18-742 Fall 2012
Parallel Computer Architecture
Lecture 4: Multi-Core Processors

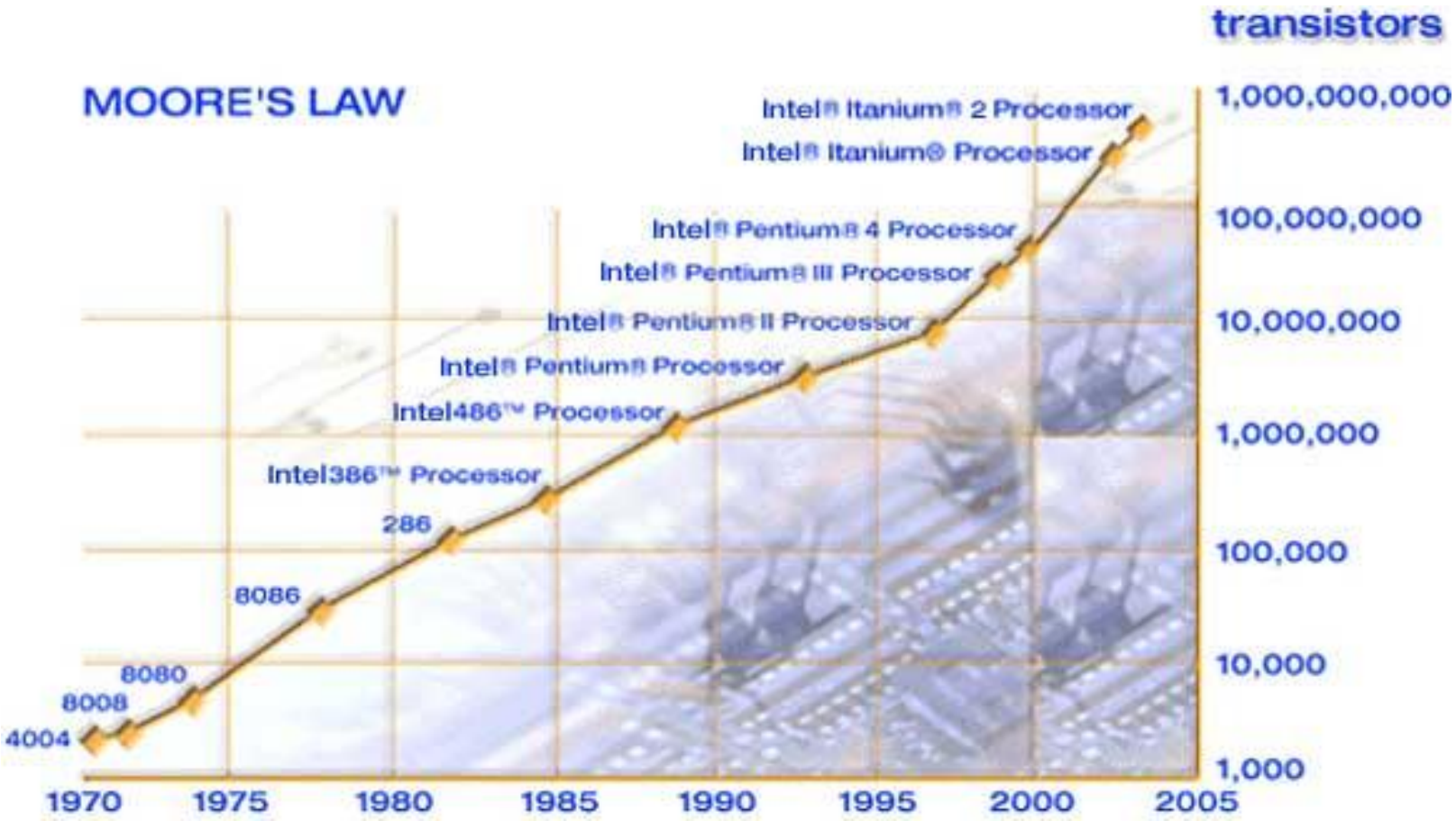
Prof. Onur Mutlu
Carnegie Mellon University
9/14/2012

Reminder: Reviews Due Sunday

- Sunday, September 16, 11:59pm.
- Suleman et al., "Accelerating Critical Section Execution with Asymmetric Multi-Core Architectures," ASPLOS 2009.
- Suleman et al., "Data Marshaling for Multi-core Architectures," ISCA 2010.
- Joao et al., "Bottleneck Identification and Scheduling in Multithreaded Applications," ASPLOS 2012.

