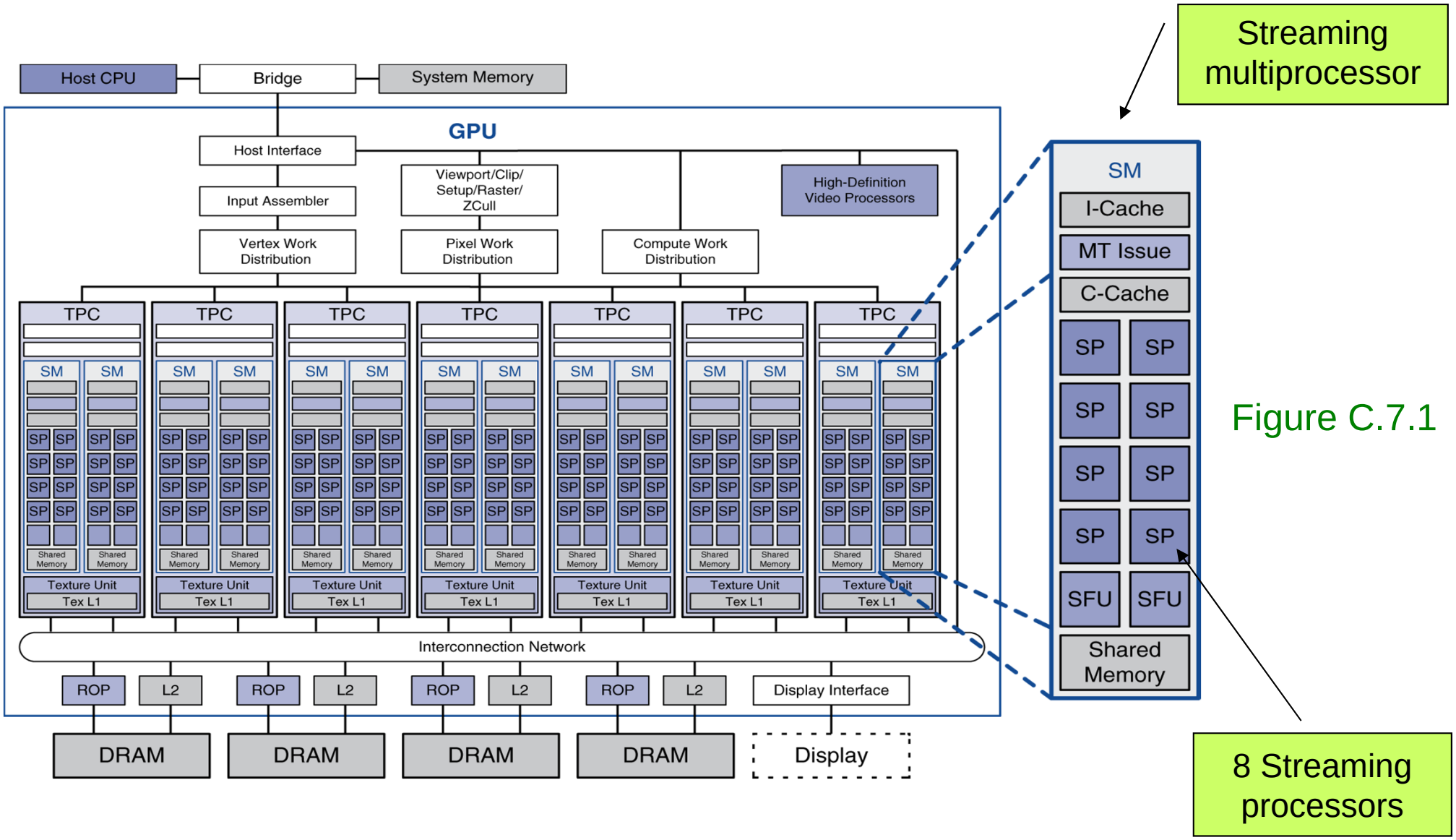


Figure C.7.1

6.7 Clusters, Warehouse Scale Computers and Other Message-Passing Multiprocessors

- **Message-Passing Multiprocessors**
  - Private memory multiprocessors
  - Each processor has private physical address space
  - Hardware sends/receives messages between processors

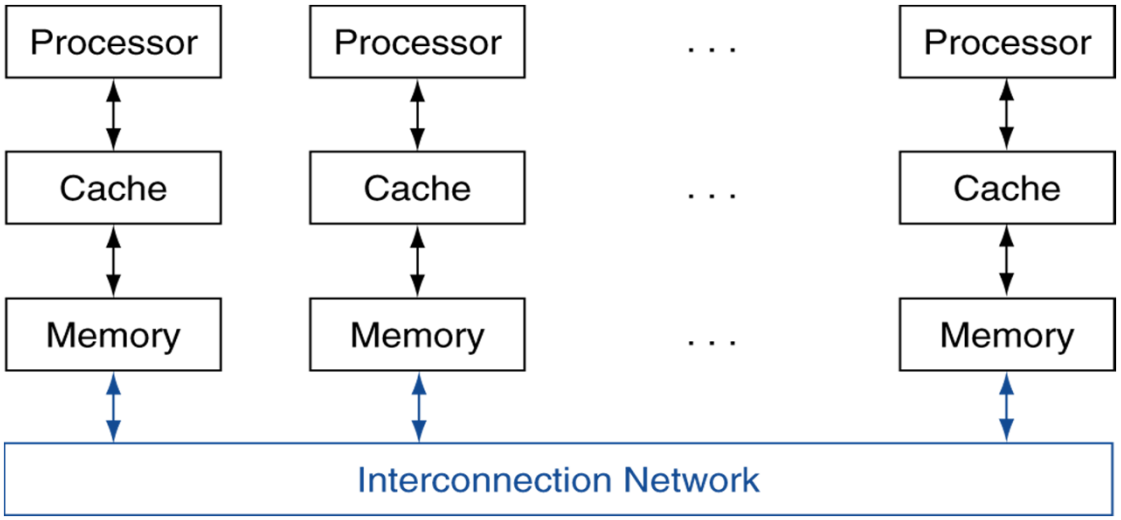


Figure 6.13

Loosely Coupled Clusters

- **Network of independent computers**
  - Each has private memory and OS
  - Connected using I/O system
    - e.g., Ethernet/switch, Internet
  - Lower cost, higher availability, and rapid, incremental expandability
- **Suitable for applications with independent tasks**
  - Web servers, databases, simulations, ...
- **Problems**
  - Administration cost
    - Virtual machines make clusters easier to administer
  - Low interconnect bandwidth

    (cf) processor/memory bandwidth on an SMP
  - Overhead in the division of memory
    - n independent memories and n copies of the operating system

Warehouse-Scale Computers

- **Warehouse-scale computer (WSC)**
  - As computation moves into the cloud, the computing platform of interest is no longer a pizza box or a refrigerator, but a warehouse full of computers.
  - Hardware and software resources in these facilities must work in concert to deliver Internet service performance.
  - Need to treat the datacenter itself as one massive warehouse-scale computer (WSC).
- **Luiz André Barroso and Urs Hölzle, The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Morgan & Claypool Publishers**

