Thread-Level Parallelism

Thread-Level Parallelism in Multithreaded, and Multicore Processors

- Multithreading
- Multithreaded systems
- Types of multithreading
- Simultaneous multithreading
- Design challenges
- SMT performance
- Modeling performance
- Comparing multicore systems

ILP to TLP

- Limits to ILP (power efficiency, compilers, dependencies…) seem to limit to three to six issues/cycle for practical options.

- Explicitly parallel (thread-level or data-level parallelism) is next step for better performance.

Performance beyond Single-Thread ILP

- There can be much higher natural parallelism in some applications (e.g., database or scientific codes).
- Explicit thread-level parallelism or data-level parallelism.
- **Thread**: a process with its own instructions and data (or much harder on compiler: carefully selected rarely interacting code segments in the same process).
  - Thread may be one process that is part of a parallel program of multiple processes, or it may be an independent program.
  - Each thread has all the state (instructions, data, PC, register state, and so on) necessary to allow it to execute.
- **Data-level parallelism:** Perform identical operations on data and have lots of data.

New Approach: Multithreaded Execution

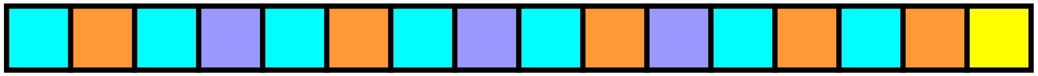
- **Multithreading**: multiple threads to share the functional units of one processor via overlapped execution
  - Processor must **duplicate** independent state of each thread, for example, a separate copy of register file, a separate PC, and, if running as independent programs, a separate page table.
  - **Memory shared** through the virtual memory mechanisms, which already support multiple processes.
  - HW for **fast thread switch** (0.1 to 10 clocks) is much faster than a full process switch (100s to 1000s of clocks) that copies state.
- When to switch among threads?
  - Alternate instruction per thread (fine grain).
  - When a thread is stalled, perhaps for a cache miss, another thread can be executed (coarse grain).
  - In cacheless multiprocessors, at start of each memory access.

Multithreading Costs

- Each thread requires its own user state.
  - PC
  - GPRs
- Also, needs its own system state.
  - Virtual memory page table base register
  - Exception handling registers
- Other overheads:
  - Additional cache/TLB conflicts from competing threads
  - (Or add larger cache/TLB capacity)
  - More OS overhead to schedule more threads (where do all these threads come from?)

Thread-Scheduling Policies

- Fixed interleave (CDC 6600 PPUs, 1964)
  - Each of N threads executes one instruction every N cycles.
  - If thread not ready to go in its slot, insert pipeline bubble.
- Software-controlled interleave (TI ASC PPUs, 1971)
  - OS allocates S pipeline slots amongst N threads.
  - Hardware performs fixed interleave over S slots, executing whichever thread is in that slot.
- Hardware-controlled thread scheduling (HEP, 1982)
  - Hardware keeps track of which threads are ready to go.
  - Picks next thread to execute based on hardware priority scheme.



ENGINEERING@SYRACUSE

Multithreaded Systems

Denelcor Heterogeneous Element Processor (HEP) (Smith 1982)

- First commercial machine to use hardware threading in main CPU
  - 120 threads per processor
  - 10 MHz clock rate
  - Up to 8 processors
  - Precursor to Tera MTA (multithreaded architecture)

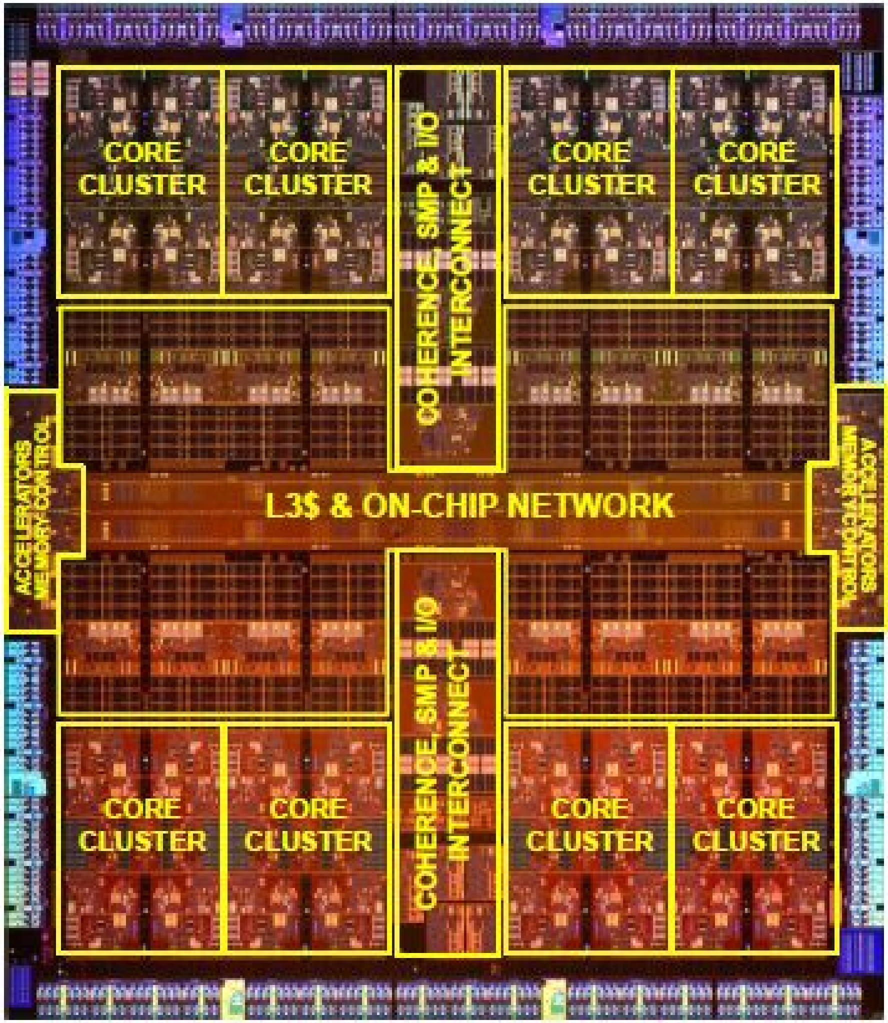
IBM PowerPC RS64-IV (2000)

- Commercial coarse-grain multithreading CPU.
- Based on PowerPC with quad-issue in-order five-stage pipeline.
- Each physical CPU supports two virtual CPUs.
- On L2 cache miss, pipeline is flushed, and execution switches to second thread.
  - Short pipeline minimizes flush penalty (four cycles), small compared to memory access latency.
  - Flush pipeline to simplify exception handling.

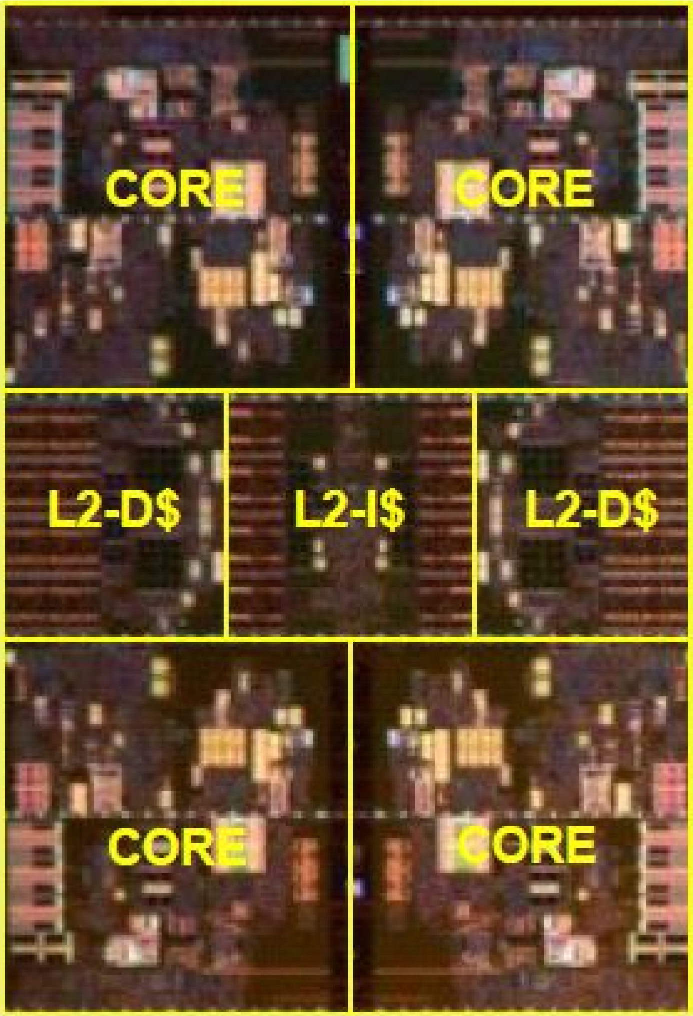
Oracle/Sun Niagara Processors

- Target is **datacenters** running web servers and databases, with many concurrent requests.
- Provide multiple simple cores each with multiple hardware threads, reduced energy/operation though much lower, single-thread performance.
- Niagara-1 [2004], eight cores, four threads/core.
- Niagara-2 [2007], eight cores, eight threads/core.
- Niagara-3 [2009], 16 cores, eight threads/core.
- SPARC M7 [2015], 32 cores, eight threads/core.
  - Can scale up to 1024 cores, 8192 threads 64 TB RAM, at speeds over 3.6GHz

SPARC M7


CORE CLUSTER
CORE CLUSTER
COHERENCE, SMP & I/O INTERCONNECT
CORE CLUSTER
CORE CLUSTER
ACCELERATORS MEMORY CONTROL
L3$ & ON-CHIP NETWORK
ACCELERATORS MEMORY CONTROL
CORE CLUSTER
CORE CLUSTER
COHERENCE, SMP & I/O INTERCONNECT
CORE CLUSTER
CORE CLUSTER

The die for Oracle's SPARC M7 CPU;
10 billion transistors on it



Inside a SPARC M7 core cluster

Simultaneous Multithreading (SMT) for Out-of-Order (OoO) Superscalars

- Techniques presented so far have all been “vertical” multithreading where each pipeline stage works on one thread at a time.
- SMT uses fine-grain control already present inside an OoO superscalar to allow instructions from multiple threads to enter execution on same clock cycle.
- Gives better utilization of machine resources.

Fine-Grained Multithreading

- Switches between threads on each instruction, causing the execution of multiples threads to be interleaved.
- Usually done in a round-robin fashion, skipping any stalled threads.
- CPU must be able to switch threads every clock.
- Advantage: hides both short and long stalls, since instructions from other threads execute when one thread stalls.
- Disadvantage: slows down execution of individual threads, since a thread ready to execute without stalls will be delayed by instructions from other threads.
- Used on Sun’s Niagara chip.

