18-742 Fall 2012
Parallel Computer Architecture
Lecture 5: Multi-Core Processors II

Prof. Onur Mutlu
Carnegie Mellon University
9/17/2012

New Review Assignments

- Due: Friday, September 21, 11:59pm.
- Smith, "Architecture and applications of the HEP multiprocessor computer system," SPIE 1981.
- Tullsen et al., "Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor," ISCA 1996.
- Chappell et al., "Simultaneous Subordinate Microthreading (SSMT)," ISCA 1999.
- Reinhardt and Mukherjee, "Transient Fault Detection via Simultaneous Multithreading," ISCA 2000.

Last Lecture: 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?

Today

- An Early History of Multi-Core
- Homogeneous Multi-Core Evolution
- From Symmetry to Asymmetry

Multi-Core Evolution (An Early History)

Piranha Chip Multiprocessor

- Barroso et al., "Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing," ISCA 2000.

- An early example of a symmetric multi-core processor
- Large-scale server based on CMP nodes
- Designed for commercial workloads

- Read:
  - Barroso et al., "Memory System Characterization of Commercial Workloads," ISCA 1998.
  - Ranganathan et al., "Performance of Database Workloads on Shared-Memory Systems with Out-of-Order Processors," ASPLOS 1998.

Commercial Workload Characteristics

- Memory system is the main bottleneck
  - Very high CPI
  - Execution time dominated by memory stall times
  - Instruction stalls as important as data stalls
  - Fast/large L2 caches are critical
- Very poor Instruction Level Parallelism (ILP) with existing techniques
  - Frequent hard-to-predict branches
  - Large L1 miss ratios
  - Small gains from wide-issue out-of-order techniques
- No need for floating point and multimedia units