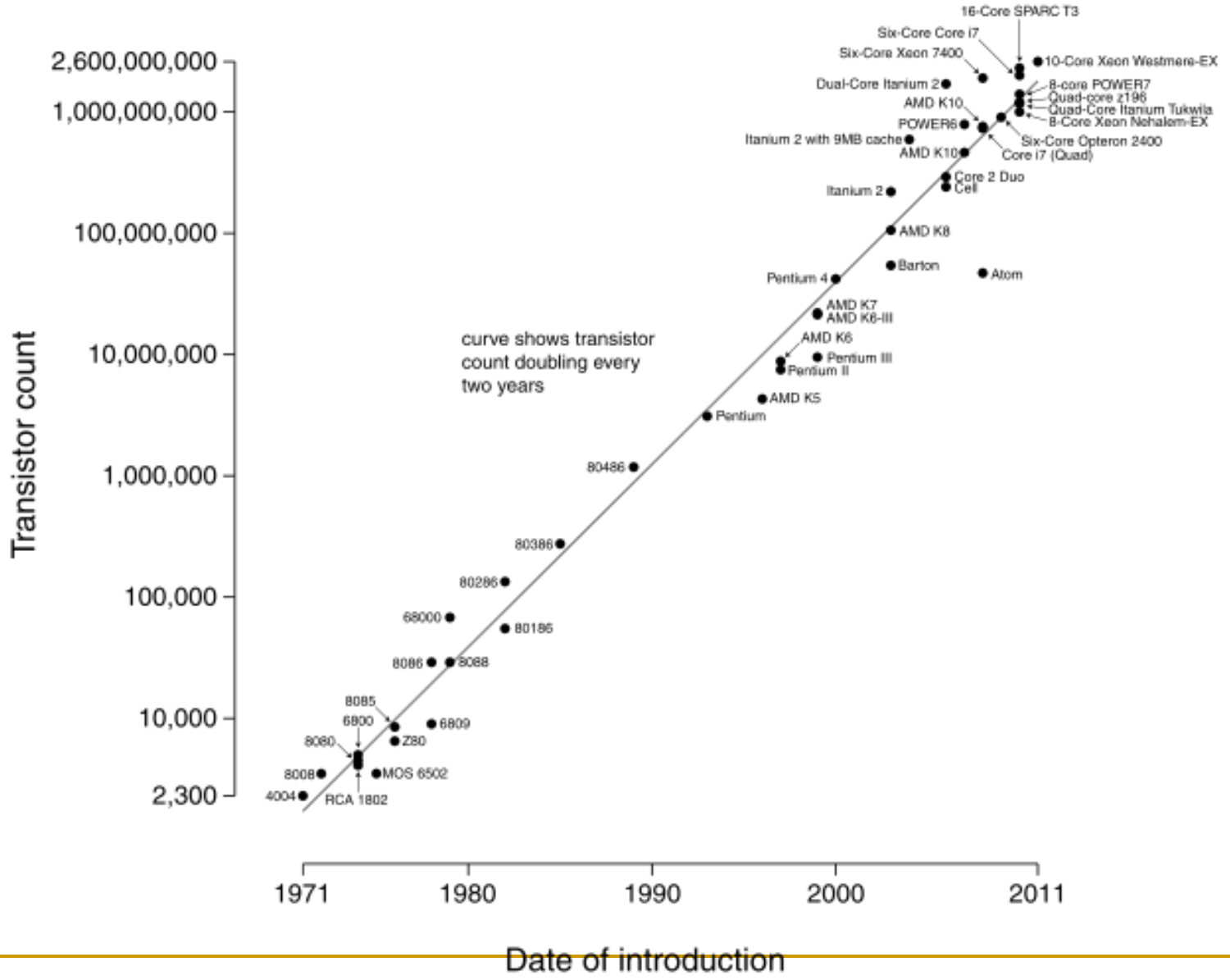
Multi-Core Processors

Moore's Law



Moore, "Cramming more components onto integrated circuits," Electronics, 1965.