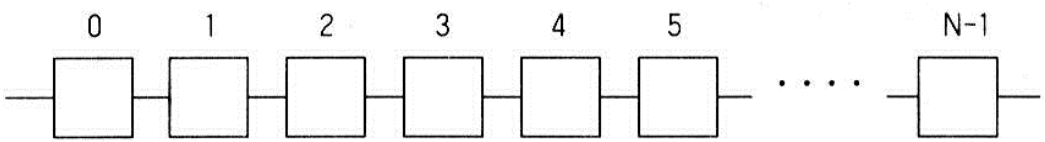
Three Major Distinctions from Servers

1. **Ample, easy parallelism**
   - Batch applications like MapReduce
     - Large number of independent data sets that need independent processing
   - Soft ware as a Service (SaaS)
     - Millions of independent users of interactive Internet services
2. **Operational Costs Count**
   - WSC have longer lifetimes—10 or more years—so the operational costs add up
   - Energy, power distribution, and cooling represent more than 30% of the costs of a WSC over 10 years
3. **Scale and the Opportunities/Problems Associated with Scale**
   - Volume discounts
   - Large AFR … fault tolerance is even more important

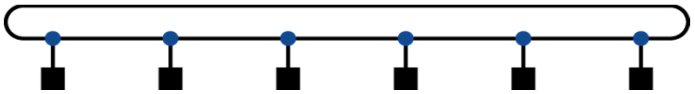
6.8 Introduction to Multiprocessor Network Topologies

- **Network topologies**
  - Arrangements of processors, switches, and links
  - Diameter, bisection width, I/O bandwidth, node degree, symmetry, recursive structure
- **Network cost**
  - Number of switches, degree of a switch, width per link, length of the links
- **Network performance**
  - Bandwidth, average and variance of latency, degree of fault tolerance, power efficiency, functionality, scalability
- **Static networks (= direct networks)**
  - Linear array, ring, chordal ring, tree, mesh, torus, hypercube, shuffle-exchange, barrel-shifter
- **Dynamic networks (= indirect networks)**
  - Bus, multistage interconnection network(MIN), crossbar switch matrix

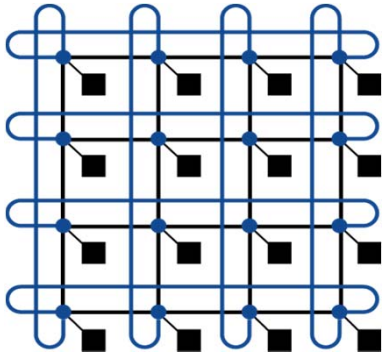
Static Networks (1/2)



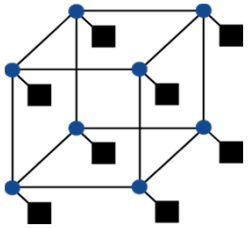
Linear array



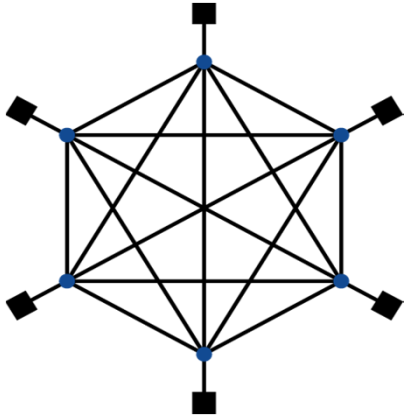
Ring



2D Mesh



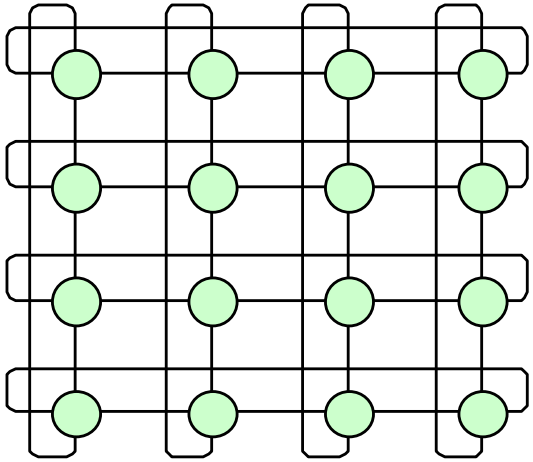
N-cube (N = 3)



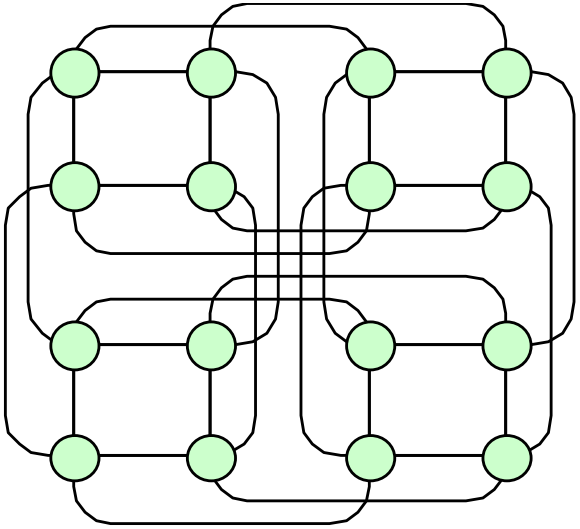
Fully connected

Figure 6.14

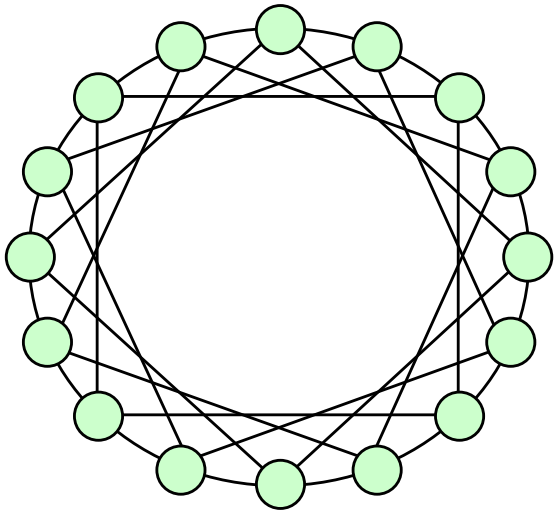
Static Networks (2/2)