Piranha Processing Node

CPU

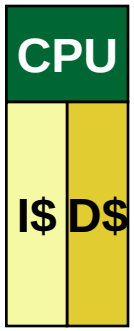
Alpha core:
1-issue, in-order,
500MHz

Next few slides from

Luiz Barroso's ISCA 2000 presentation of

Piranha: A Scalable Architecture
Based on Single-Chip Multiprocessing

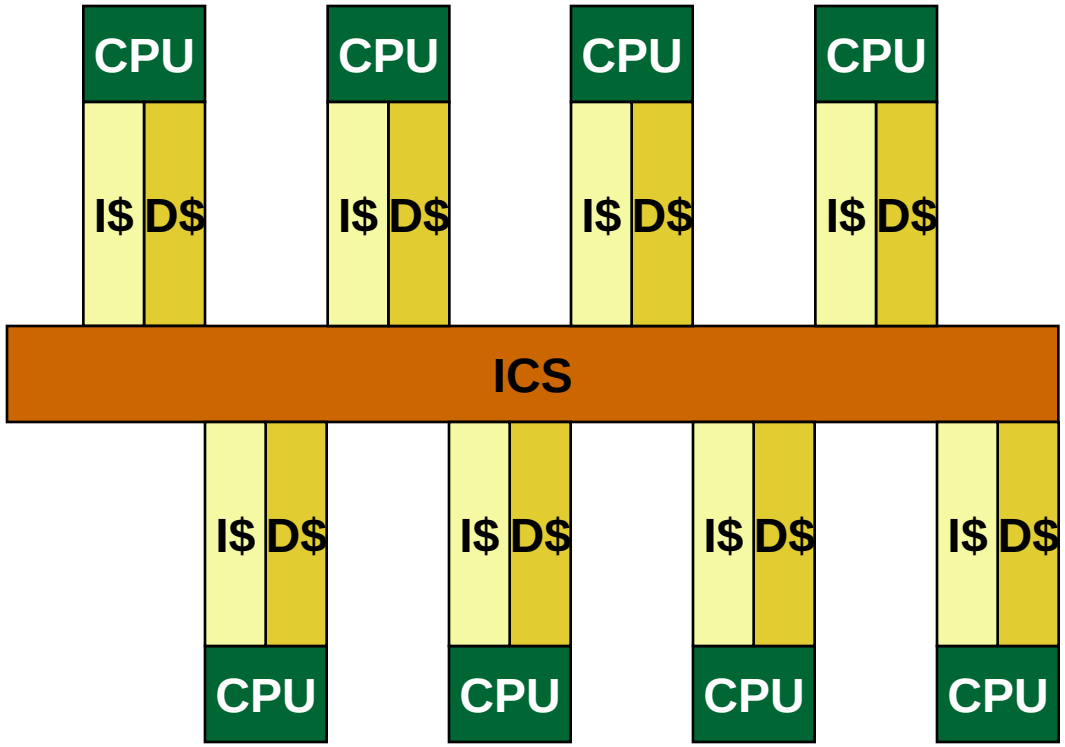
Piranha Processing Node



Alpha core:
- 1-issue, in-order, 500MHz

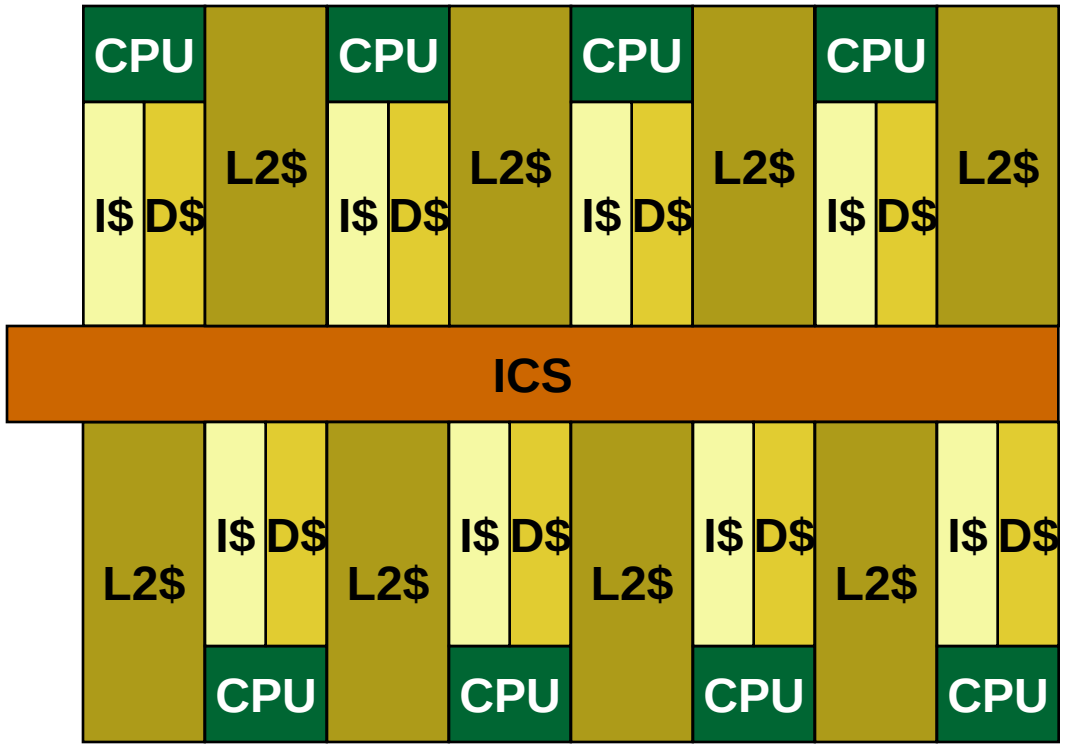
L1 caches:
- I&D, 64KB, 2-way