Microprocessor Transistor Counts 1971-2011 & Moore's Law



Multi-Core

- Idea: Put multiple processors on the same die.
- Technology scaling (Moore's Law) enables more transistors to be placed on the same die area
- What else could you do with the die area you dedicate to multiple processors?
  - Have a bigger, more powerful core
  - Have larger caches in the memory hierarchy
  - Simultaneous multithreading
  - Integrate platform components on chip (e.g., network interface, memory controllers)

Why Multi-Core?

- Alternative: Bigger, more powerful single core
  - Larger superscalar issue width, larger instruction window, more execution units, large trace caches, large branch predictors, etc

+ Improves single-thread performance transparently to programmer, compiler

- Very difficult to design (Scalable algorithms for improving single-thread performance elusive)

- Power hungry – many out-of-order execution structures consume significant power/area when scaled. Why?

- Diminishing returns on performance

- Does not significantly help memory-bound application performance (Scalable algorithms for this elusive)

Large Superscalar vs. Multi-Core

- Olukotun et al., "The Case for a Single-Chip Multiprocessor," ASPLOS 1996.

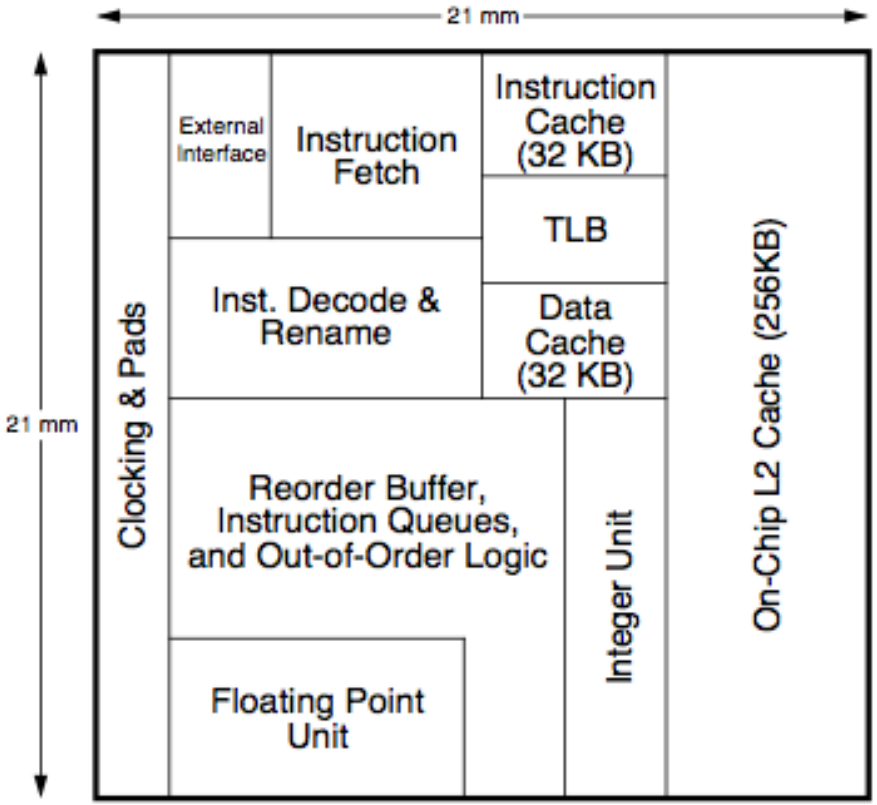


Figure 2. Floorplan for the six-issue dynamic superscalar microprocessor.

