15-740/18-740 Computer Architecture
Lecture 2: SIMD, MIMD, and ISA Principles

Prof. Onur Mutlu
Carnegie Mellon University
Fall 2011, 9/14/2011

Readings Referenced Today

- ISA and Compilers (Review Set 2)
  - Colwell et al., "Instruction Sets and Beyond: Computers, Complexity, and Controversy," IEEE Computer 1985.
  - Wulf, "Compilers and Computer Architecture," IEEE Computer 1981.

- Alternative ISA and execution models
  - Fisher, "Very Long Instruction Word Architectures and the ELI-512," ISCA 1983.
  - Russell, "The CRAY-1 computer system," CACM 1978.
  - Lindholm et al., "NVIDIA Tesla: A Unified Graphics and Computing Architecture," IEEE Micro 2008.

Readings Referenced Today (Perhaps)

- On-chip networks
  - Dally and Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks," DAC 2001.
  - Wentzlaff et al., "On-Chip Interconnection Architecture of the Tile Processor," IEEE Micro 2007.

- Main memory controllers
  - Moscibroda and Mutlu, "Memory performance attacks: Denial of memory service in multi-core systems," USENIX Security 2007.
  - Rixner et al., "Memory Access Scheduling," ISCA 2000.

- Architecture reference manuals
  - Digital Equipment Corp., "VAX11 780 Architecture Handbook," 1977-78.
  - Intel Corp. "Intel 64 and IA-32 Architectures Software Developer's Manual"

Review Set 2

- Colwell et al., "Instruction Sets and Beyond: Computers, Complexity, and Controversy," IEEE Computer 1985.
- Wulf, "Compilers and Computer Architecture," IEEE Computer 1981.

- Due next Wednesday

CALCM Seminar Today

- "Efficient, Heterogeneous, Parallel Processing: The Design of a Micropolygon Rendering Pipeline"
- Kayvon Fatahalian, CMU
- 4-5 pm, September 14, Wednesday
- Singleton Room, Roberts Engineering Hall
- http://www.ece.cmu.edu/~calcm/doku.php?id=seminars:seminar_11_09_14

- Attend and provide a review online
  - Review Set 3

Review: Data Parallelism

- Concurrency arises from performing the same operations on different pieces of data
  - Single instruction multiple data (SIMD)
  - E.g., dot product of two vectors
- Contrast with thread ("control") parallelism
  - Concurrency arises from executing different threads of control in parallel
- Contrast with data flow
  - Concurrency arises from executing different operations in parallel (in a data driven manner)
- SIMD exploits instruction-level parallelism
  - Multiple instructions concurrent: instructions happen to be the same

Review: SIMD Processing

- Single instruction operates on multiple data elements
  - In time or in space
- Multiple processing elements

- Time-space duality
  - Array processor: Instruction operates on multiple data elements at the same time
  - Vector processor: Instruction operates on multiple data elements in consecutive time steps

Review: Vector Processors

- A vector is a one-dimensional array of numbers
- Many scientific/commercial programs use vectors

```
for (i = 0; i<=49; i++)
    C[i] = (A[i] + B[i]) / 2
```

- A vector processor is one whose instructions operate on vectors rather than scalar (single data) values
- Basic requirements
  - Need to load/store vectors → vector registers (contain vectors)
  - Need to operate on vectors of different lengths → vector length register (VLEN)
  - Elements of a vector might be stored apart from each other in memory → vector stride register (VSTR)
    - Stride: distance between two elements of a vector

Review: Vector Processors (II)

- A vector instruction performs an operation on each element in consecutive cycles
  - Vector functional units are pipelined
  - Each pipeline stage operates on a different data element

- Vector instructions allow deeper pipelines
  - No intra-vector dependencies → no hardware interlocking within a vector
  - No control flow within a vector
  - Known stride allows prefetching of vectors into memory

Review: Vector Processor Advantages

+ No dependencies within a vector
  - Pipelining, parallelization work well
  - Can have very deep pipelines, no dependencies!