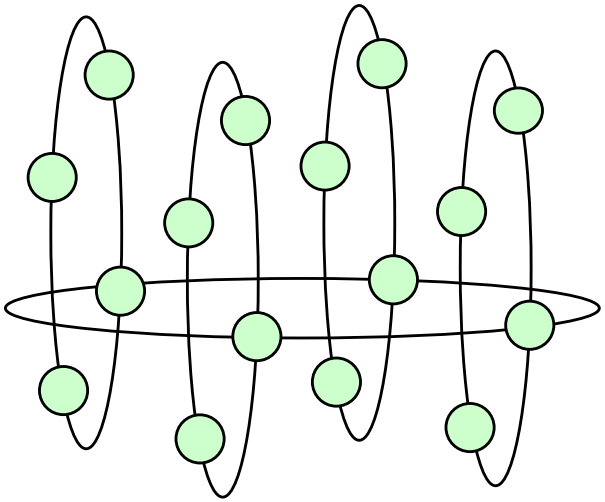
(a) 2D torus



(b) 4D hypercube

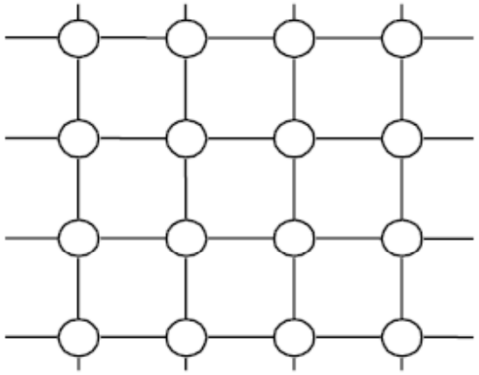


(c) Chordal ring

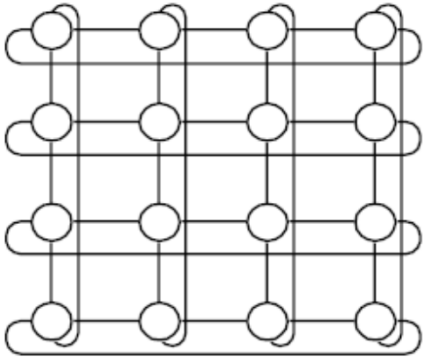


(d) Ring of rings

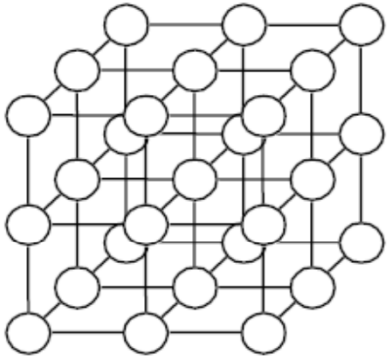
Meshes and Hypercubes



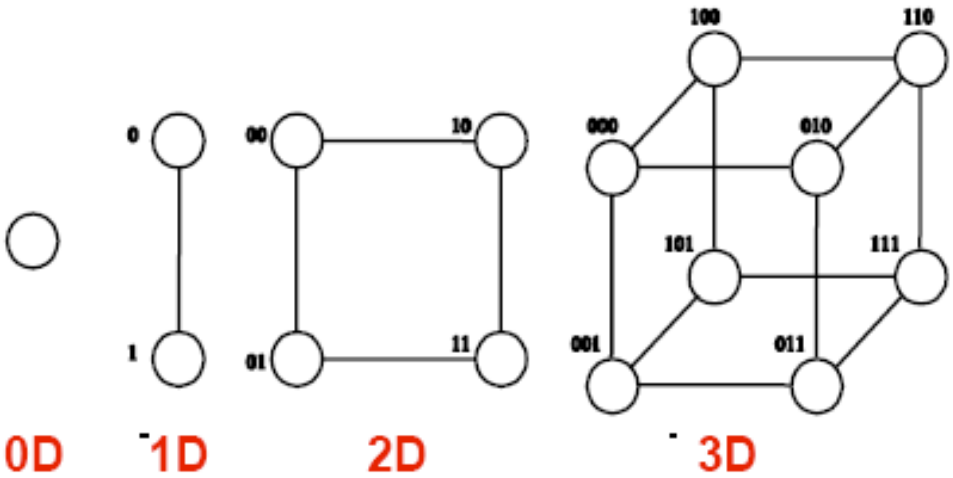
2D mesh

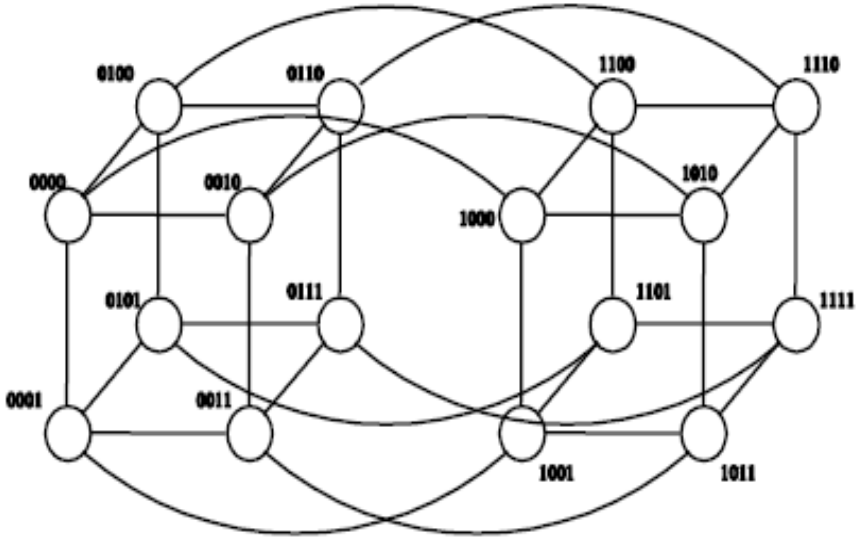


2D torus

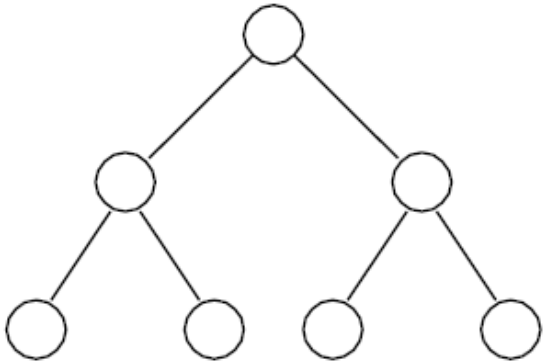


3D mesh

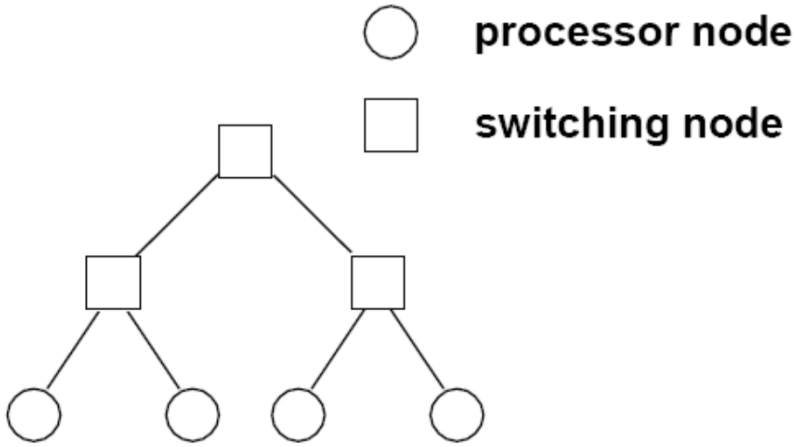




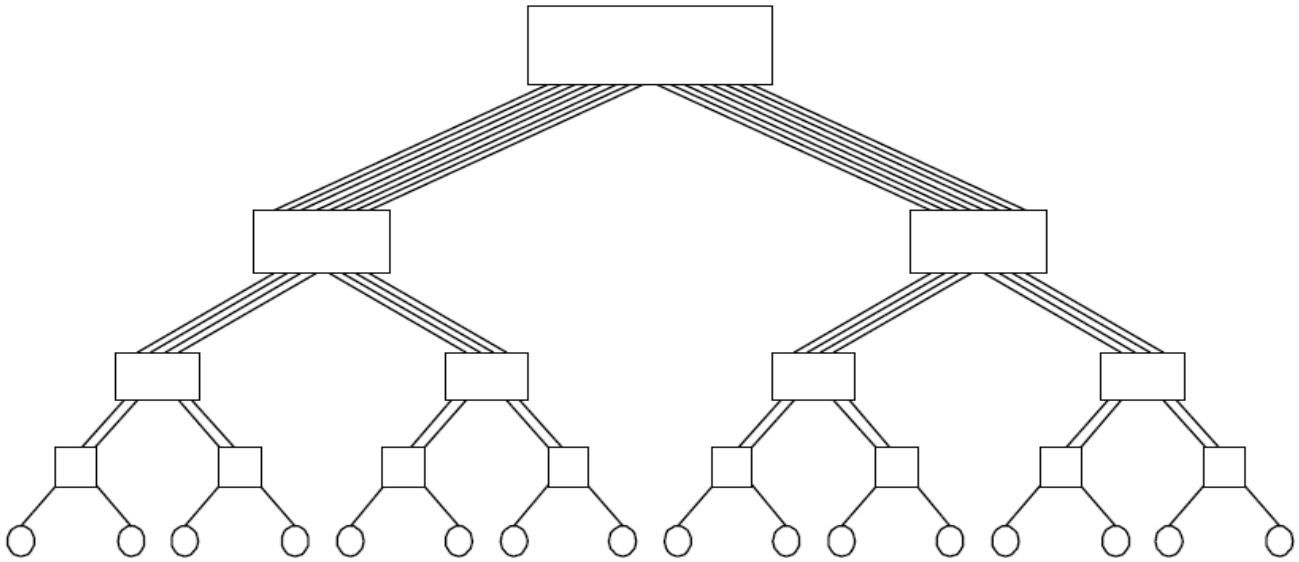