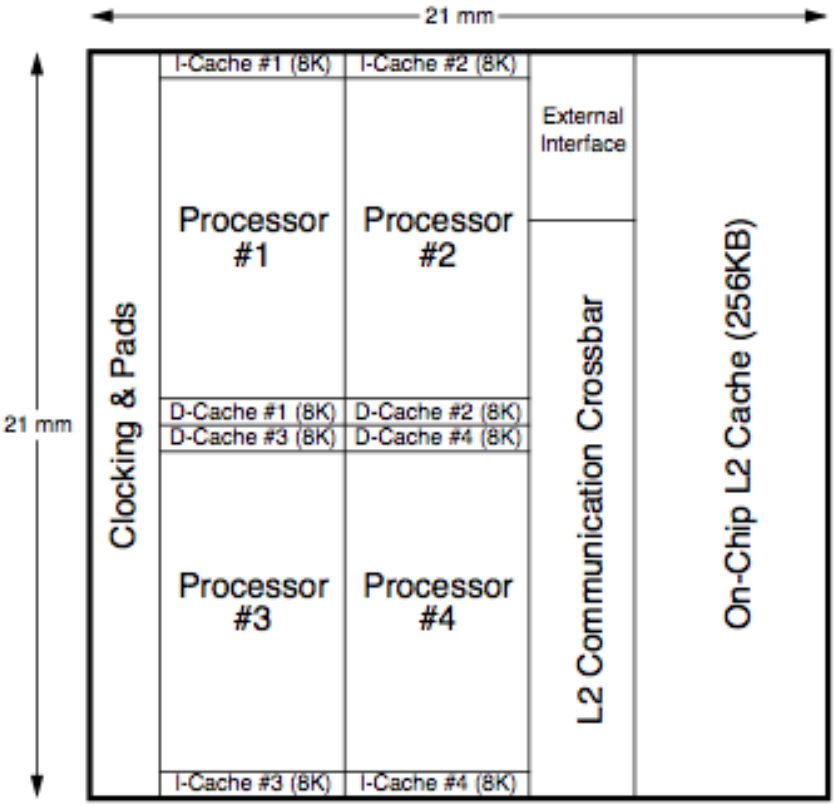


Figure 3. Floorplan for the four-way single-chip multiprocessor.

Multi-Core vs. Large Superscalar

- Multi-core advantages
  + Simpler cores → more power efficient, lower complexity, easier to design and replicate, higher frequency (shorter wires, smaller structures)
  + Higher system throughput on multiprogrammed workloads → reduced context switches
  + Higher system throughput in parallel applications

- Multi-core disadvantages
  - Requires parallel tasks/threads to improve performance (parallel programming)
  - Resource sharing can reduce single-thread performance
  - Shared hardware resources need to be managed
  - Number of pins limits data supply for increased demand

Large Superscalar vs. Multi-Core

- Olukotun et al., "The Case for a Single-Chip Multiprocessor," ASPLOS 1996.

- Technology push
  - Instruction issue queue size limits the cycle time of the superscalar, OoO processor → diminishing performance
    - Quadratic increase in complexity with issue width
  - Large, multi-ported register files to support large instruction windows and issue widths → reduced frequency or longer RF access, diminishing performance
- Application pull
  - Integer applications: little parallelism?
  - FP applications: abundant loop-level parallelism
  - Others (transaction proc., multiprogramming): CMP better fit

Comparison Points…

	6-way SS	4x2-way MP
# of CPUs	1	4
Degree superscalar	6	4 x 2
# of architectural registers	32int / 32fp	4 x 32int / 32fp
# of physical registers	160int / 160fp	4 x 40int / 40fp
# of integer functional units	3	4 x 1
# of floating pt. functional units	3	4 x 1
# of load/store ports	8 (one per bank)	4 x 1
BTB size	2048 entries	4 x 512 entries
Return stack size	32 entries	4 x 8 entries
Instruction issue queue size	128 entries	4 x 8 entries
I cache	32 KB, 2-way S.A.	4 x 8 KB, 2-way S.A.
D cache	32 KB, 2-way S.A.	4 x 8 KB, 2-way S.A.
L1 hit time	2 cycles (4 ns)	1 cycle (2 ns)
L1 cache interleaving	8 banks	N/A
Unified L2 cache	256 KB, 2-way S.A.	256 KB, 2-way S.A.
L2 hit time / L1 penalty	4 cycles (8 ns)	5 cycles (10 ns)
Memory latency / L2 penalty	50 cycles (100 ns)	50 cycles (100 ns)