+ Each instruction generates a lot of work
  - Reduces instruction fetch bandwidth

+ Highly regular memory access pattern
  - Interleaving multiple banks for higher memory bandwidth
  - Prefetching

+ No need to explicitly code loops
  - Fewer branches in the instruction sequence

Review: Vector ISA Advantages

- Compact encoding
  - one short instruction encodes N operations

- Expressive, tells hardware that these N operations:
  - are independent
  - use the same functional unit
  - access disjoint registers
  - access registers in same pattern as previous instructions
  - access a contiguous block of memory (unit-stride load/store)
  - access memory in a known pattern (strided load/store)

- Scalable
  - can run the same code in parallel pipelines (*lanes)*

Review: Scalar Code Example

- For I = 1 to 50
  - C[i] = (A[i] + B[i]) / 2

- Scalar code

```
        MOVI R0 = 50                1
        MOVA R1 = A                 1
        MOVA R2 = B                 1
        MOVA R3 = C                 1
X:      LD R4 = MEM[R1++]           11  ;autoincrement addressing
        LD R5 = MEM[R2++]           11
        ADD R6 = R4 + R5            4
        SHFR R7 = R6 >> 1           1
        ST MEM[R3++] = R7           11
        DECBNZ --R0, X              2   ;decrement and branch if NZ
```

304 dynamic instructions

Review: Vector Code Example

- A loop is vectorizable if each iteration is independent of any other
- For I = 0 to 49
  - C[i] = (A[i] + B[i]) / 2

7 dynamic instructions

- Vectorized loop:

MOVI VLEN = 50	1
MOVI VSTR = 1	1
VLD V0 = A	11 + VLN - 1
VLD V1 = B	11 + VLN – 1
VADD V2 = V0 + V1	4 + VLN - 1
VSHFR V3 = V2 >> 1	1 + VLN - 1
VST C = V3	11 + VLN – 1

Scalar Code Execution Time

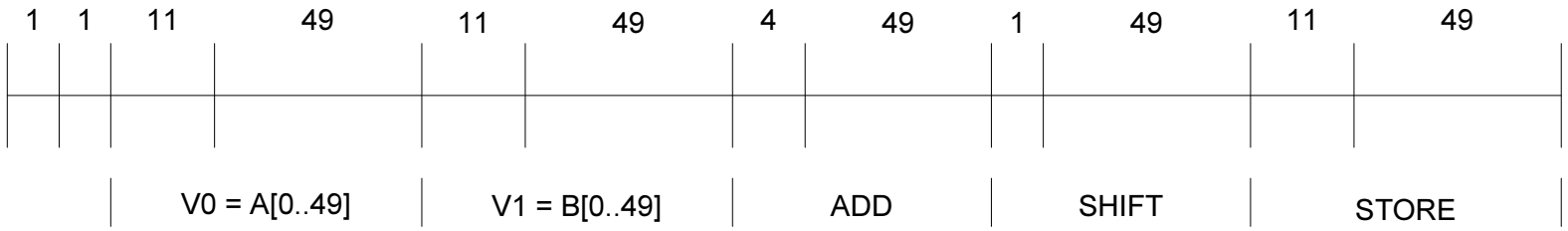
- Scalar execution time on an in-order processor with 1 bank
  - First two loads in the loop cannot be pipelined 2*11 cycles
  - 4 + 50*40 = 2004 cycles

- Scalar execution time on an in-order processor with 16 banks (word-interleaved)
  - First two loads in the loop can be pipelined
  - 4 + 50*30 = 1504 cycles

- Why 16 banks?
  - 11 cycle memory access latency
  - Having 16 (>11) banks ensures there are enough banks to overlap enough memory operations to cover memory latency

Vector Code Execution Time

- No chaining
  - i.e., output of a vector functional unit cannot be used as the input of another (i.e., no vector data forwarding)
- 16 memory banks (word-interleaved)



- 285 cycles

Vector Processor Disadvantages

-- Works (only) if parallelism is regular (data/SIMD parallelism)

++ Vector operations

-- Very inefficient if parallelism is irregular

-- How about searching for a key in a linked list?

> To program a vector machine, the compiler or hand coder must make the data structures in the code fit nearly exactly the regular structure built into the hardware. That's hard to do in first place, and just as hard to change. One tweak, and the low-level code has to be rewritten by a very smart and dedicated programmer who knows the hardware and often the subtleties of the application area. Often the rewriting is

Fisher, "Very Long Instruction Word Architectures and the ELI-512," ISCA 1983.