Trees and Fat Trees



static tree network



dynamic tree network

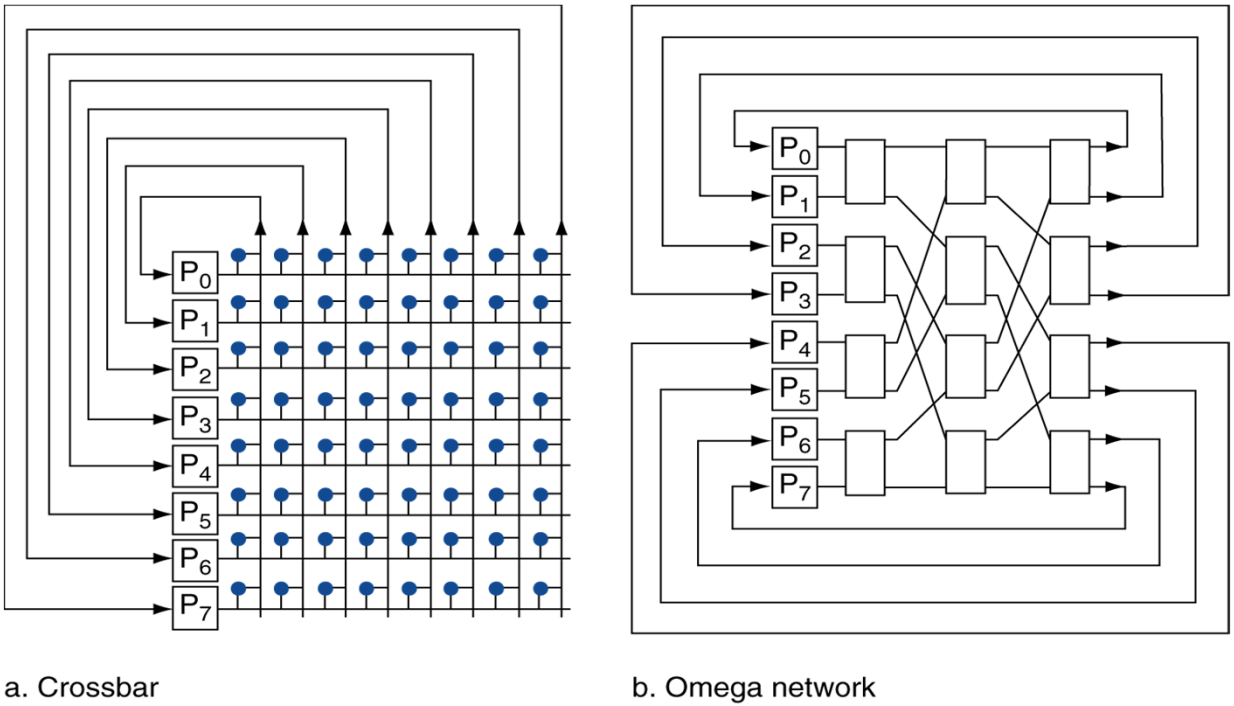


fat tree network

Dynamic Networks

- **Bus**
  - Excellent cost scalability … O(1)
  - Poor performance scalability
- **Crossbars switch matrix**
  - Excellent performance scalability
  - Poor cost scalability … $O(n^2)$
- **Multistage interconnection network (MIN)**
  - Compromise between these extremes
  - Cost … $O(n \cdot \log n)$
  - Blocking networks
  - Nonblocking networks