ENGINEERING@SYRACUSE

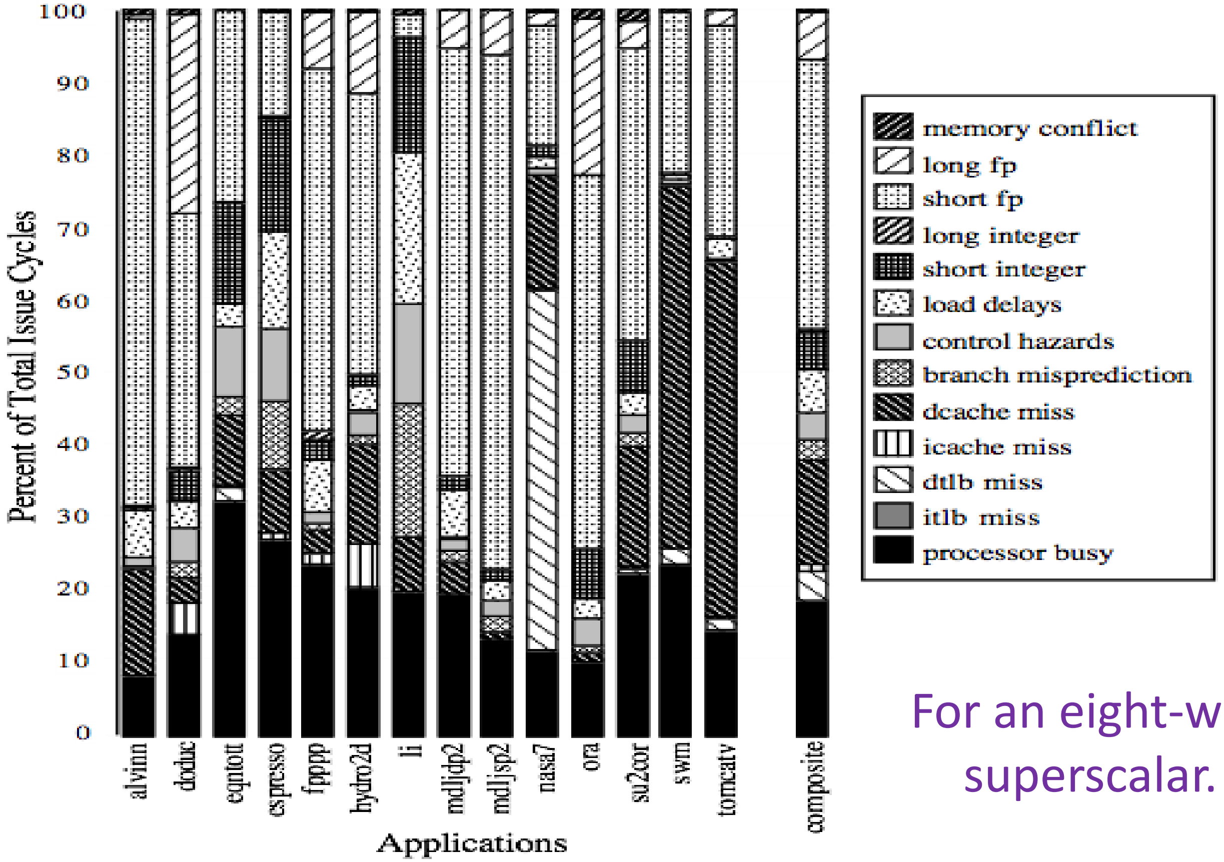
Types of Multithreading

Course-Grained Multithreading

- Switches threads only on costly stalls, such as L2 cache misses.
- Advantage is no need for very fast thread-switching and does not slow down any thread, since instructions from other threads issued only when the thread encounters a costly stall.
- Disadvantage is that it is hard to overcome throughput losses from shorter stalls, because of pipeline start-up costs.
  - Since CPU normally issues instructions from just one thread, when a stall occurs, the pipeline must be emptied or frozen.
  - New thread must fill pipeline before instructions can complete.
- Because of this start-up overhead, coarse-grained multithreading is efficient for reducing penalty only of high-cost stalls, where pipeline refill << stall time.
- Used IBM AS/400 (1988, for small to medium business).

Stalls during Executions

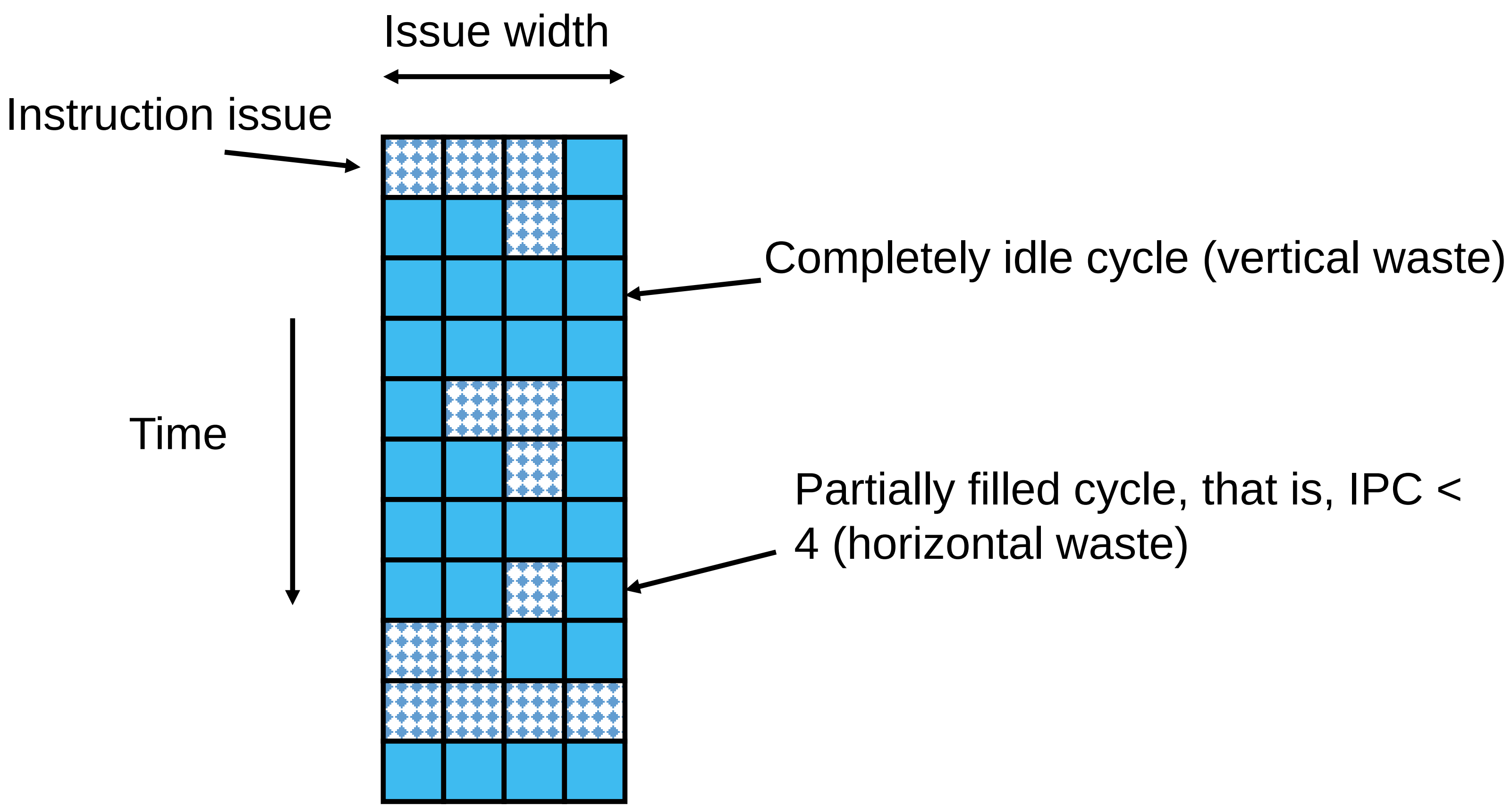
- For most applications, the execution units stall 80 percent or more of time during “execution.”



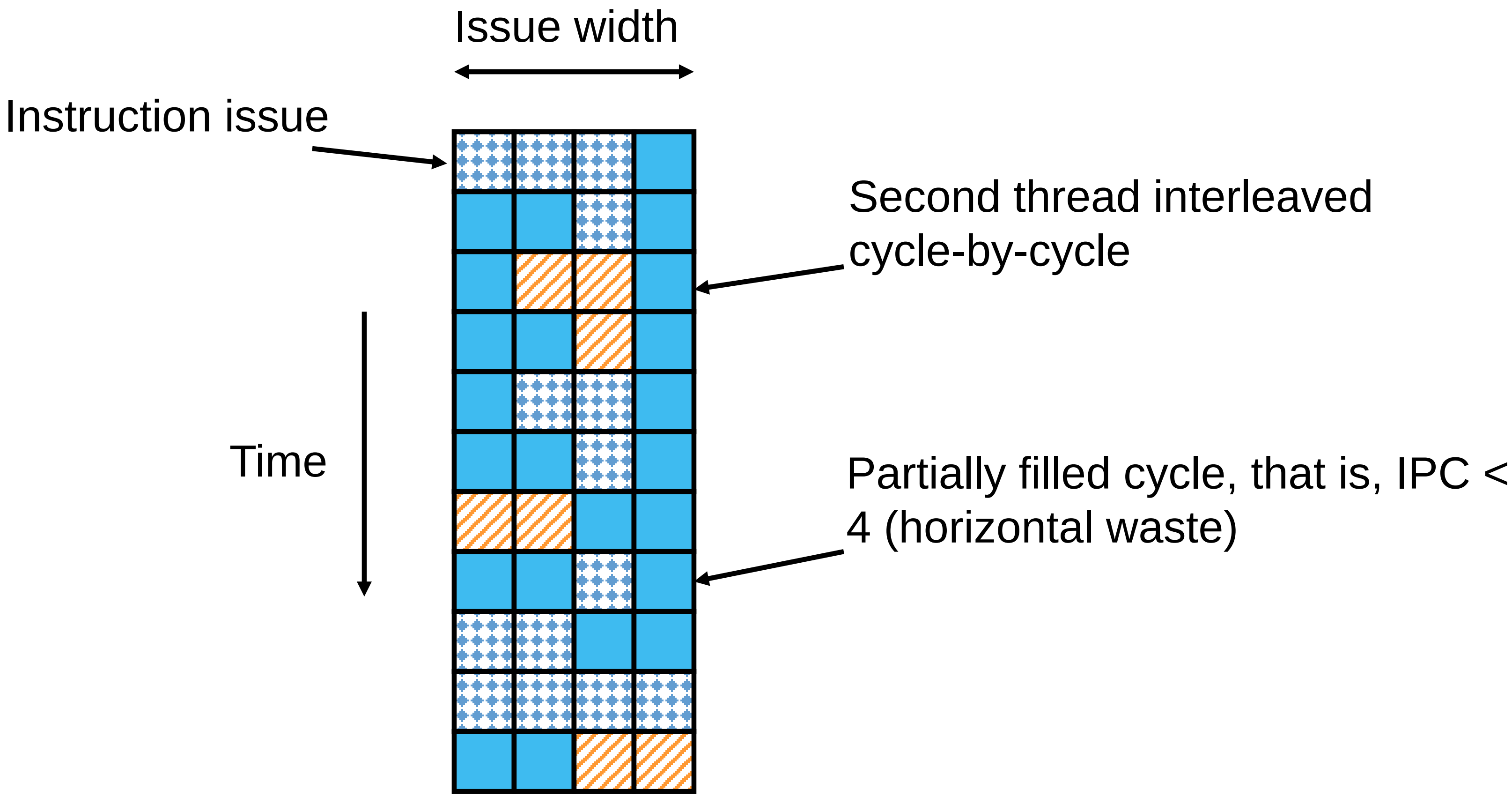
18 percent CPU usefully busy

For an eight-way superscalar.