Table 1. Key characteristics of the two microarchitectures

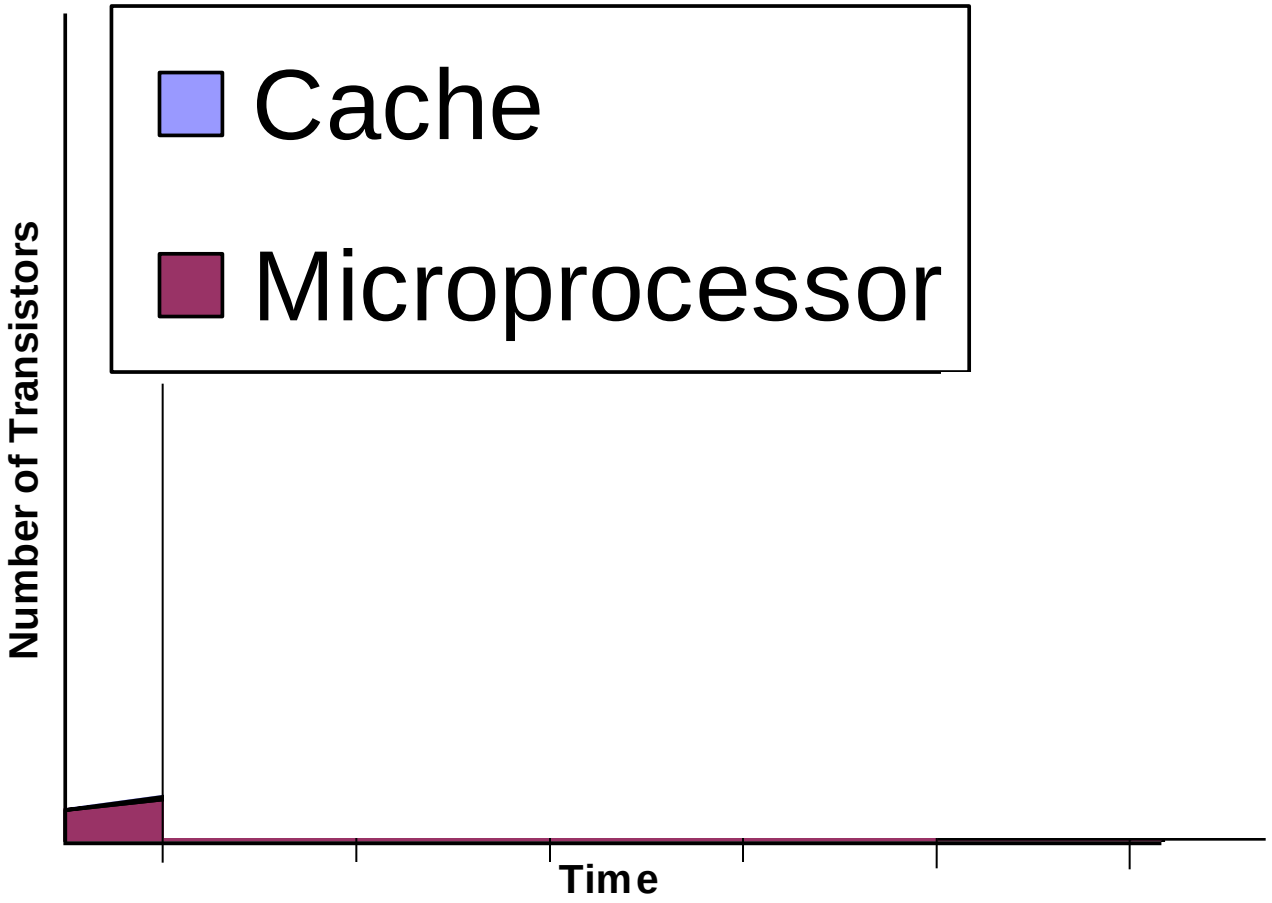
Why Multi-Core?

- Alternative: Bigger caches

  + Improves single-thread performance transparently to programmer, compiler
  + Simple to design

  - Diminishing single-thread performance returns from cache size. Why?
  - Multiple levels complicate memory hierarchy

Cache vs. Core



Why Multi-Core?