Vector Processor Limitations

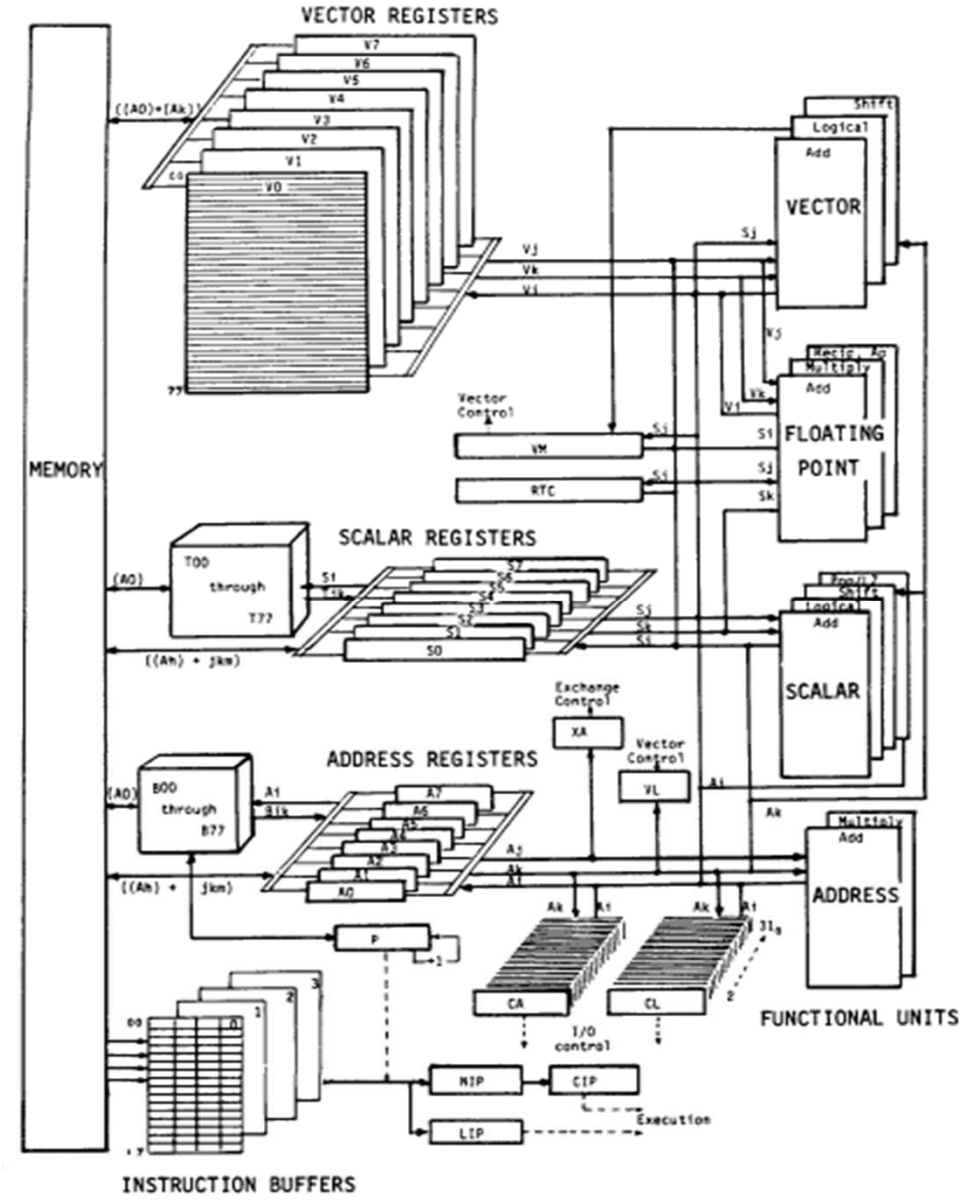
-- Memory (bandwidth) can easily become a bottleneck, especially if

1. compute/memory operation balance is not maintained
2. data is not mapped appropriately to memory banks

Further Reading: SIMD and GPUs

- Recommended
  - H&P, Appendix on Vector Processors
  - Russell, "The CRAY-1 computer system," CACM 1978.
  - Lindholm et al., "NVIDIA Tesla: A Unified Graphics and Computing Architecture," IEEE Micro 2008.

Vector Machine Example: CRAY-1



- CRAY-1
- Russell, "The CRAY-1 computer system," CACM 1978.