Superscalar Machine Efficiency

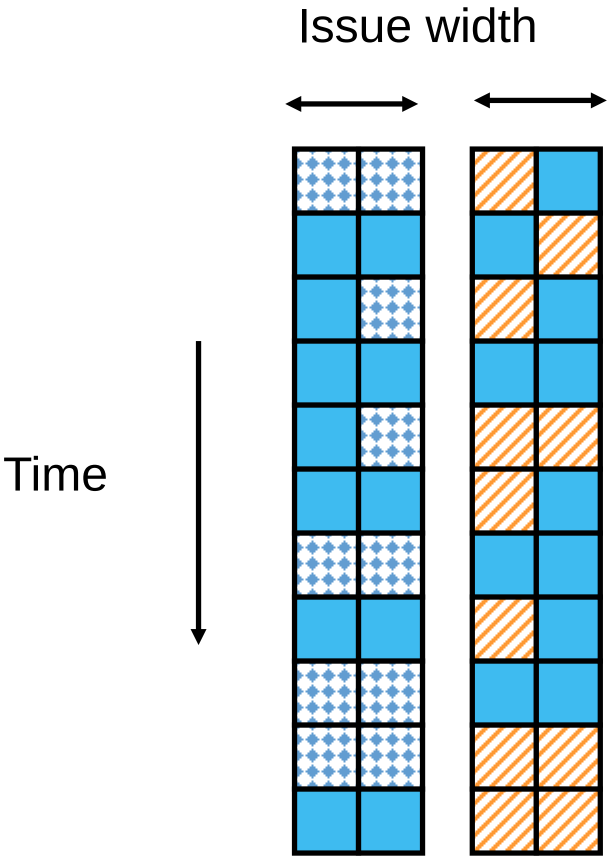

Issue width
Instruction issue
Time
Completely idle cycle (vertical waste)
Partially filled cycle, that is, IPC < 4 (horizontal waste)

Vertical Multithreading


Issue width
Instruction issue
Time
Second thread interleaved cycle-by-cycle
Partially filled cycle, that is, IPC < 4 (horizontal waste)

- What is the effect of cycle-by-cycle interleaving?
  - Removes vertical waste, but leaves some horizontal waste

Chip Multiprocessing (CMP)

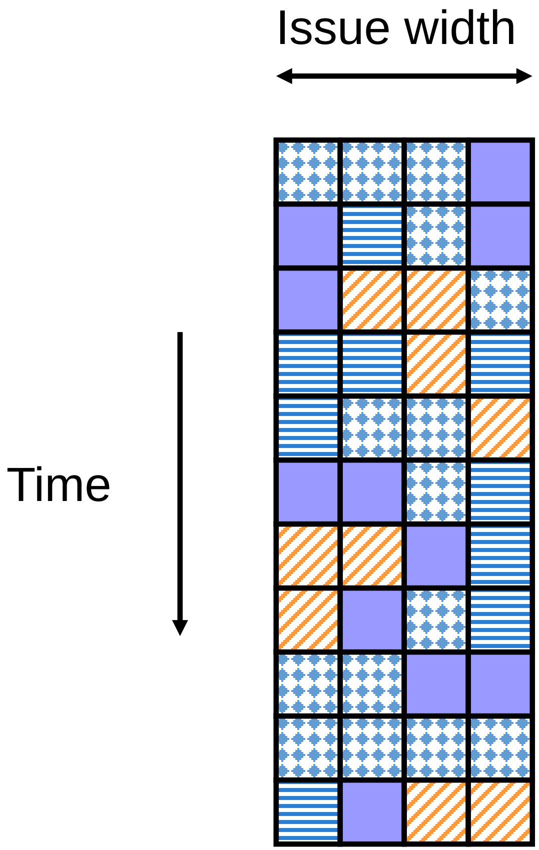


- What is the effect of splitting into multiple processors?
  - Reduces horizontal waste
  - Leaves some vertical waste
  - Puts upper limit on peak throughput of each thread

ENGINEERING@SYRACUSE

Simultaneous Threading

Ideal Superscalar Multithreading

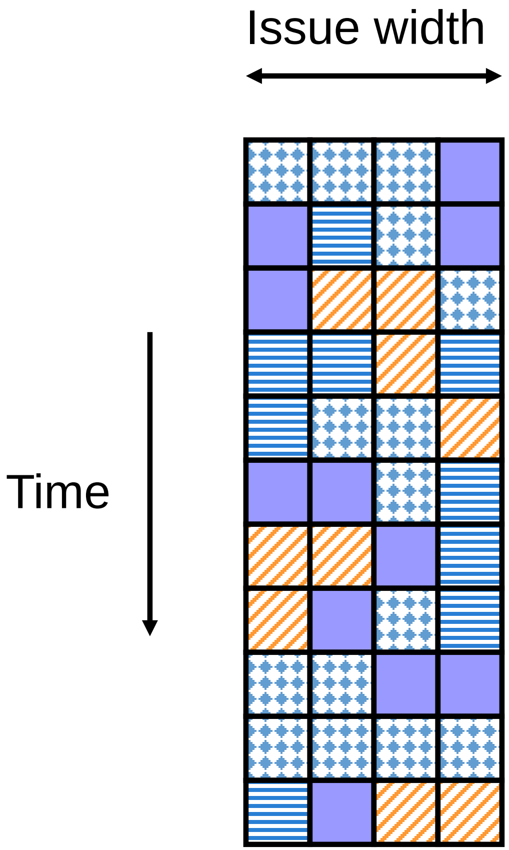


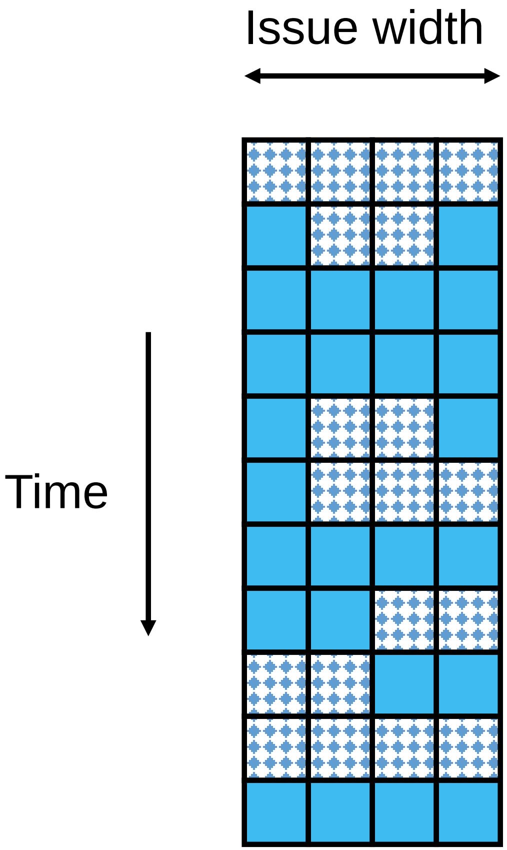
Interleave multiple threads to multiple issue slots with no restrictions.

SMT Adaptation to Parallelism

For regions with high thread-level parallelism (TLP), entire machine width is shared by all threads.

For regions with low thread-level parallelism (TLP), entire machine width is available for instruction-level parallelism (ILP).





Simultaneous Multithreading

One thread, eight functional units

Cycle	M	M	FX	FX	FP	FP	BR	CC
1	🟨							🟨
2	🟨	🟨					🟨	
3				🟨	🟨			
4								
5								
6								
7	🟨			🟨		🟨		
8		🟨			🟨			
9				🟨				

Busy: 13/72 = 18.0 percent

Two threads, eight functional units

Cycle	M	M	FX	FX	FP	FP	BR	CC
1	🟨	🟦	🟦					🟨
2	🟨	🟨	🟦			🟦	🟨	
3	🟦			🟨	🟨			
4	🟦	🟦				🟦		
5		🟦						🟦
6								
7	🟨		🟦	🟨	🟦	🟨		
8		🟨		🟦	🟨	🟦		
9	🟦	🟦		🟨		🟦		

Busy: 30/72 = 41.7 percent

M = load/store, FX = fixed point, FP = floating point, BR = branch, CC = condition codes

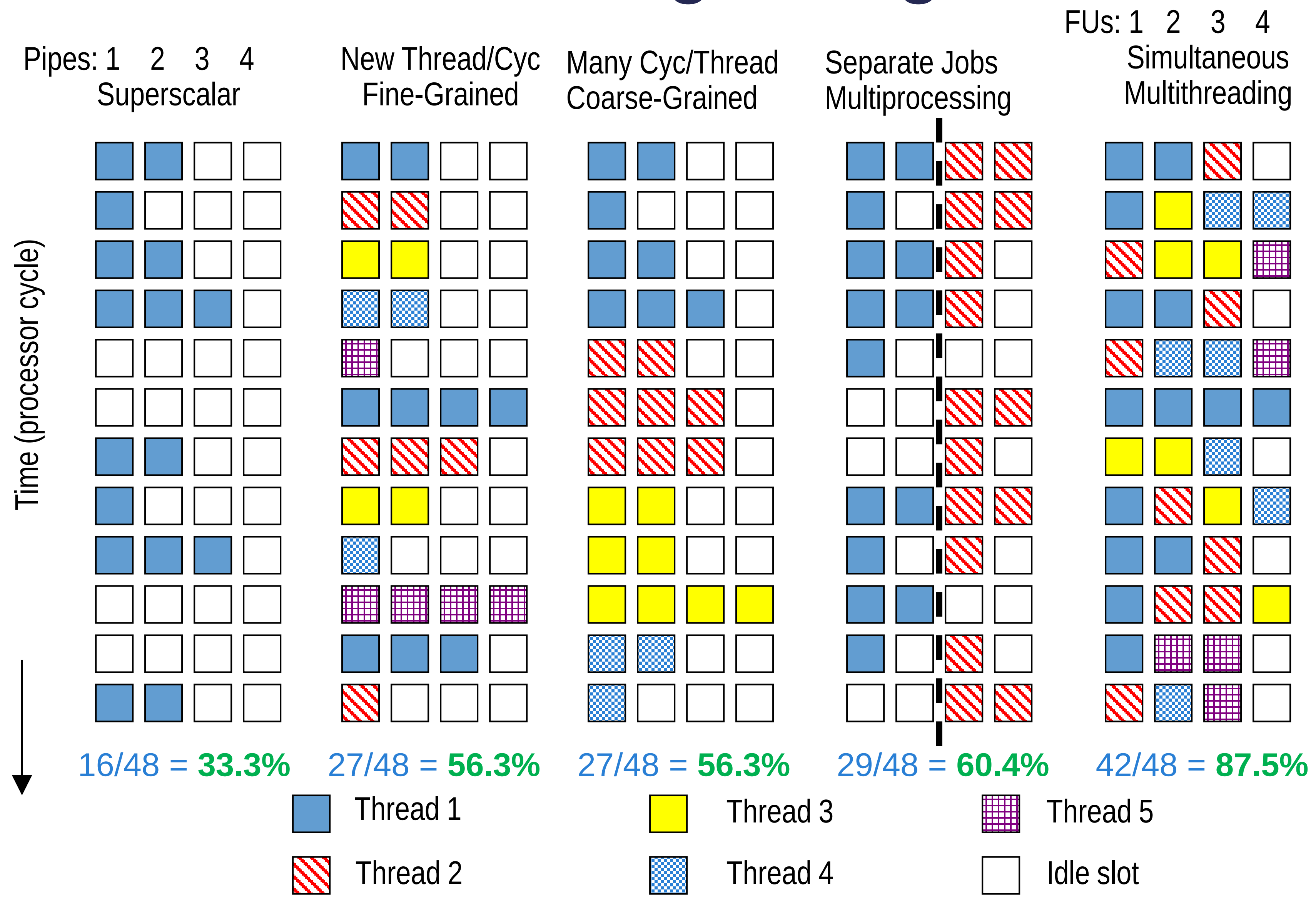
Do Both ILP and TLP?

- TLP and ILP exploit two different kinds of parallel structure in a program.
- Could a processor oriented toward ILP be used to exploit TLP?
  - Functional units are often idle in datapaths designed for ILP because of either stalls or dependences in the code.
- Could the TLP be used as a source of independent instructions that might keep the processor busy during stalls?
- Could TLP be used to employ the functional units that would otherwise lie idle when insufficient ILP exists?

Simultaneous Multithreading

- Simultaneous multithreading (SMT): insight that a dynamically scheduled processor already has many HW mechanisms to support multithreading
  - Large set of virtual registers that can be used to hold the register sets of independent threads.
  - Register renaming provides unique register identifiers, so instructions from multiple threads can be mixed in datapath without confusing sources and destinations across threads.
  - Out-of-order completion allows the threads to execute out of order and get better utilization of the HW.
- Just need to add a per-thread renaming table and keeping separate PCs
  - Independent commitment can be supported by logically keeping a separate reorder buffer for each thread.