- Alternative: (Simultaneous) Multithreading

+ Exploits thread-level parallelism (just like multi-core)
+ Good single-thread performance with SMT
+ No need to have an entire core for another thread
+ Parallel performance aided by tight sharing of caches

- Scalability is limited: need bigger register files, larger issue width (and associated costs) to have many threads → complex with many threads
- Parallel performance limited by shared fetch bandwidth
- Extensive resource sharing at the pipeline and memory system reduces both single-thread and parallel application performance

Why Multi-Core?

- Alternative: Integrate platform components on chip instead

  + Speeds up many system functions (e.g., network interface cards, Ethernet controller, memory controller, I/O controller)

  - Not all applications benefit (e.g., CPU intensive code sections)

Why Multi-Core?

- Alternative: More scalable superscalar, out-of-order engines
  - Clustered superscalar processors (with multithreading)

+ Simpler to design than superscalar, more scalable than simultaneous multithreading (less resource sharing)
+ Can improve both single-thread and parallel application performance

- Diminishing performance returns on single thread: Clustering reduces IPC performance compared to monolithic superscalar. Why?
- Parallel performance limited by shared fetch bandwidth
- Difficult to design

Clustered Superscalar+OoO Processors

- Clustering (e.g., Alpha 21264 integer units)
  - Divide the scheduling window (and register file) into multiple clusters
  - Instructions steered into clusters (e.g. based on dependence)
  - Clusters schedule instructions out-of-order, within cluster scheduling can be in-order
  - Inter-cluster communication happens via register files (no full bypass)