- Scalar and vector modes
- 8 64-element vector registers
- 64 bits per element
- 16 memory banks
- 8 64-bit scalar registers
- 8 24-bit address registers

Another Way of Exploiting Parallelism

- SIMD:
  - Concurrency arises from performing the same operations on different pieces of data
- MIMD:
  - Concurrency arises from performing different operations on different pieces of data
    - Control/thread parallelism: execute different threads of control in parallel → multithreading, multiprocessing
  - Idea: Use multiple processors to solve a problem

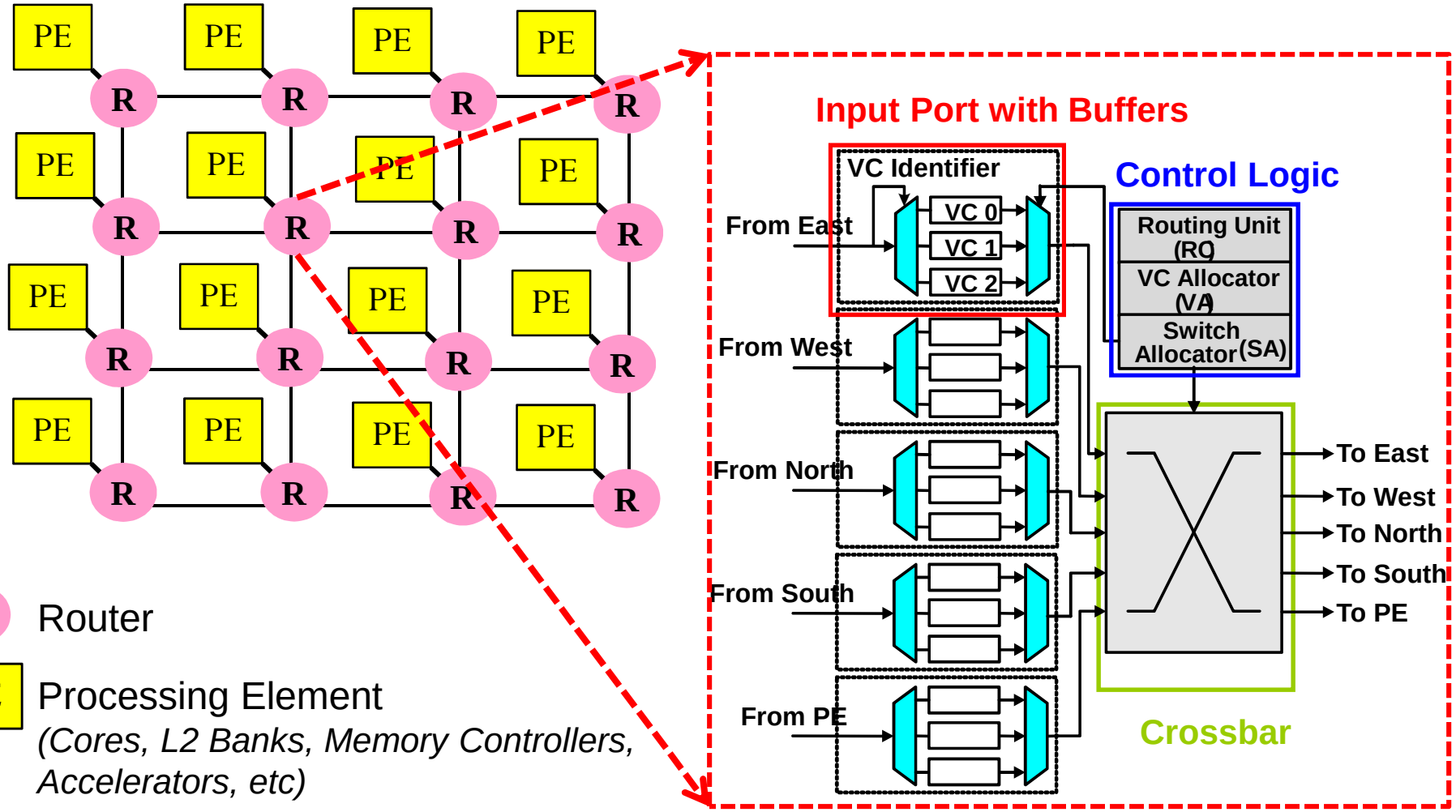
Flynn's Taxonomy of Computers

- Mike Flynn, "Very High-Speed Computing Systems," Proc. of IEEE, 1966

- SISD: Single instruction operates on single data element
- SIMD: Single instruction operates on multiple data elements
  - Array processor
  - Vector processor
- MISD: Multiple instructions operate on single data element
  - Closest form: systolic array processor, streaming processor
- MIMD: Multiple instructions operate on multiple data elements (multiple instruction streams)
  - Multiprocessor
  - Multithreaded processor

On-Chip Network Based Multi-Core Systems

- A scalable multi-core is a distributed system on a chip



Idea of On-Chip Networks

- Problem: Connecting many cores with a single bus is not scalable
  - Single point of connection limits communication bandwidth
    - What if multiple core pairs want to communicate with each other at the same time?
  - Electrical loading on the single bus limits bus frequency

- Idea: Use a network to connect cores
  - Connect neighboring cores via short links
  - Communicate between cores by routing packets over the network

- Dally and Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks," DAC 2001.