Multithreading Categories



ENGINEERING@SYRACUSE

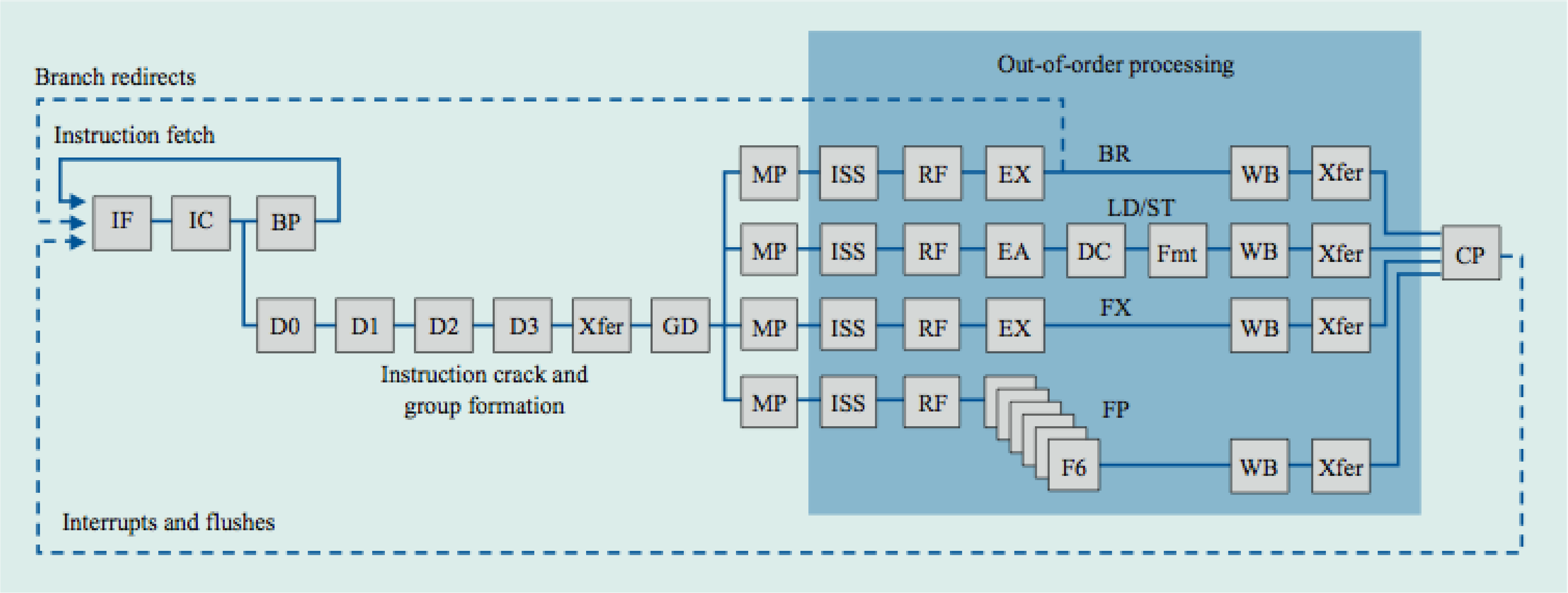
Design Challenges

Design Challenges in SMT

- Since SMT makes sense only with fine-grained implementation, what is impact of fine-grained scheduling on single-thread performance?
  - Does designating a preferred thread allow sacrificing neither throughput nor single-thread performance?
  - Unfortunately, with a preferred thread, the processor is likely to sacrifice some throughput when the preferred thread stalls.
- Larger register file needed to hold multiple contexts.
- Try not to affect clock cycle time, especially in:
  - Instruction issue: More candidate instructions need to be considered.
  - Instruction completion: Choosing which instructions to commit may be challenging.
- Ensuring that cache and TLB conflicts generated by SMT do not degrade performance.

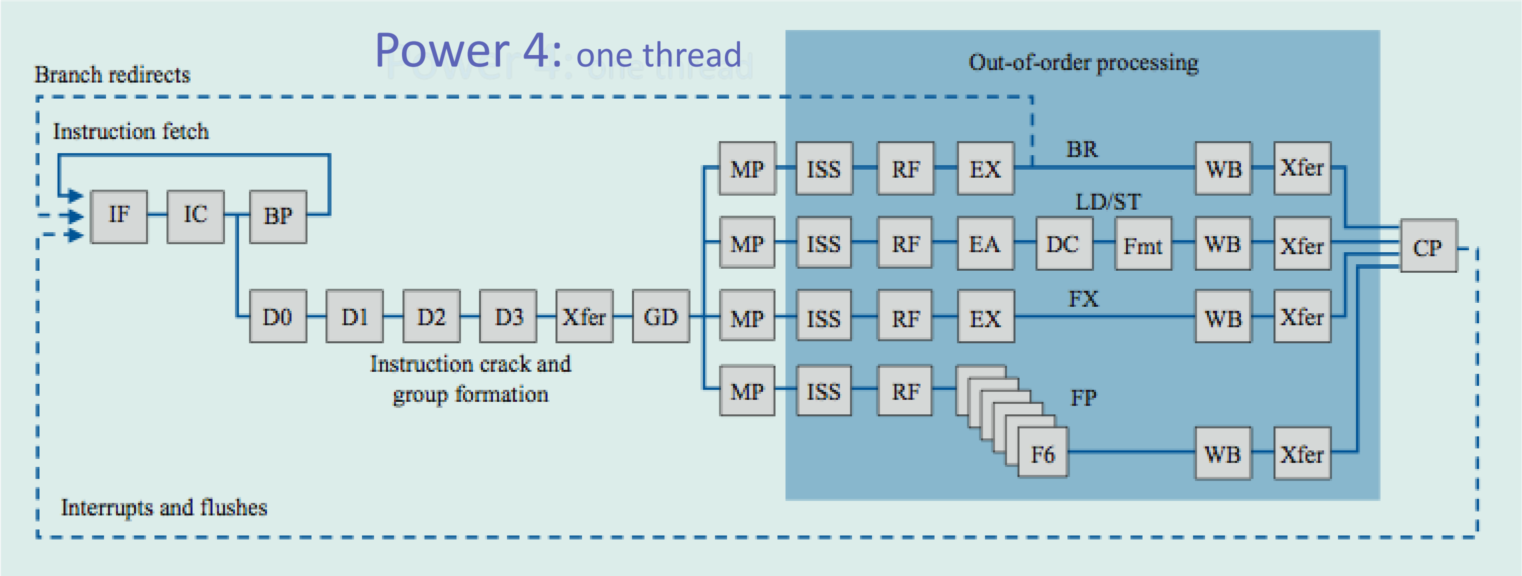
Power 4

- Single-threaded predecessor to power 5. With eight execution units in an out-of-order engine, each unit may issue one instruction each cycle.

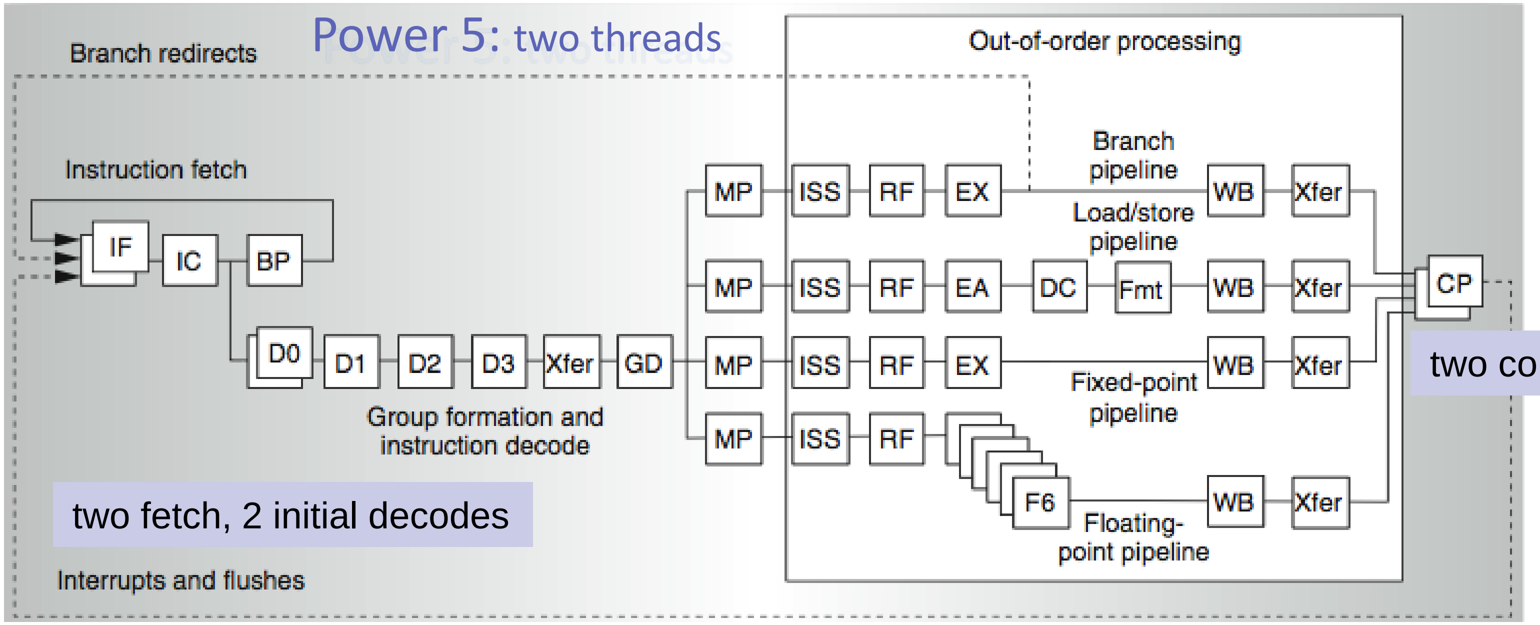


Instruction pipeline (IF: instruction fetch, IC: instruction cache, BP: branch predict, D0: decode stage 0, Xfer: transfer, GD: group dispatch, MP: mapping, ISS: instruction issue, RF: register file read, EX: execute, EA: compute address, DC: data caches, F6: six-cycle floating-point execution pipe, Fmt: data format, WB: write back, and CP: group commit)

Power 4: one thread



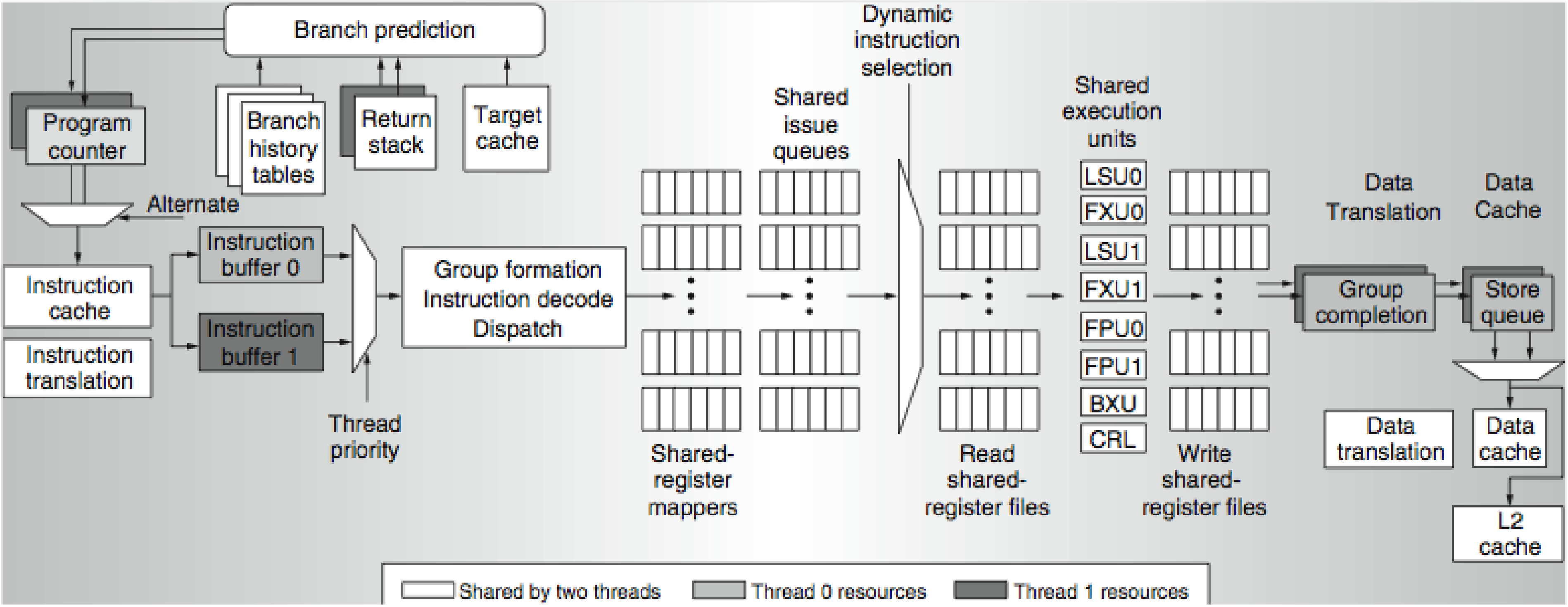
Power 5: two threads



two fetch, 2 initial decodes

two completes

Power 5 Data Flow



LSU = load/store unit, FXU = fixed-point execution unit, FPU = floating-point unit, BXU = branch execution unit, and CRL = condition register logical execution unit.