+ Smaller scheduling windows, simpler wakeup algorithms

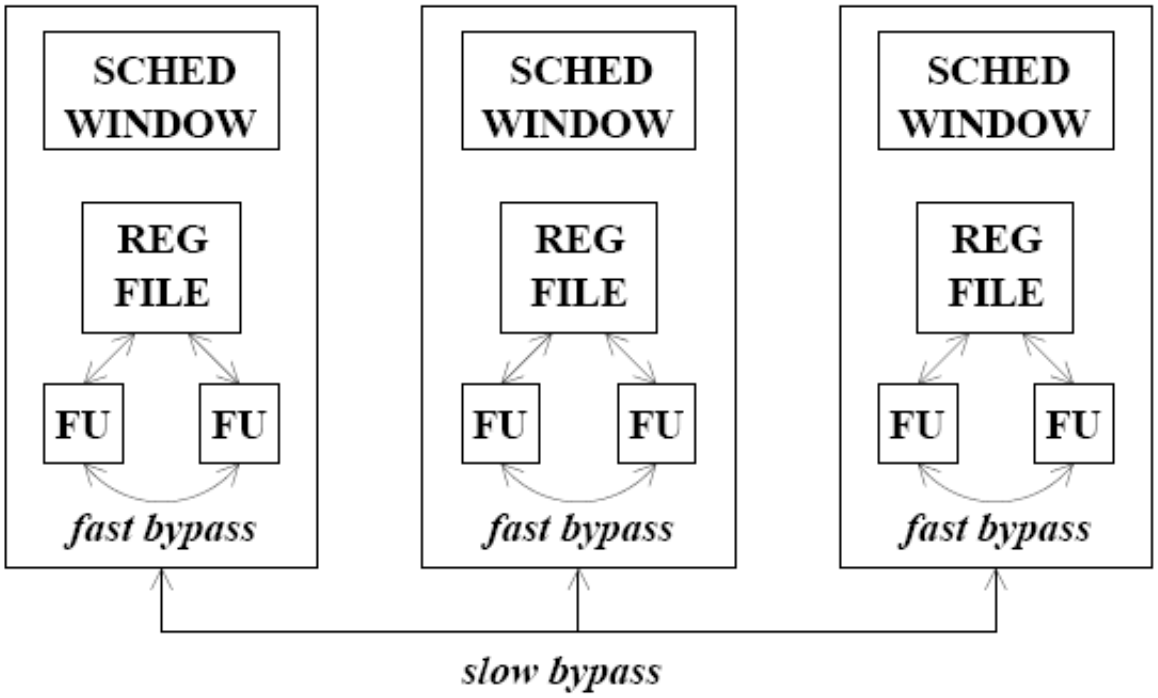
+ Smaller ports into register files

+ Faster within-cluster bypass

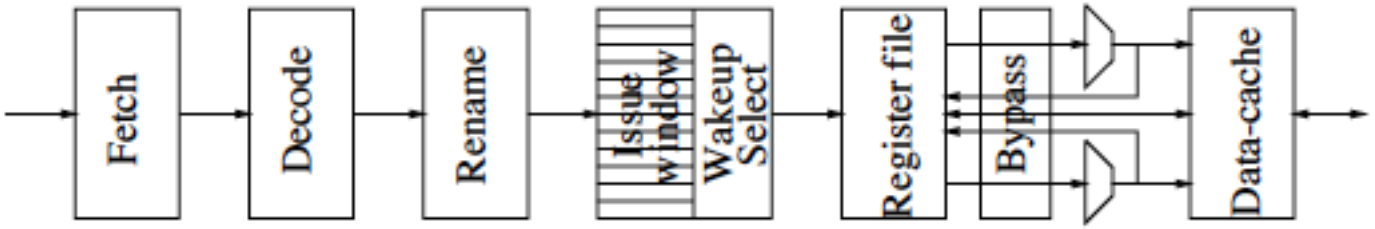
-- Extra delay when instructions require across-cluster communication

Clustering (I)

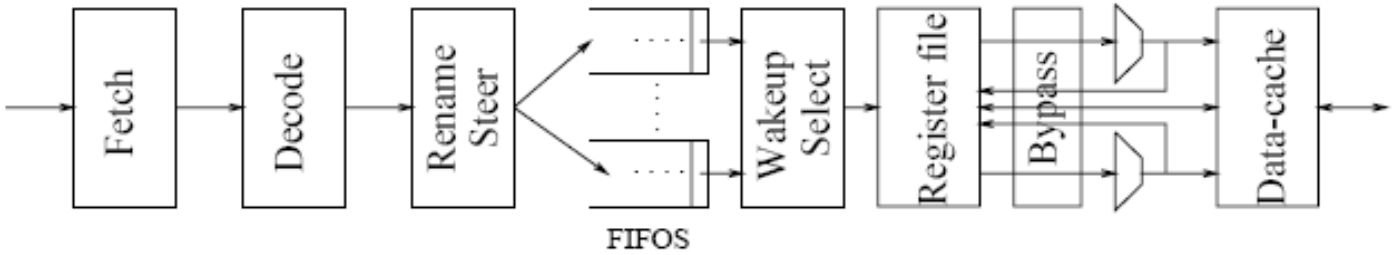
- Scheduling within each cluster can be out of order



Clustering (II)