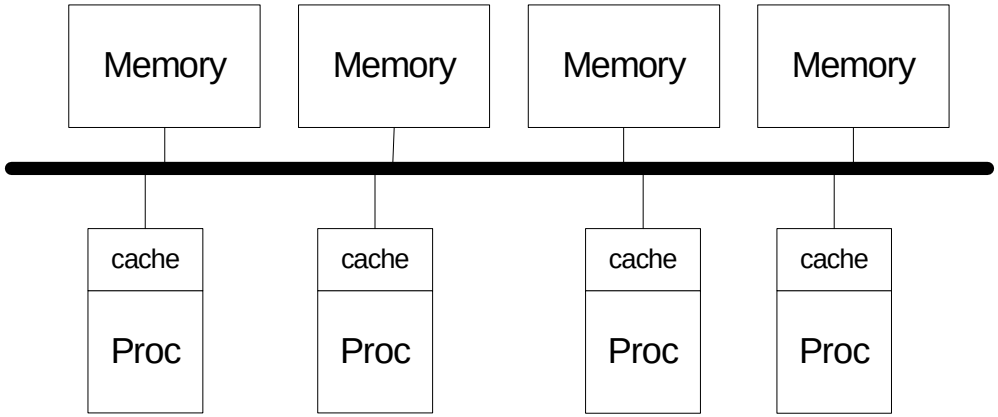
Advantages/Disadvantages of NoCs

- Advantages compared to bus
  - + More links → more bandwidth → multiple core-to-core transactions can occur in parallel in the system (no single point of contention) → higher performance
  - + Links are short and less loaded → high frequency
  - + More scalable system → more components/cores can be supported on the network than on a single bus
  - + Eliminates single point of failure

- Disadvantages
  - - Requires routers that can route data/control packets → costs area, power, complexity
  - - Maintaining cache coherence is more complex

Bus

+ Simple
+ Cost effective for a small number of nodes
+ Easy to implement coherence (snooping)
- Not scalable to large number of nodes (limited bandwidth, electrical loading → reduced frequency)
- High contention



Crossbar

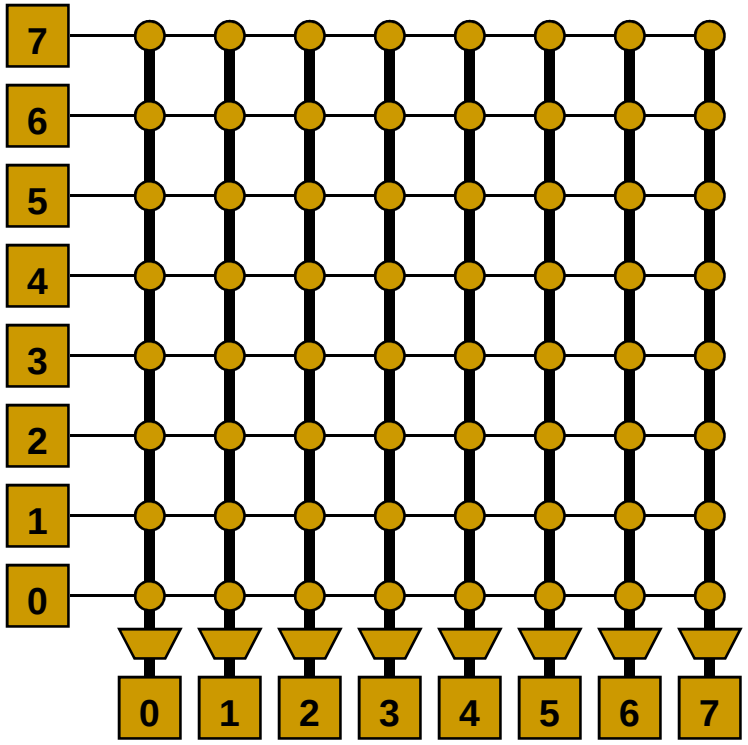
- Every node connected to every other
- Good for small number of nodes