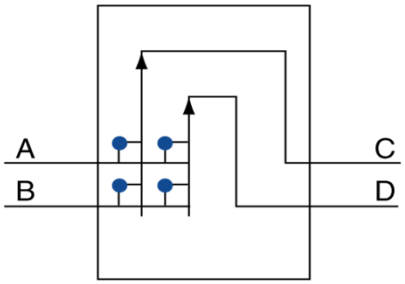
Crossbar and MIN



a. Crossbar

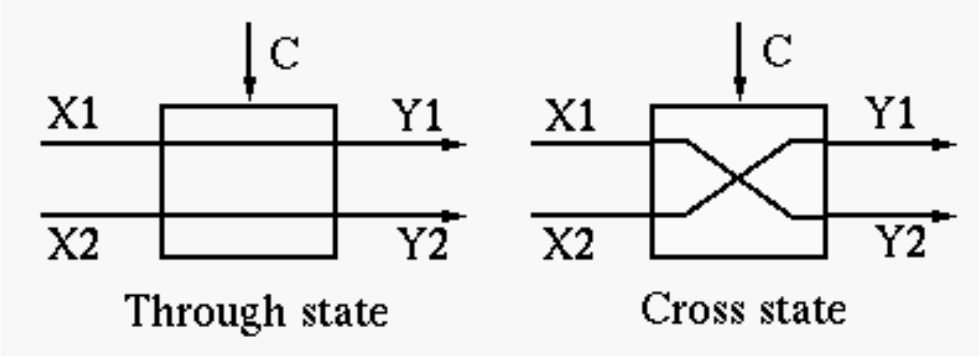
b. Omega network

Figure 6.15

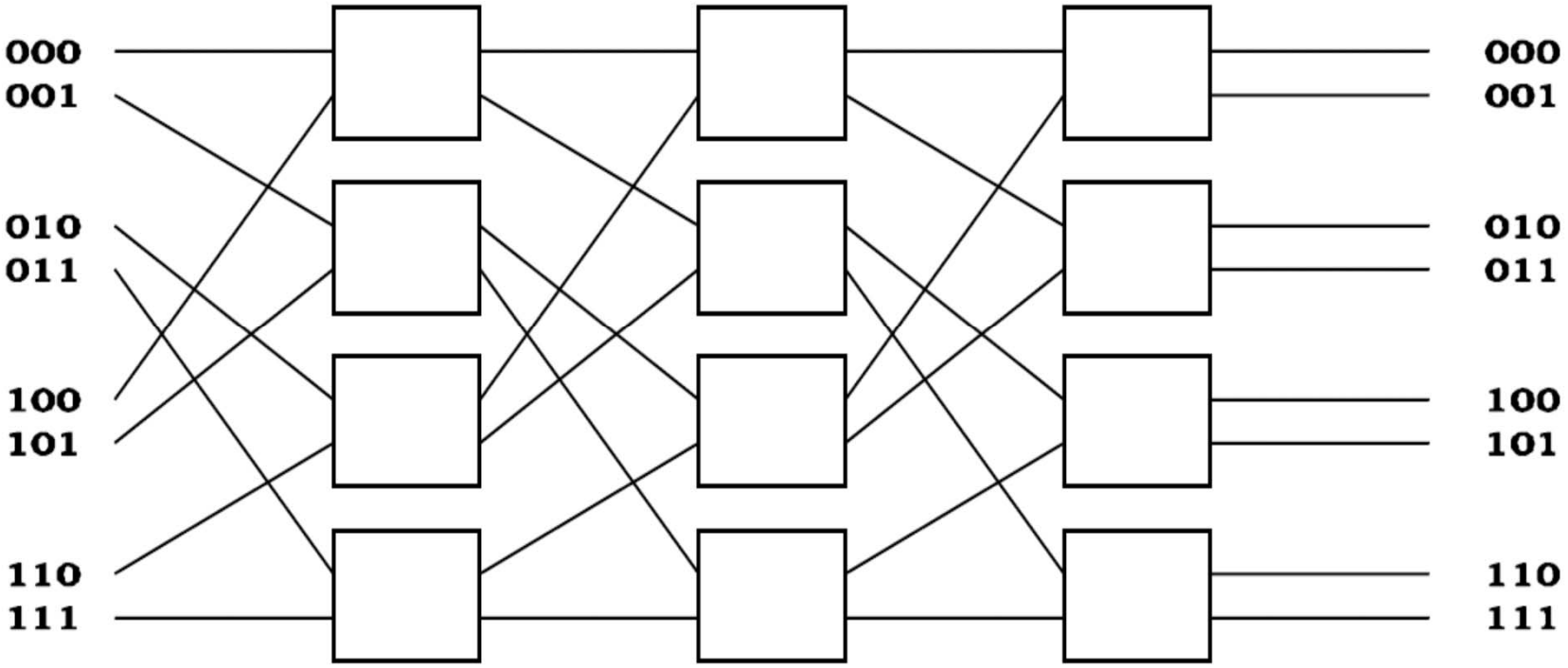


c. Omega network switch box

2x2 switching element (SE)