- Why only two threads?
  - With four, some shared resource (physical registers, cache, memory bandwidth) would often bottleneck.

Changes in Power 5 to Support SMT

- Increased associativity of L1 instruction cache and the instruction address translation buffers.
- Added per thread load and store queues.
- Increased size of the L2 (1.92 vs. 1.44 MB) and L3 caches.
- Added separate instruction prefetch and buffering per thread.
- Increased the number of virtual registers from 152 to 240
- Increased the size of several issue queues.
- The Power 5 core is about 24 percent larger than the Power 4 core because of the addition of SMT support.

ENGINEERING@SYRACUSE

SMT Performance

Initial Performance of SMT

- Pentium 4 Extreme SMT yields 1.01 speedup for SPECint_rate benchmark and 1.07 for SPECfp_rate.
  - Pentium 4 is dual-threaded SMT.
  - SPECRate requires that each SPEC benchmark be run against a vendor-selected number of copies of the same benchmark.
- Running on Pentium 4 with each of 26 SPEC benchmarks paired with every other (26*26 runs) gave speed-ups from 0.90 to 1.58; average was 1.20.
- Power 5, eight-processor server 1.23 faster for SPECint_rate with SMT, 1.16 faster for SPECfp_rate.
- Power 5 running two copies of each application gave speed-ups between 0.89 and 1.41.
- Most gained some.
- Floatin-point applications had most cache conflicts and least gains.

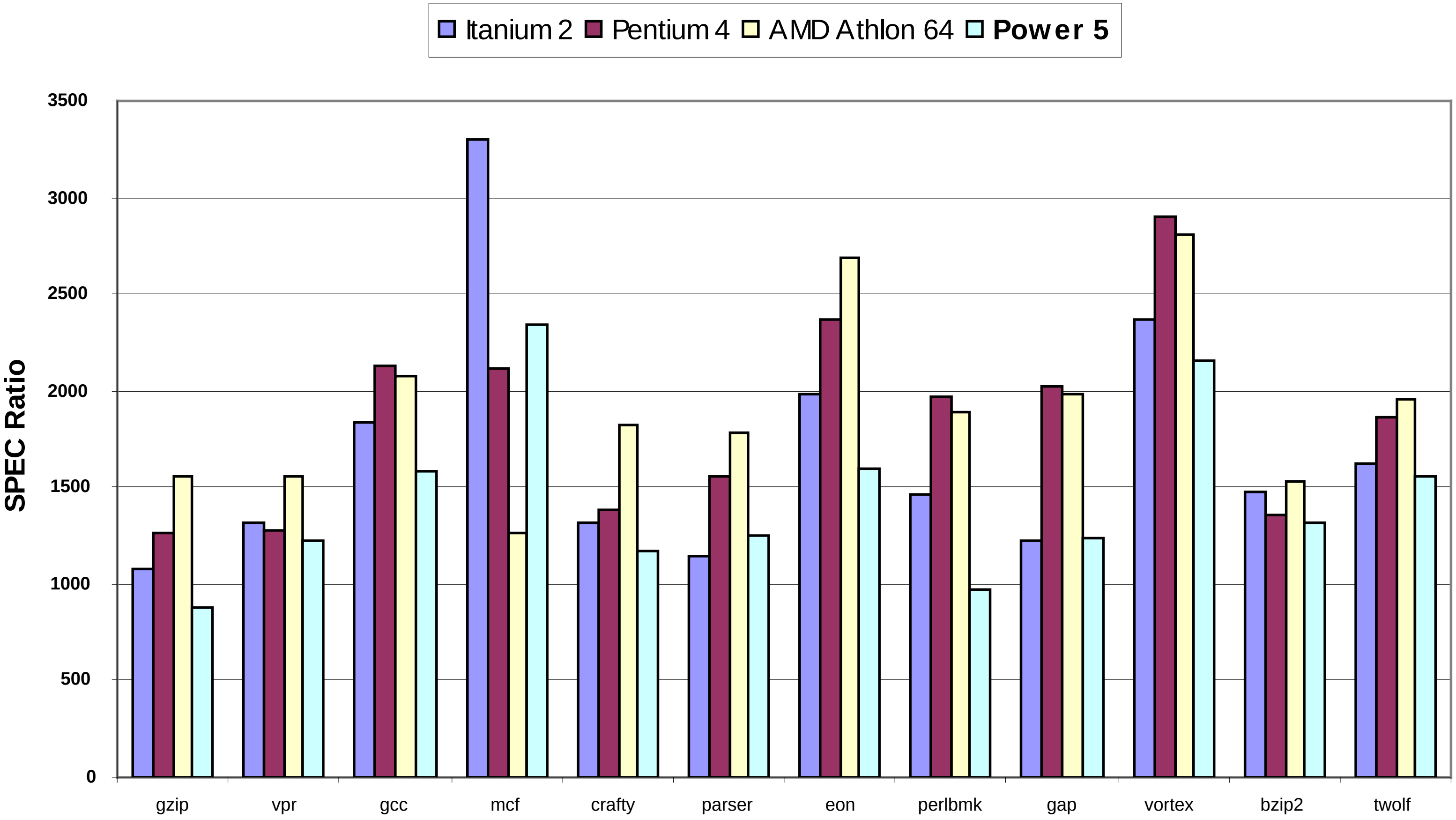
Pentium 4 Hyperthreading (2002)

- First commercial SMT design (two-way SMT).
  - Hyperthreading == SMT
- Logical processors share nearly all resources of the physical processor.
  - Caches, execution units, branch predictors
- Die area overhead of hyperthreading ~ 5 percent.
- When one logical processor is stalled, the other can make progress.
  - No logical processor can use all entries in queues when two threads are active.
- Processor running only one active software thread runs at approximately same speed with or without hyperthreading.
- Hyperthreading dropped on OoO P6 based follow-ons to Pentium 4 (Pentium-M, Core Duo, Core 2 Duo), until revived with Nehalem-generation machines in 2008.
- Intel Atom (in-order x86 core) has two-way vertical multithreading.

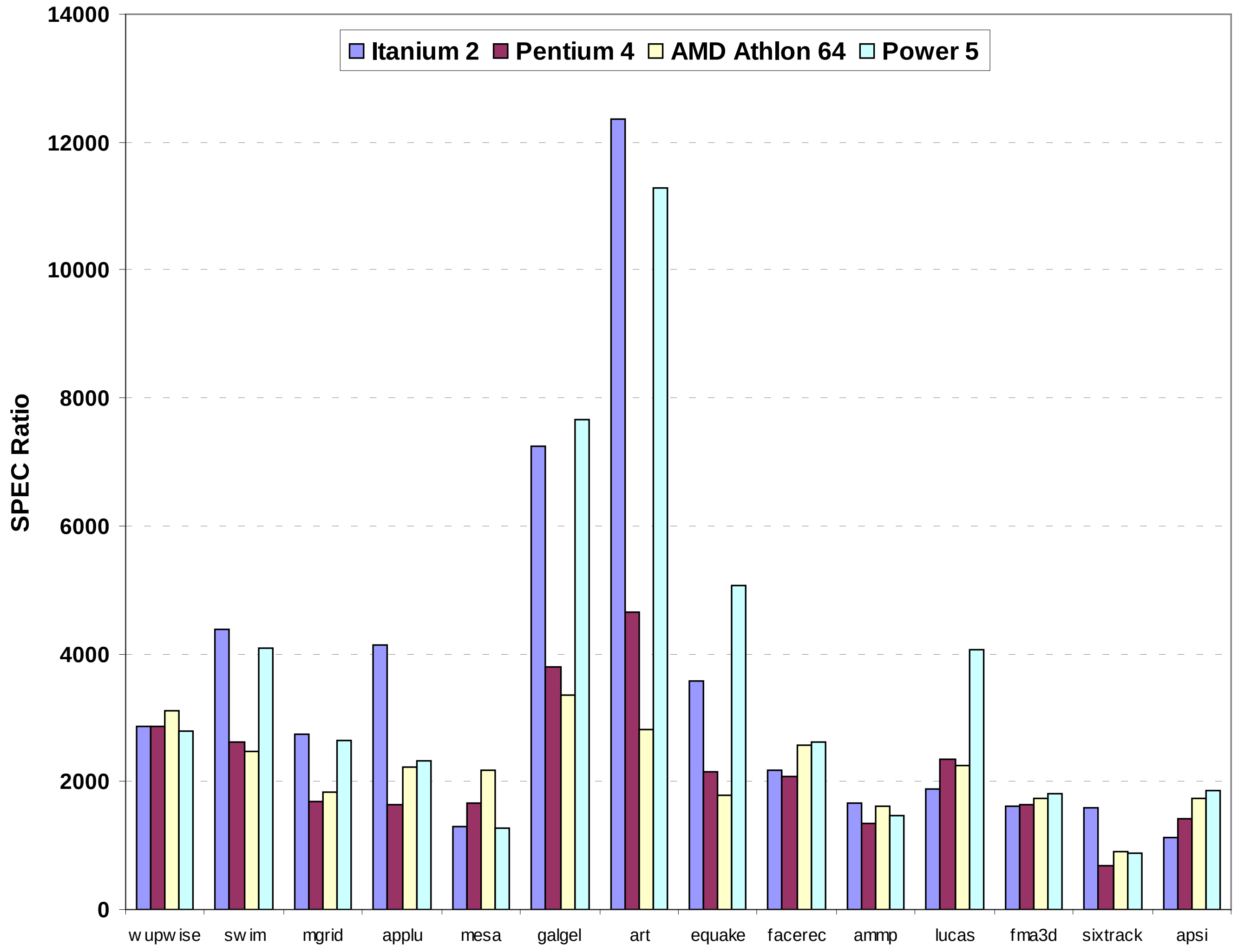
Head-to-Head ILP Competition

Processor	Microarchitecture	Fetch/Issue/ Execute	Functional Units	Clock Rate (GHz)	Transis-tors Die Size	Power
Intel Pentium 4 Extreme	Speculative dynamically scheduled; deeply pipelined; SMT	3/3/4	7 int. 1 FP	3.8	125 M 122 mm^2	115 W
AMD Athlon 64 FX-57	Speculative dynamically scheduled	3/3/4	6 int. 3 FP	2.8	114 M 115 mm^2	104 W
IBM Power5 (one CPU only)	Speculative dynamically scheduled; SMT; two CPU cores/chip	8/4/8	6 int. 2 FP	1.9	200 M 300 mm^2 (est.)	80 W (est.)
Intel Itanium 2	Statically scheduled VLIW-style	6/5/11	9 int. 2 FP	1.6	592 M 423 mm^2	130 W