+ Least contention in the network: high bandwidth

- Expensive
- Not scalable due to quadratic cost

Used in core-to-cache-bank networks in

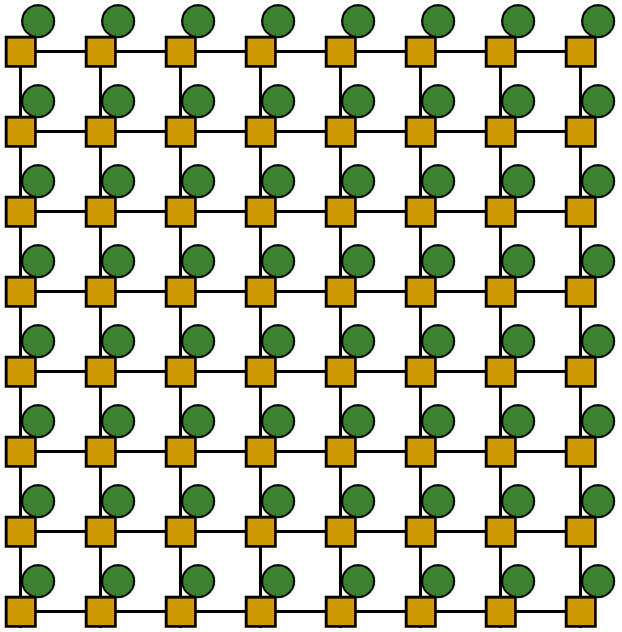
- IBM POWER5
- Sun Niagara I/II



Mesh

- O(N) cost
- Average latency: O(sqrt(N))
- Easy to layout on-chip: regular and equal-length links
- Path diversity: many ways to get from one node to another

- Used in Tilera 100-core
- And many on-chip network prototypes



Torus

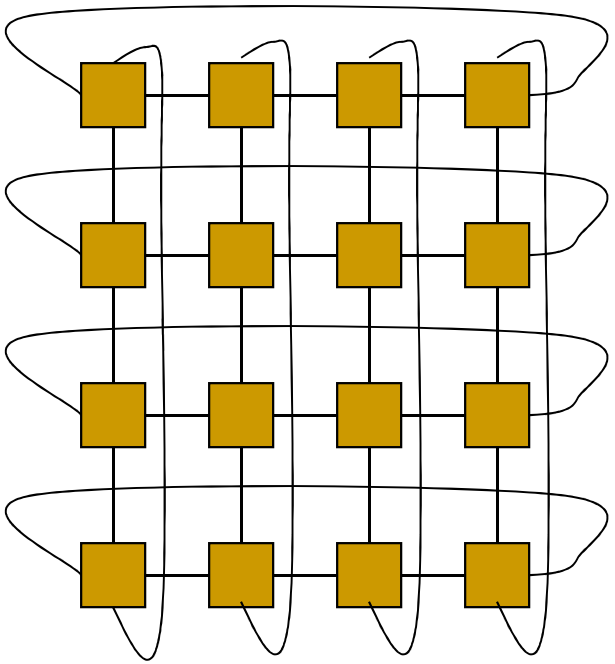
- Mesh is not symmetric on edges: performance very sensitive to placement of task on edge vs. middle
- Torus avoids this problem

+ Higher path diversity than mesh

- Higher cost

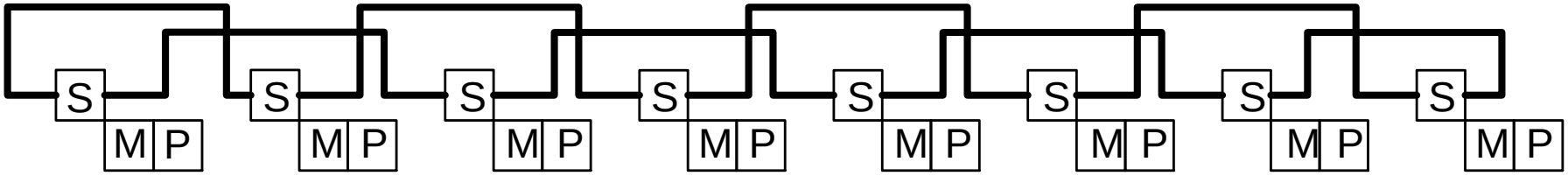
- Harder to lay out on-chip

  - Unequal link lengths



Torus, continued

- Weave nodes to make inter-node latencies ~constant



Example NoC: 100-core Tilera Processor

- Wentzlaff et al., "On-Chip Interconnection Architecture of the Tile Processor," IEEE Micro 2007.