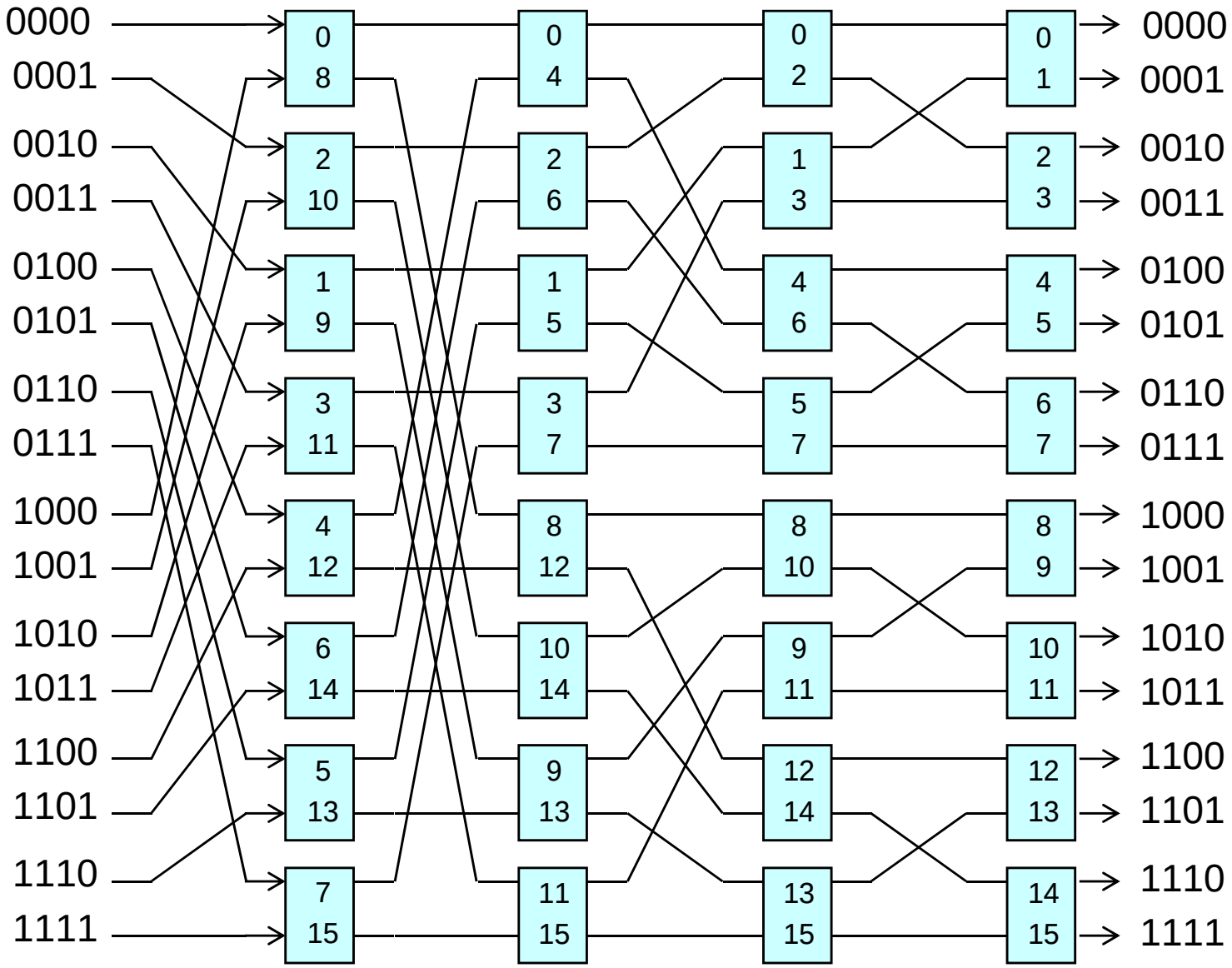


8x8 Omega Network

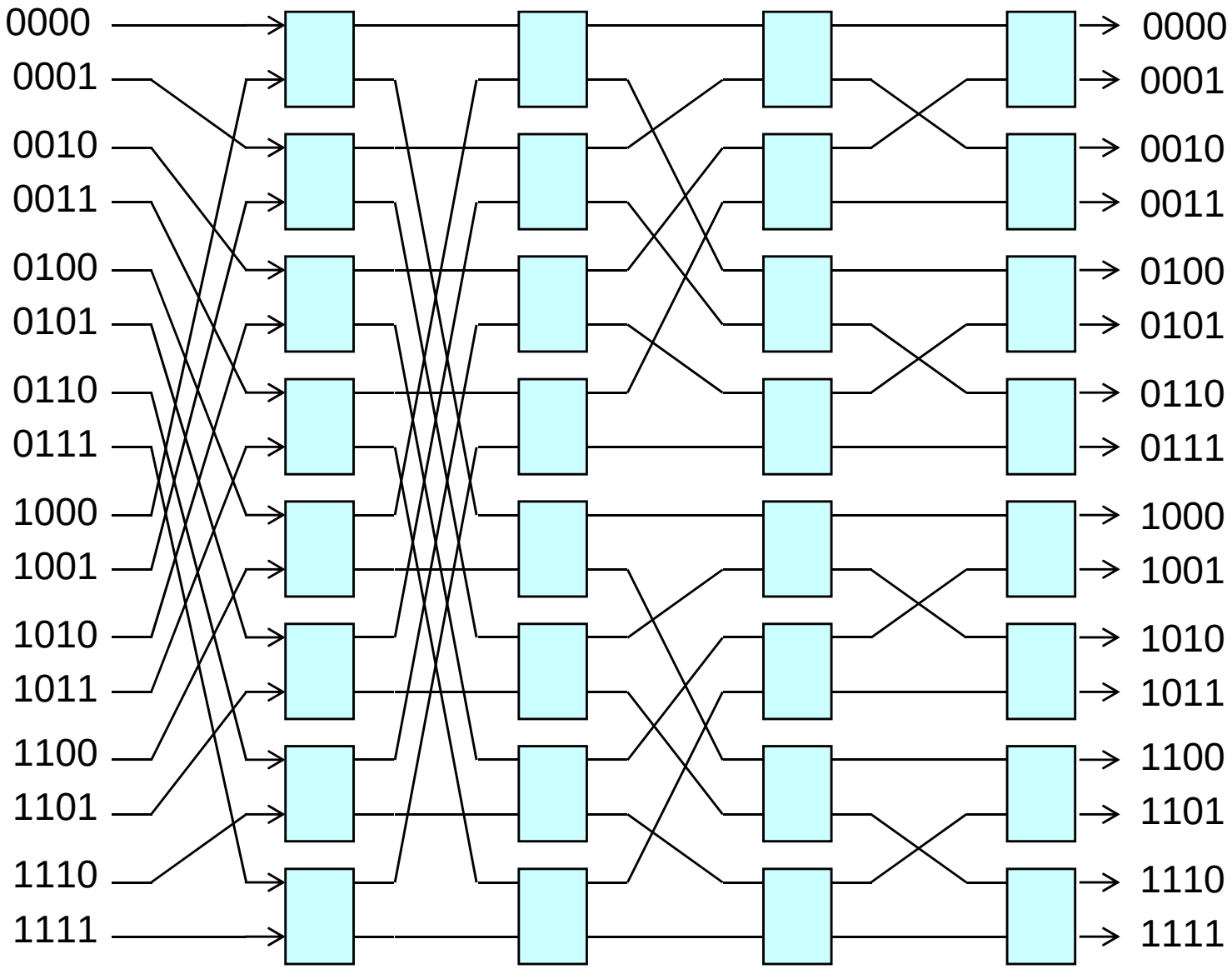


Cost: n/2 × log n switching elements → O(n log n)

Multistage Cube Network



Routing in Multistage Cube Network



6.10 Multiprocessor Benchmarks and Performance Models

- **Linpack**
  - Matrix linear algebra
- **SPECrate**
  - Parallel run of SPEC CPU programs
  - Job-level parallelism
- **SPLASH (Stanford Parallel Applications for Shared Memory) and SPLASH2**
  - Mix of kernels and applications, strong scaling
- **NAS (NASA Advanced Supercomputing) suite**
  - Computational fluid dynamics kernels
- **PARSEC (Princeton Application Repository for Shared Memory Computers) suite**
  - Multithreaded applications using Pthreads and OpenMP

Performance Models

- **Performance model**
  - A model that offered insights into the performance of different architectures
  - Need not be perfect, just insightful
  - e.g. 3Cs for cache performance
- **Berkeley Design Patterns [Asanovic et al., 2006]**
  - 13 design patterns that the researchers at the University of California at Berkeley claim will be part of applications of the future
- **Arithmetic intensity of a kernel**
  - FLOPs per byte of memory accessed

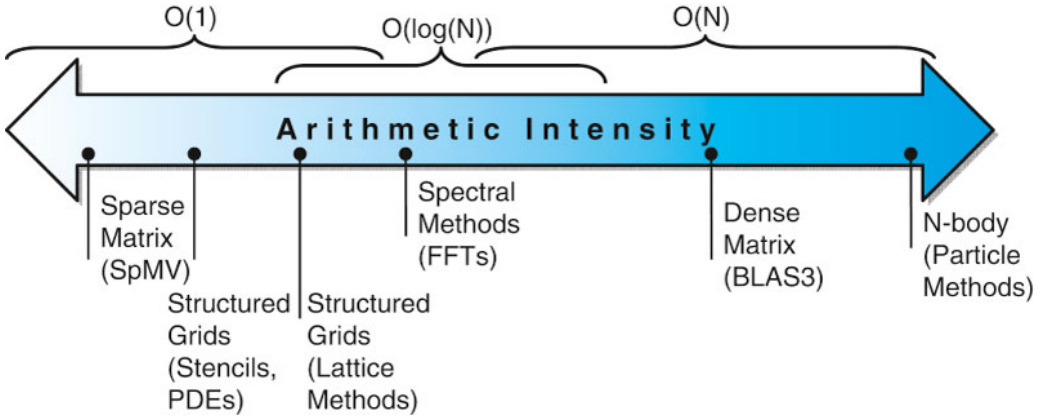


Figure 6.17

The Roofline Model

- **Ties together floating-point performance and memory performance for a target machine**
  - Peak GFLOPS (from hardware specification)
  - Peak memory bytes/sec (using Stream benchmark)
- **Attainable GPLOPs/sec**