Performance on SPECint2000

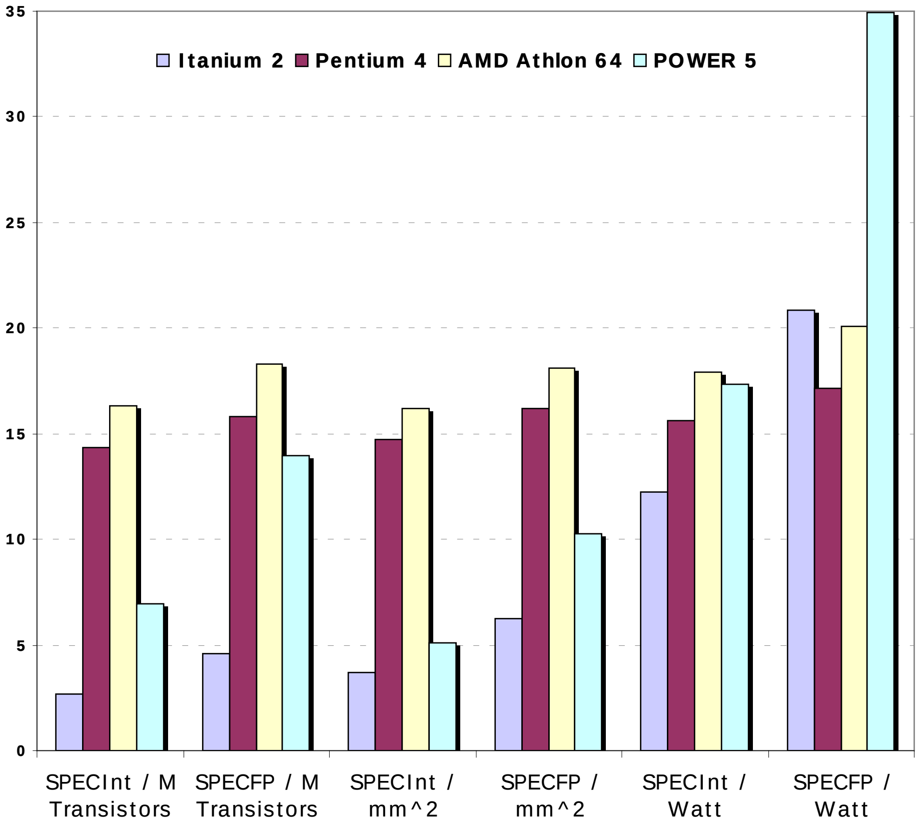


Performance on SPECfp2000



ENGINEERING@SYRACUSE

Normalized Performance: Efficiency



Rank	Itanium 2	P4	Athlon	Power5
Int/Trans	4	2	1	3
FP/Trans	4	2	1	3
Int/area	4	2	1	3
FP/area	4	2	1	3
Int/Watt	4	3	1	2
FP/Watt	2	4	3	1

No Clear Winner

- No obvious over-all leader in performance.
- The AMD Athlon leads on SPECint performance followed by the Pentium 4, Itanium 2, and Power 5.
- Itanium 2 and Power5, which perform similarly on SPECfp, clearly dominate the Athlon and Pentium 4 on SPECfp.
- Itanium 2 is the most inefficient processor both for floating point and integer code for all but one efficiency measure (SPECfp/Watt).
- Athlon and Pentium 4 both make good use of transistors and area in terms of efficiency.
- IBM Power 5 is the most effective user of energy on SPECfp and essentially tied on SPECint.

Limits to ILP

- Doubling issue rates above today's three to six instructions per clock, say to six to 12 instructions, probably requires a processor to:
  - Issue three or four data memory accesses per cycle
  - Resolve two or three branches per cycle
  - Rename and access more than 20 registers per cycle
  - Fetch 12 to 24 instructions per cycle
- The complexities of implementing these capabilities is likely to mean sacrifices in the maximum clock rate.
  - E.g, widest issue processor is the Itanium 2, but it also has the slowest clock rate, despite the fact that it consumes the most power!

Limits to ILP (cont.)

- Most techniques for increasing performance increase power consumption.
- The key question is whether a technique is energy efficient: Does it increase performance faster than it increases power consumption?
- Multiple issue processor techniques all are **energy inefficient**.
  - Issuing multiple instructions incurs some overhead in logic that grows faster than the issue rate grows.
  - Growing gap between peak issue rates and sustained performance.
- Number of transistors switching = f(peak issue rate), and performance = f(sustained rate), growing gap between peak and sustained performance.

  ➔ Increasing energy per unit of performance

In Conclusion

- Itanium architecture does not represent a significant breakthrough in scaling ILP or in avoiding the problems of complexity and power consumption.
- Instead of pursuing more ILP, architects are increasingly focusing on TLP implemented with **single-chip multiprocessors.**
- In 2000, IBM announced the first commercial single-chip, general-purpose multiprocessor, the Power 4, which contained two Power 3 processors and an integrated L2 cache.
  - Since then, Sun Microsystems, AMD, and Intel have switched to a focus on single-chip multiprocessors rather than more aggressive uniprocessors.
- Right balance of ILP and TLP is unclear.
  - Perhaps right choice for server market, which can exploit more TLP, may differ from desktop, where single-thread performance may continue to be a primary requirement.

ENGINEERING@SYRACUSE

Modeling Performance

Parallel Benchmarks

- Linpack: matrix linear algebra
- SPECrate: parallel run of SPEC CPU programs
  - Job-level parallelism
- SPLASH: Stanford Parallel Applications for Shared Memory
  - 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

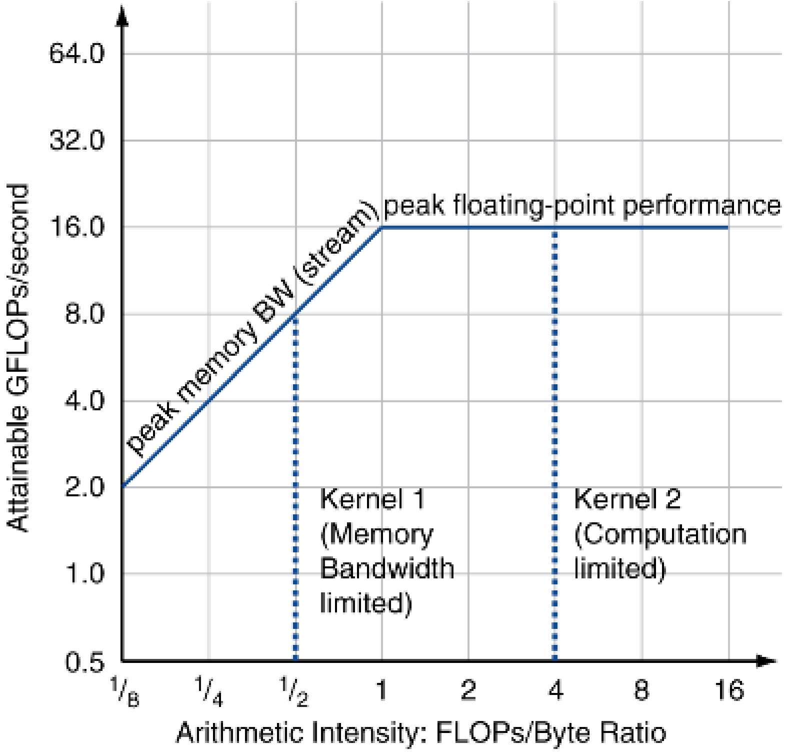
Code or Applications?

- Traditional benchmarks.
  - Fixed code and datasets
- Parallel programming is evolving.
  - Should algorithms, programming languages, and tools be part of the system?
  - Compare systems, provided they implement a given application.
  - E.g., Linpack, Berkeley Design Patterns
- Would foster innovation in approaches to parallelism.

Modeling Performance

- Assume performance metric of interest is achievable GFLOPs/sec.
  - Measured using computational kernels from Berkeley Design Patterns
- Arithmetic intensity of a kernel.
  - FLOPs per byte of memory accessed
- For a given computer, determine:
  - Peak GFLOPs (from data sheet)
  - Peak memory bytes/second (using stream benchmark)

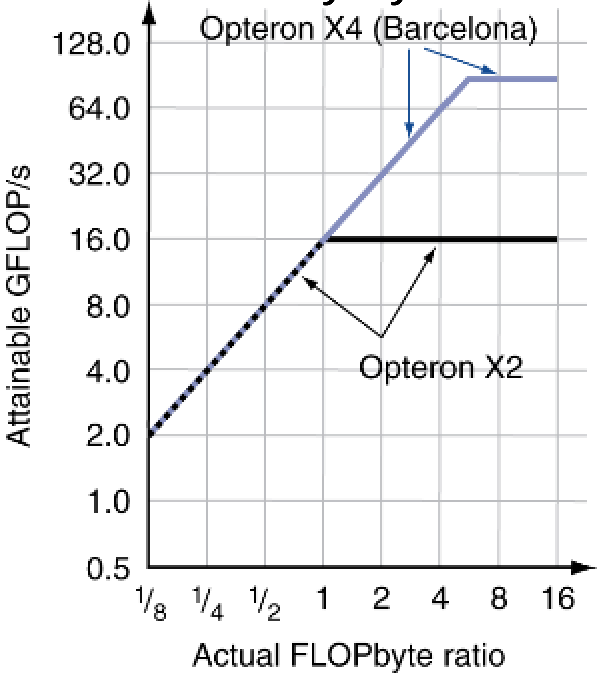
Roofline Diagram



Attainable GPLOPs/sec
= min (peak memory BW × arithmetic intensity, peak FP performance)

Comparing Systems

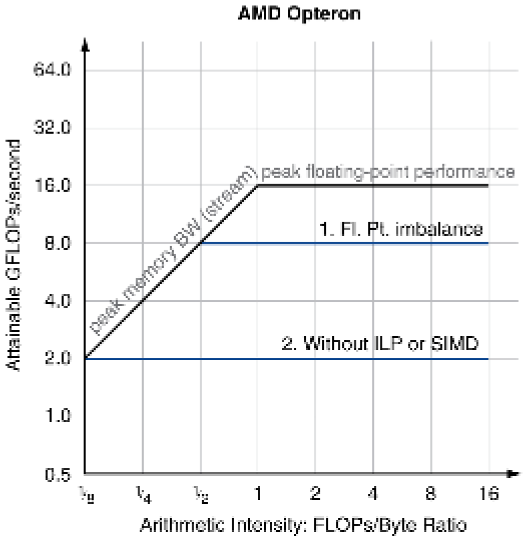
- Example: Opteron X2 vs. Opteron X4
  - Two-core vs. four-core, 2 × FP performance/core, 2.2GHz vs. 2.3GHz
  - Same memory system

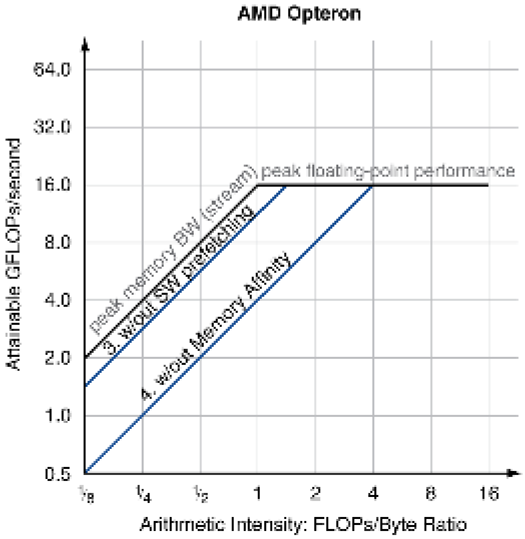


- To get higher performance on X4 than X2
  - Need high arithmetic intensity
  - Or working set must fit in X4's 2 MB L-3 cache

Optimizing Performance

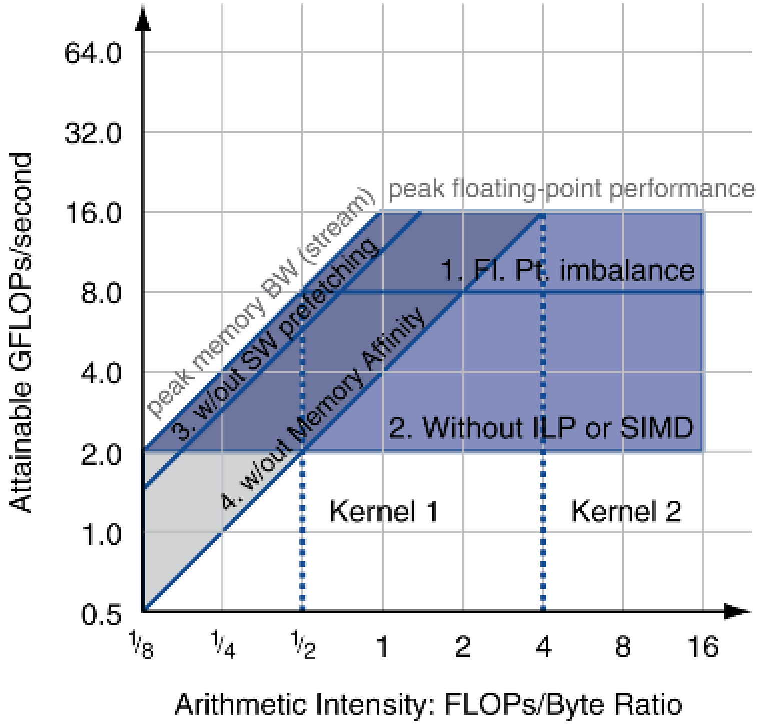
- Optimize FP performance.
  - Balance adds and multiplies.
  - Improve superscalar ILP, and use of SIMD instructions.
- Optimize memory usage.
  - Software prefetch
    - Avoid load stalls.
  - Memory affinity
    - Avoid nonlocal data accesses.