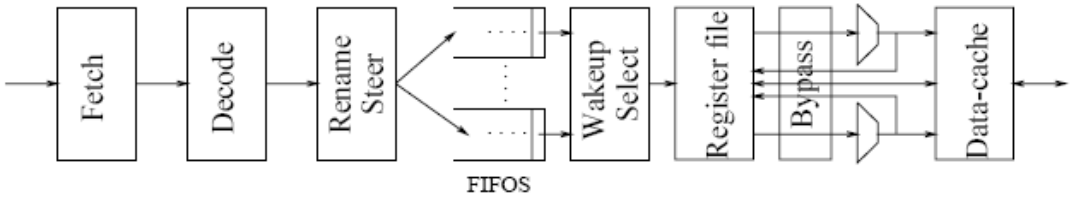


FETCH	DECODE	RENAME	WAKEUP SELECT	REG READ	EXECUTE BYPASS	DCACHE ACCESS	REG WRITE COMMIT

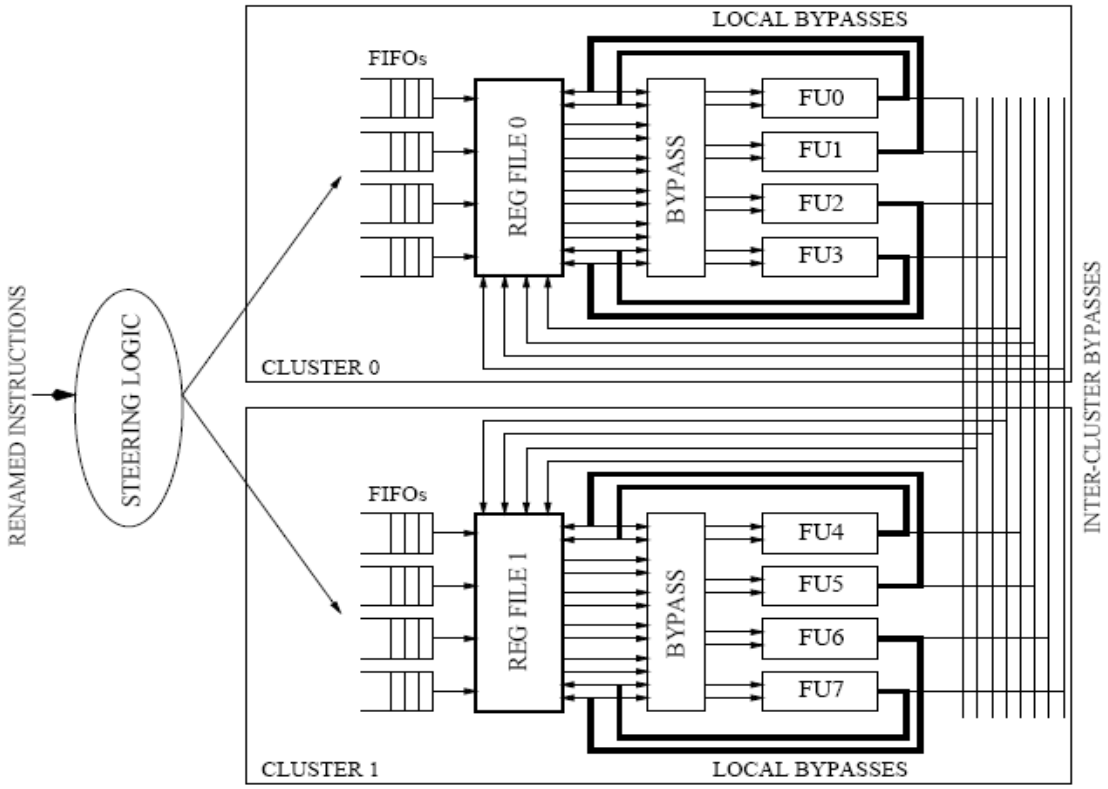


FETCH	DECODE	RENAME STEER	WAKEUP SELECT	REG READ	EXECUTE BYPASS	DCACHE ACCESS	REG WRITE COMMIT

Clustering (III)



FETCH	DECODE	RENAME STEER	WAKEUP SELECT	REG READ	EXECUTE BYPASS	DCACHE ACCESS	REG WRITE COMMIT



Each scheduler is a FIFO
+ Simpler
+ Can have N FIFOs
(OoO w.r.t. each other)
+ Reduces scheduling complexity
-- More dispatch stalls

Inter-cluster bypass: Results produced by an FU in Cluster 0 is not individually forwarded to each FU in another cluster.

- Palacharla et al., "Complexity Effective Superscalar Processors," ISCA 1997.

Why Multi-Core?

- Alternative: Traditional symmetric multiprocessors

  + Smaller die size (for the same processing core)
  + More memory bandwidth (no pin bottleneck)
  + Fewer shared resources → less contention between threads

  - Long latencies between cores (need to go off chip) → shared data accesses limit performance → parallel application scalability is limited
  - Worse resource efficiency due to less sharing → worse power/energy efficiency

Why Multi-Core?

- Other alternatives?
  - Dataflow?
  - Vector processors (SIMD)?
  - Integrating DRAM on chip?
  - Reconfigurable logic? (general purpose?)

Review: Multi-Core Alternatives

- Bigger, more powerful single core
- Bigger caches
- (Simultaneous) multithreading
- Integrate platform components on chip instead
- More scalable superscalar, out-of-order engines
- Traditional symmetric multiprocessors
- Dataflow?
- Vector processors (SIMD)?
- Integrating DRAM on chip?
- Reconfigurable logic? (general purpose?)
- Other alternatives?