  = Max (Peak Memory BW × Arithmetic Intensity,

  Peak FP Performance)

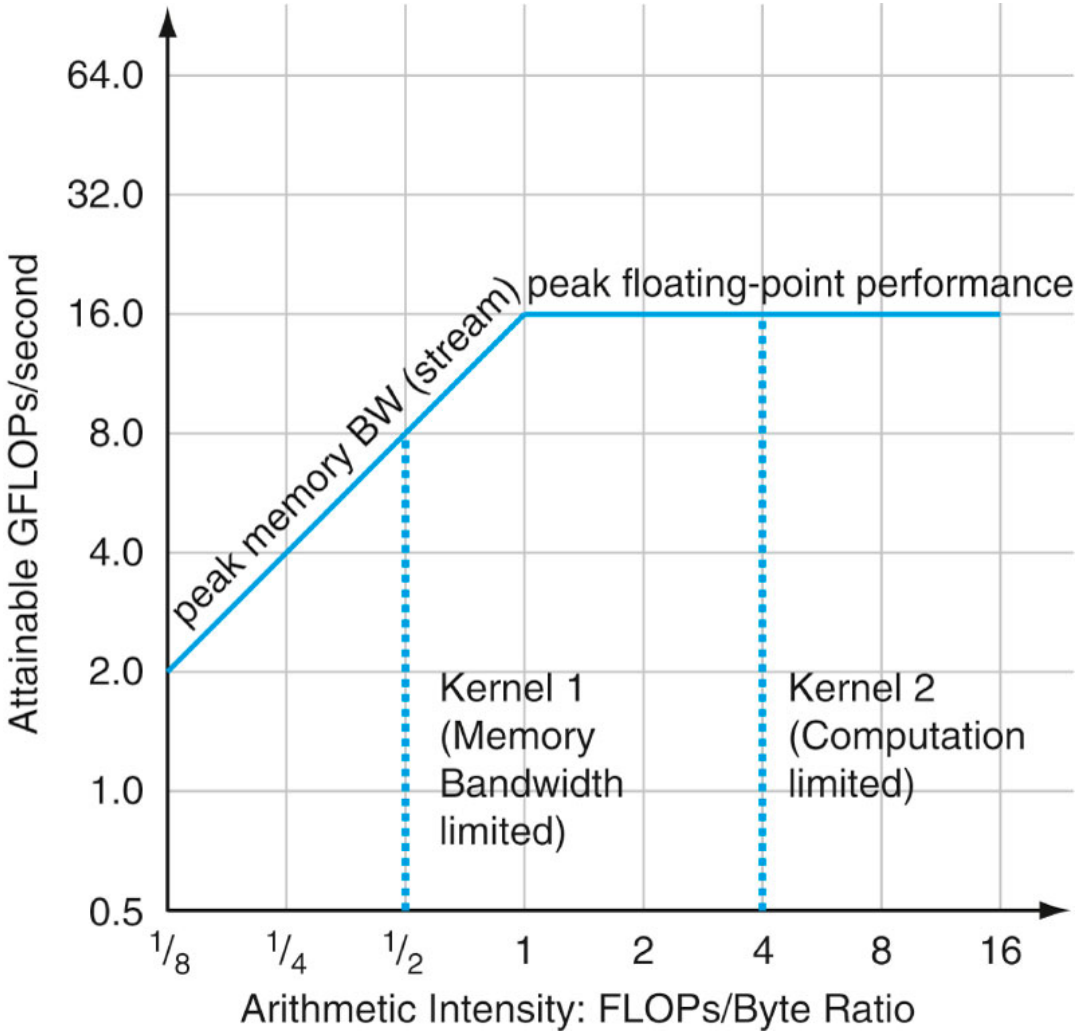


Figure 6.18

Comparing Two Generations of Opterons

- **Example: Opteron X2 vs. Opteron X4**
  - 2-core vs. 4-core, 2× FP performance/core, 2.2GHz vs. 2.3GHz
  - Same memory system

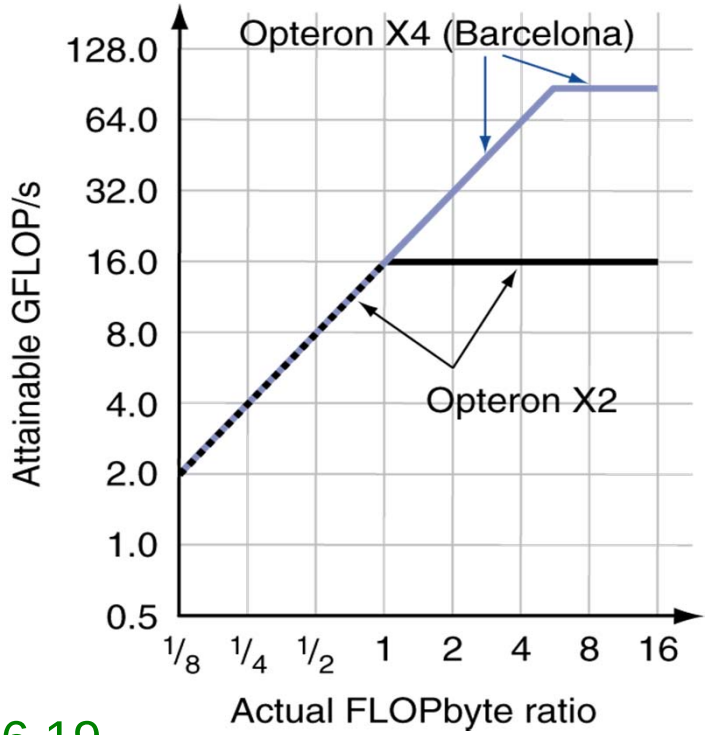


Figure 6.19

- To get higher performance on X4 than X2
  - Need high arithmetic intensity
  - Or working set must fit in X4's 2MiB L-3 cache

Optimizing Performance

- **To reduce computational bottlenecks**
  1. Floating-point operation mix
     - An equal number of nearly simultaneous additions and multiplications
  2. Improve instruction-level parallelism and apply SIMD
     - Superscalar and SIMD instructions
- **To reduce memory bottlenecks**
  3. Software prefetching
     - Predicting accesses via software prefetch instructions to avoid load stalls
  4. Memory affinity
     - Allocating data and the threads using that data to the same memory-processor pair to avoid off-chip memory access