AMD Opteron


AMD Opteron

Optimizing Performance (cont.)

- Choice of optimization depends on arithmetic intensity of cod.e

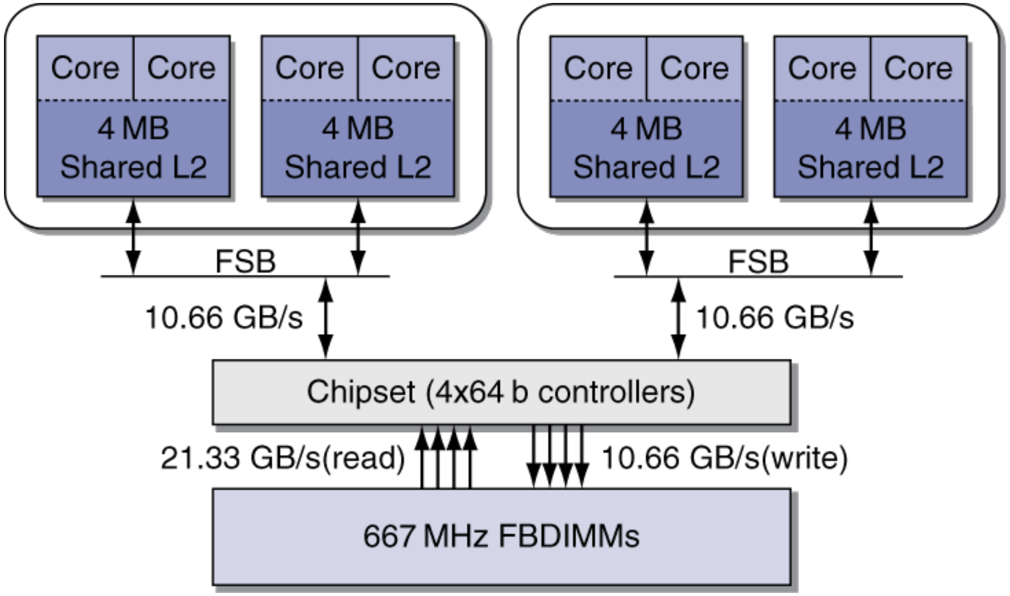


- Arithmetic intensity is not always fixed.
  - May scale with problem size.
  - Caching reduces memory accesses.
    - Increases arithmetic intensity

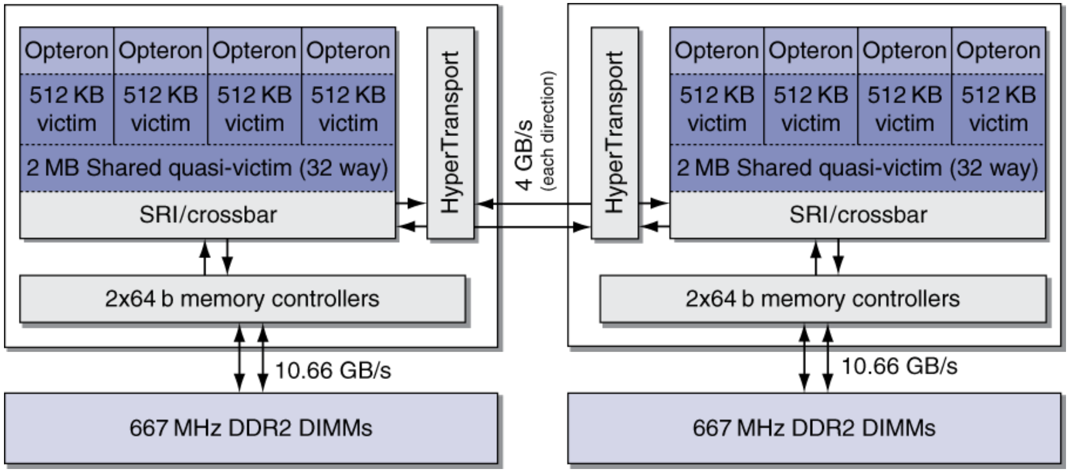
ENGINEERING@SYRACUSE

Comparing Multicore Systems

Four Example Systems

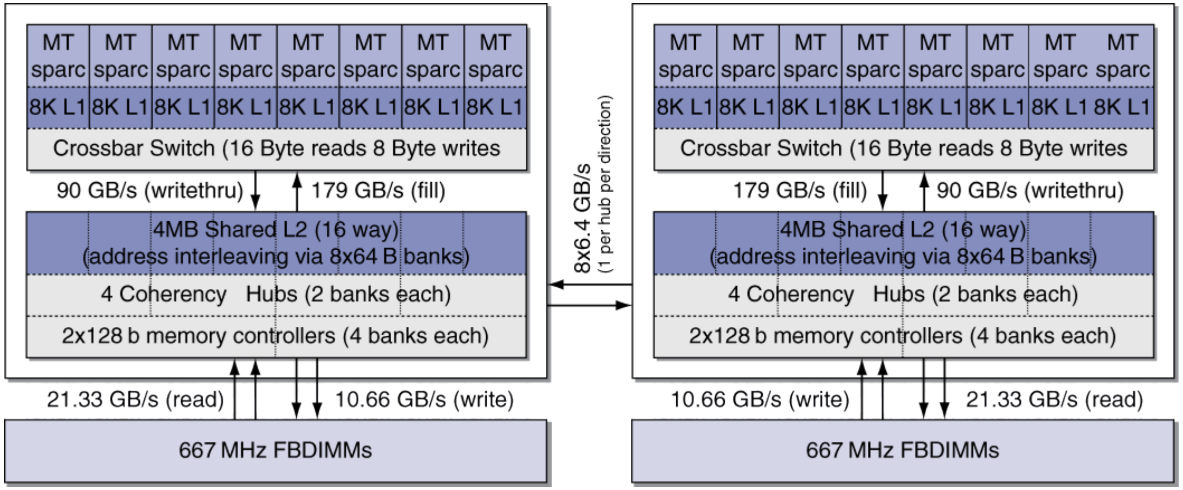


2 × quad-core
Intel Xeon e5345
(Clovertown)



2 × quad-core
AMD OpteronX4 2356
(Barcelona)

Four Example Systems (cont.)



MT sparc | MT sparc | MT sparc | MT sparc | MT sparc | MT sparc | MT sparc | MT sparc

8K L1 | 8K L1 | 8K L1 | 8K L1 | 8K L1 | 8K L1 | 8K L1 | 8K L1

Crossbar Switch (16 Byte reads 8 Byte writes

90 GB/s (writethru)

179 GB/s (fill)

4MB Shared L2 (16 way)
(address interleaving via 8x64 B banks)

4 Coherency Hubs (2 banks each)

2x128 b memory controllers (4 banks each)

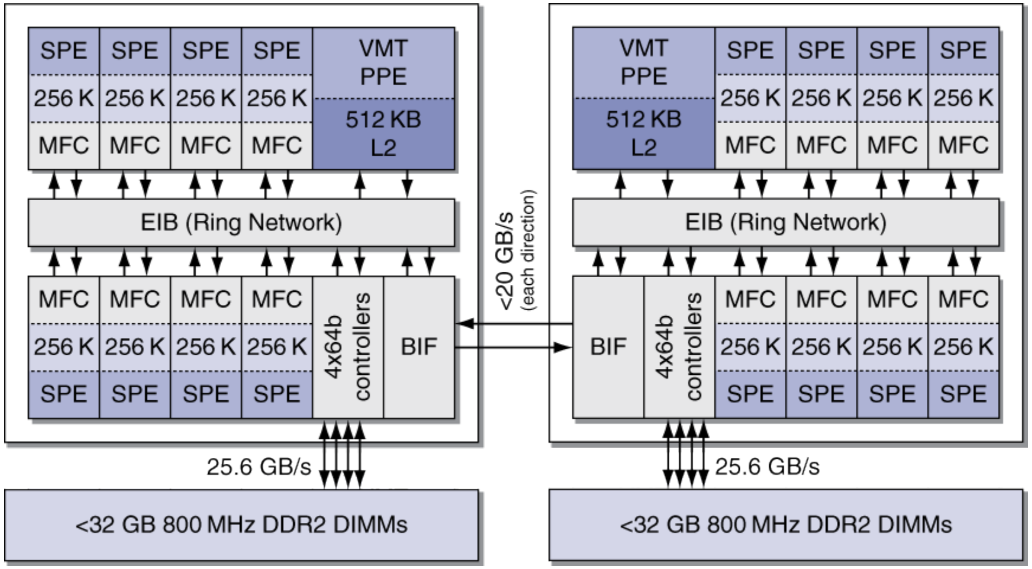
21.33 GB/s (read)

10.66 GB/s (write)

667 MHz FBDIMMs

8x6.4 GB/s
(1 per hub per direction)

2 × oct-core
Sun UltraSPARC
T2 5140 (Niagara 2)



SPE | SPE | SPE | SPE

256 K | 256 K | 256 K | 256 K

MFC | MFC | MFC | MFC

VMT PPE

512 KB L2

EIB (Ring Network)

MFC | MFC | MFC | MFC

256 K | 256 K | 256 K | 256 K

SPE | SPE | SPE | SPE

4x64b controllers

BIF

<20 GB/s
(each direction)