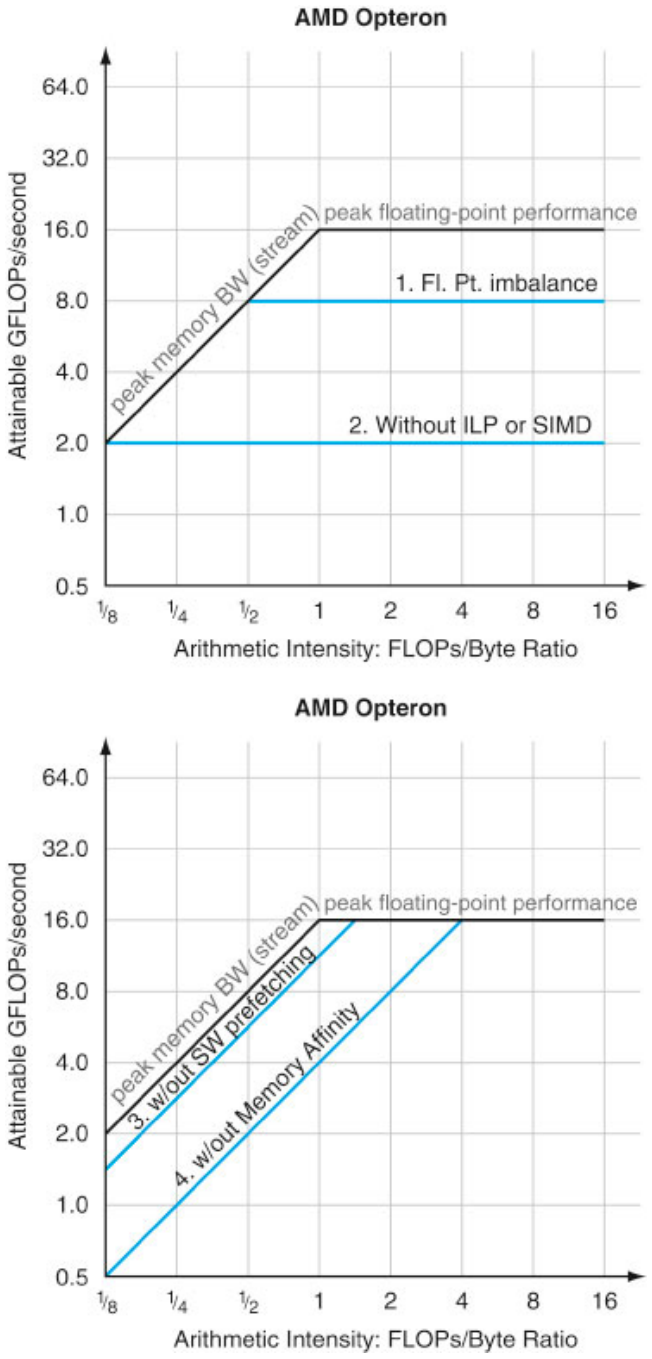


Figure 6.20

Example: Optimization with 2 Kernels

- **Kernel 1**
  - In the blue-gray parallelogram in the middle
  - Need optimizations 1, 3 and 4
- **Kernel 2**
  - In the blue trapezoid on the right
  - Need optimizations 1 and 2
- **Arithmetic intensity is not always fixed**
  - May scale with problem size
  - Caching reduces memory accesses
    - Increases arithmetic intensity

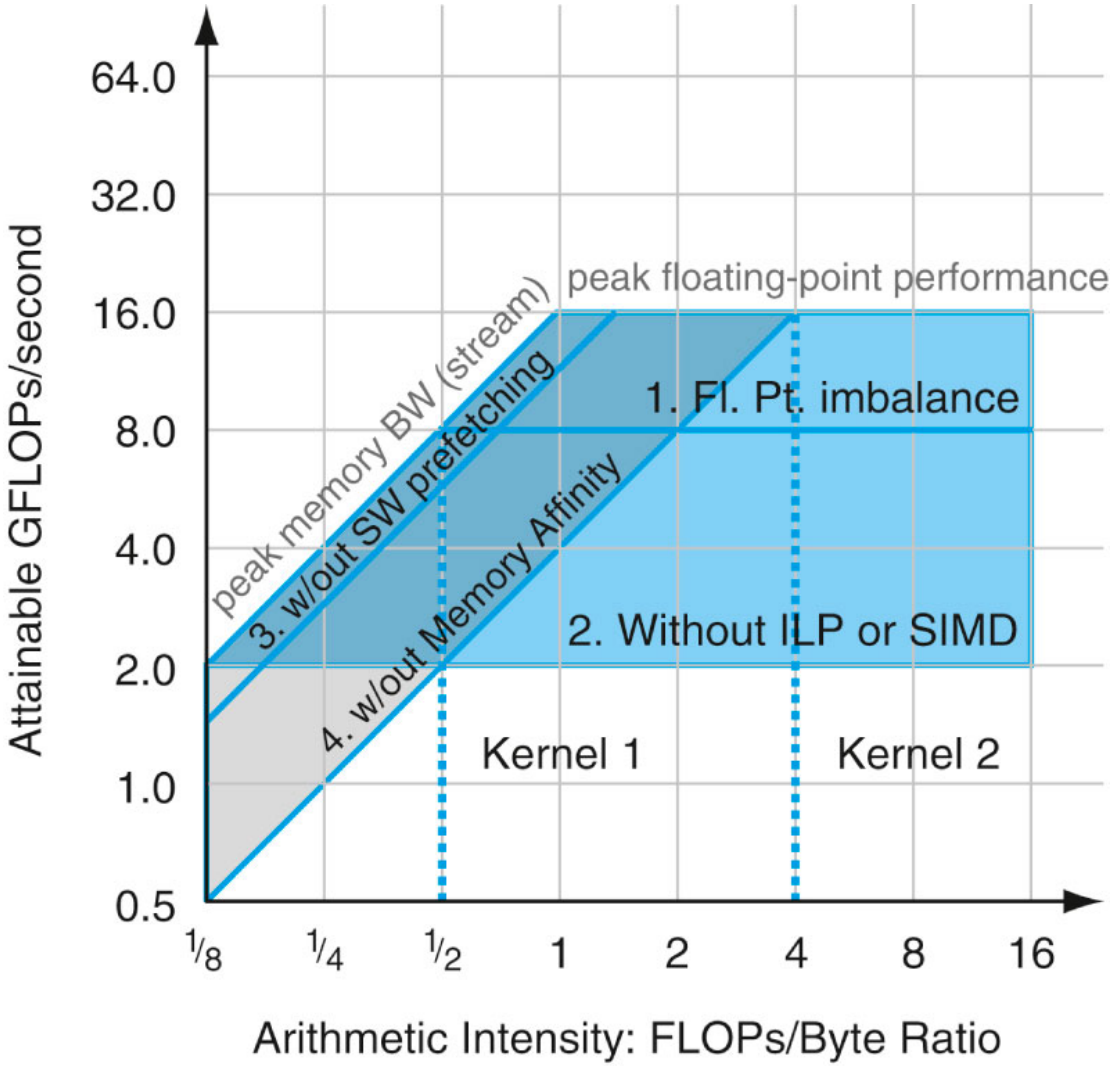


Figure 6.21