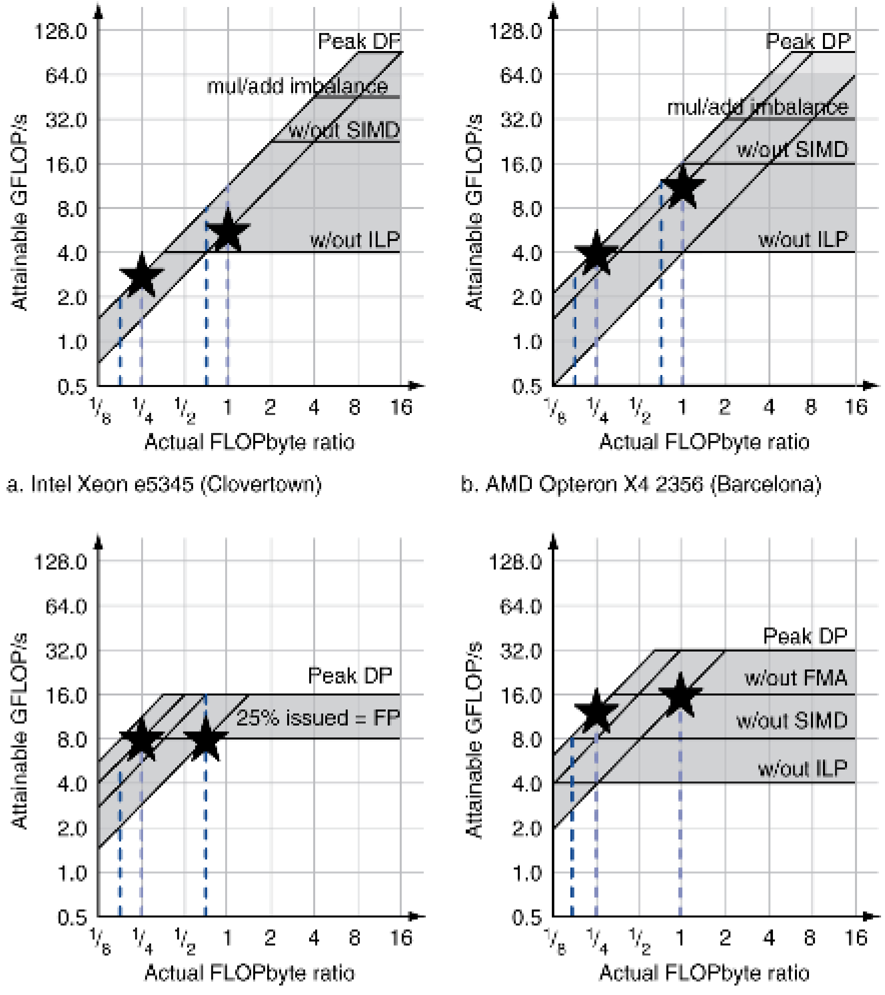
25.6 GB/s

<32 GB 800 MHz DDR2 DIMMs

2 × oct-core
IBM Cell QS20

And Their Rooflines

- Kernels:
  - SpMV (left)
  - LBHMD (right)
- Some optimizations change arithmetic intensity.
- x86 systems have higher peak GFLOPs.
  - But harder to achieve, given memory bandwidth



a. Intel Xeon e5345 (Clovertown)

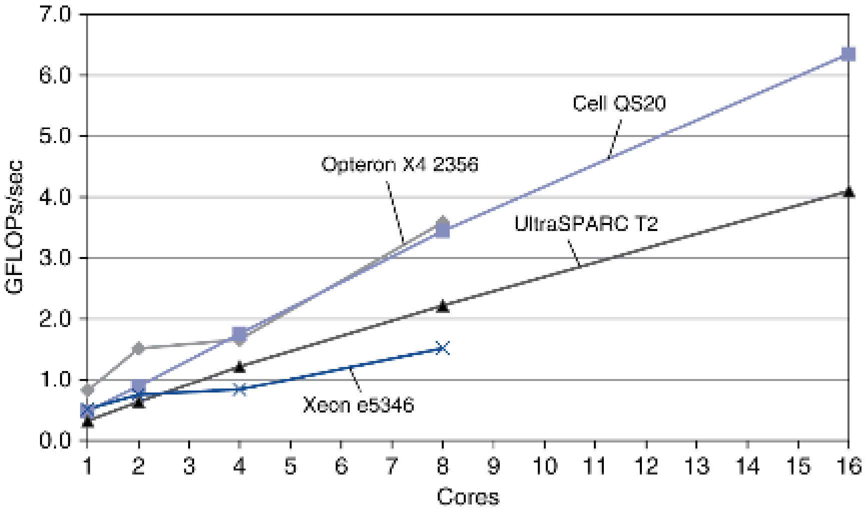
b. AMD Opteron X4 2356 (Barcelona)

c. Sun UltraSPARC T2 5140 (Niagara 2)

d. IBM Cell QS20

Performance on SpMV

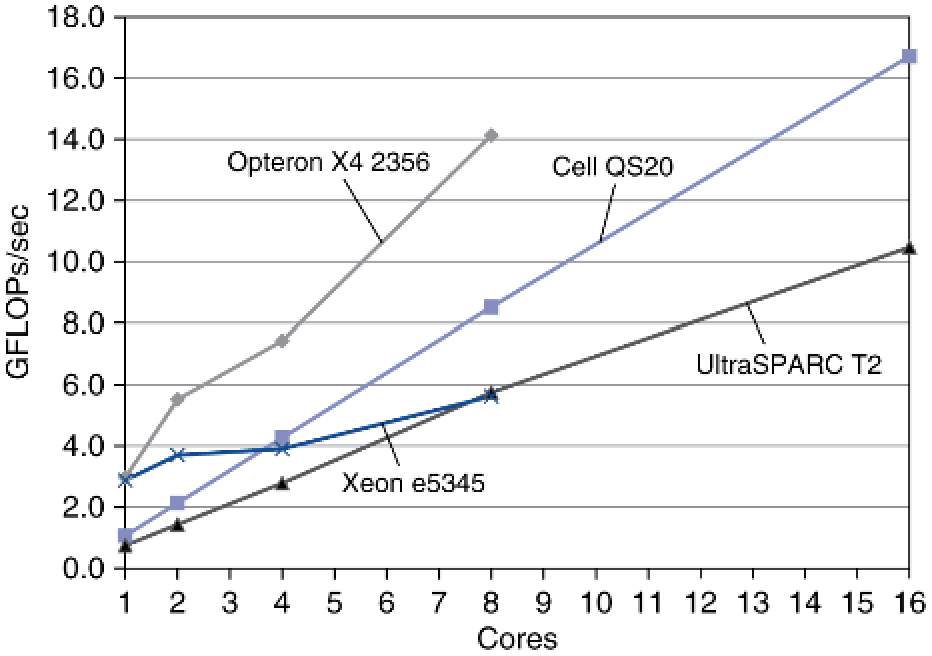
- Sparse matrix/vector multiply
  - Irregular memory accesses, memory bound
- Arithmetic intensity
  - 0.166 before memory optimization, 0.25 after



- Xeon vs. Opteron
  - Similar peak FLOPs.
  - Xeon limited by shared FSBs and chipset.
- UltraSPARC/Cell vs. x86
  - 20–30 vs. 75 peak GFLOPs
  - More cores and memory bandwidth

Performance on LBMHD

- Fluid dynamics: structured grid over time steps
  - Each point: 75 FP read/write, 1300 FP ops
- Arithmetic intensity
  - 0.70 before optimization, 1.07 after



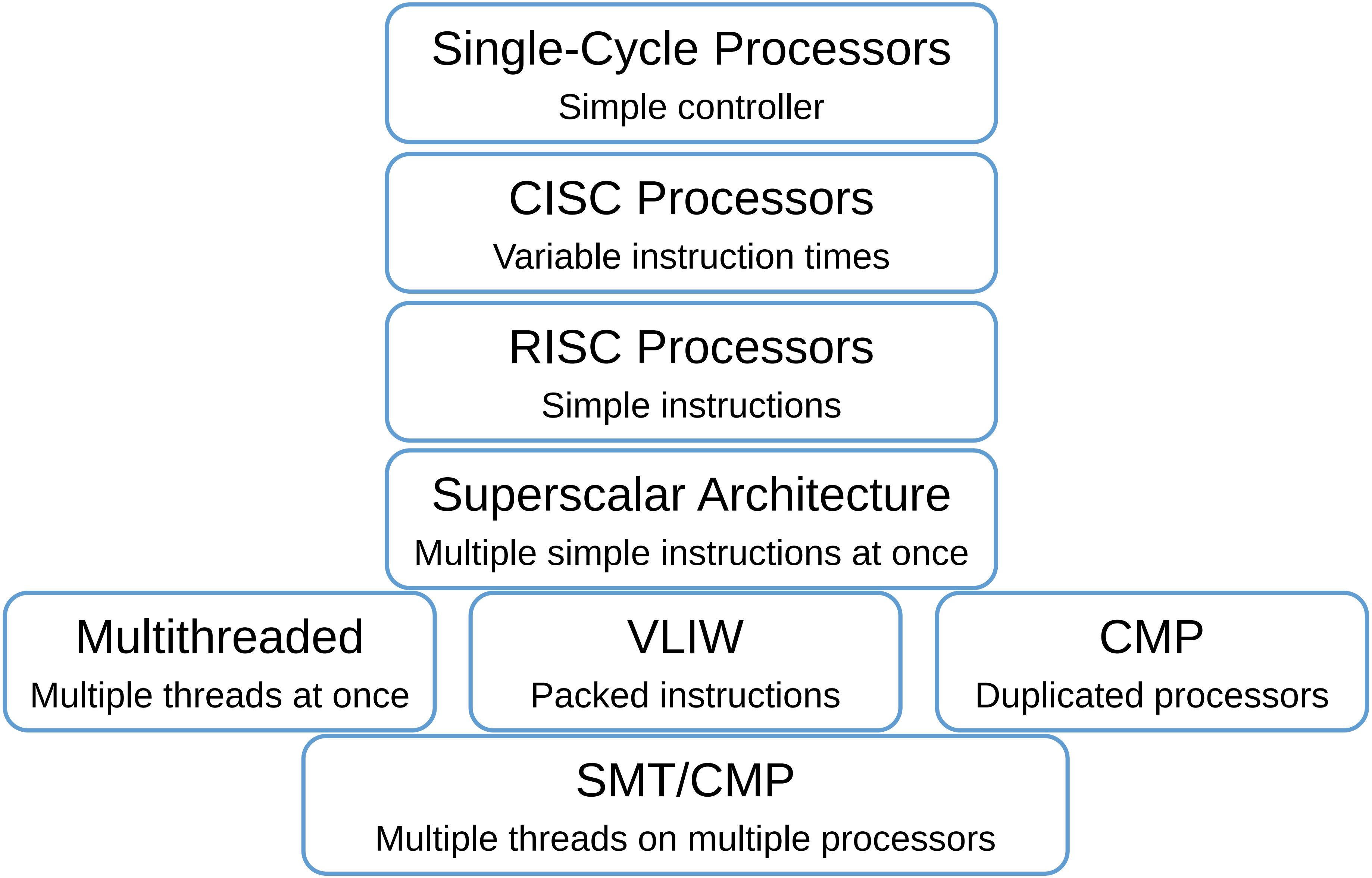
- Opteron vs. UltraSPARC
  - More powerful cores, not limited by memory bandwidth
- Xeon vs. others
  - Still suffers from memory bottlenecks

Achieving Performance

- Compare naïve vs. optimized code.
  - If naïve code performs well, it's easier to write high-performance code for the system.

System	Kernel	Naïve GFLOPs/sec	Optimized GFLOPs/sec	Naïve as % of Optimized
Intel Xeon	SpMV	1.0	1.5	64%
	LBMHD	4.6	5.6	82%
AMD Opteron X4	SpMV	1.4	3.6	38%
	LBMHD	7.1	14.1	50%
Sun UltraSPARC T2	SpMV	3.5	4.1	86%
	LBMHD	9.7	10.5	93%
IBM Cell QS20	SpMV	Naïve code not feasible	6.4	0%
	LBMHD		16.7	0%

Microprocessor Trends



Single-Cycle Processors

Simple controller

CISC Processors

Variable instruction times

RISC Processors

Simple instructions

Superscalar Architecture

Multiple simple instructions at once

Multithreaded

Multiple threads at once

VLIW

Packed instructions

CMP

Duplicated processors

SMT/CMP

Multiple threads on multiple processors

Concluding Remarks

- Goal: higher performance by using multiple processors
- Difficulties
  - Developing parallel software
  - Devising appropriate architectures
- Many reasons for optimism
  - Changing software and application environment
  - Chip-level multiprocessors with lower latency, higher bandwidth interconnect
- An ongoing challenge for computer architects!

ENGINEERING@SYRACUSE

Conclusions

In Conclusion

- Multithreading
- Multithreaded systems
- Types of multithreading
- Simultaneous multithreading
- Design challenges
- SMT performance
- Modeling performance
- Comparing multicore systems

ENGINEERING@SYRACUSE