Piranha Processing Node



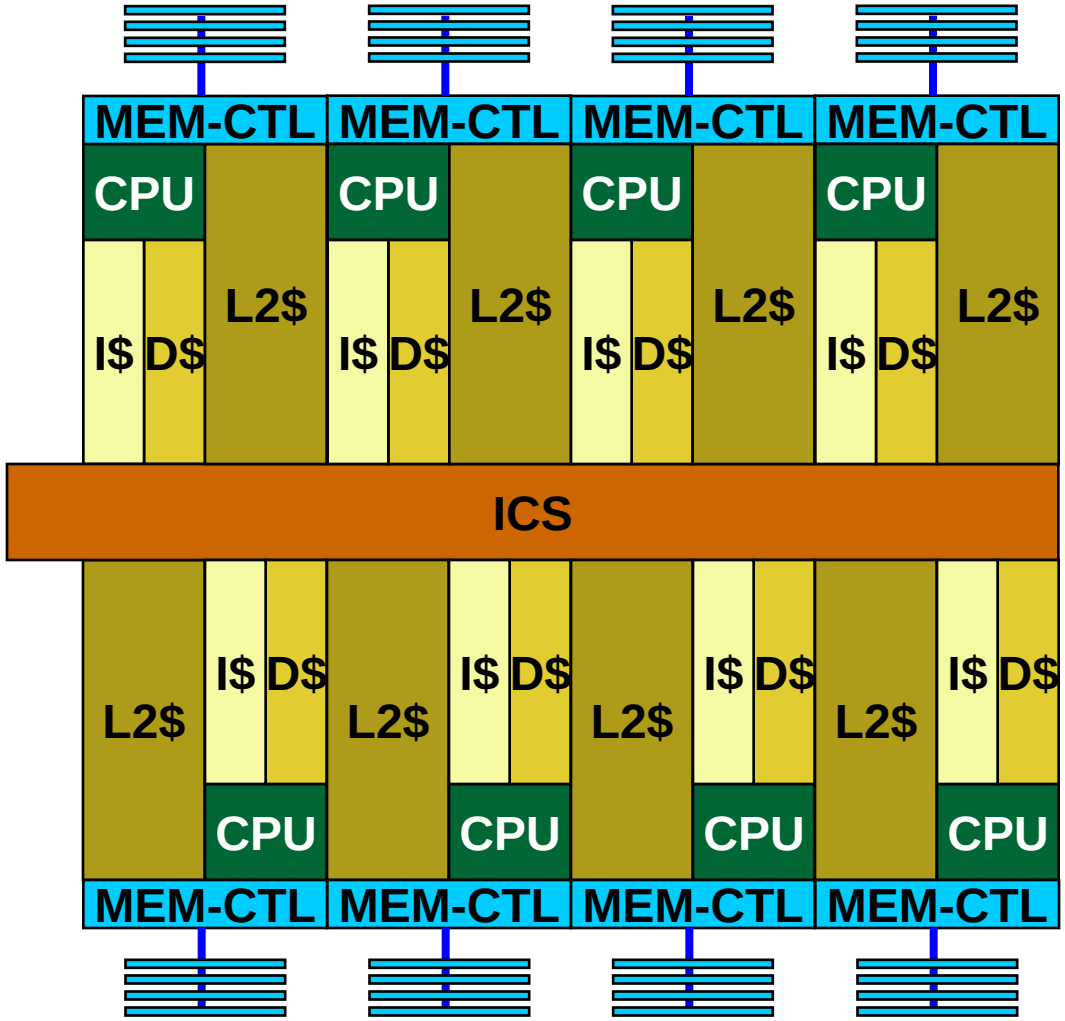
Alpha core:
1-issue, in-order, 500MHz
L1 caches:
I&D, 64KB, 2-way
Intra-chip switch (ICS)
32GB/sec, 1-cycle delay

Piranha Processing Node



Alpha core:
1-issue, in-order, 500MHz
L1 caches:
I&D, 64KB, 2-way
Intra-chip switch (ICS)
32GB/sec, 1-cycle delay
L2 cache:
shared, 1MB, 8-way

Piranha Processing Node



Alpha core:
- 1-issue, in-order, 500MHz

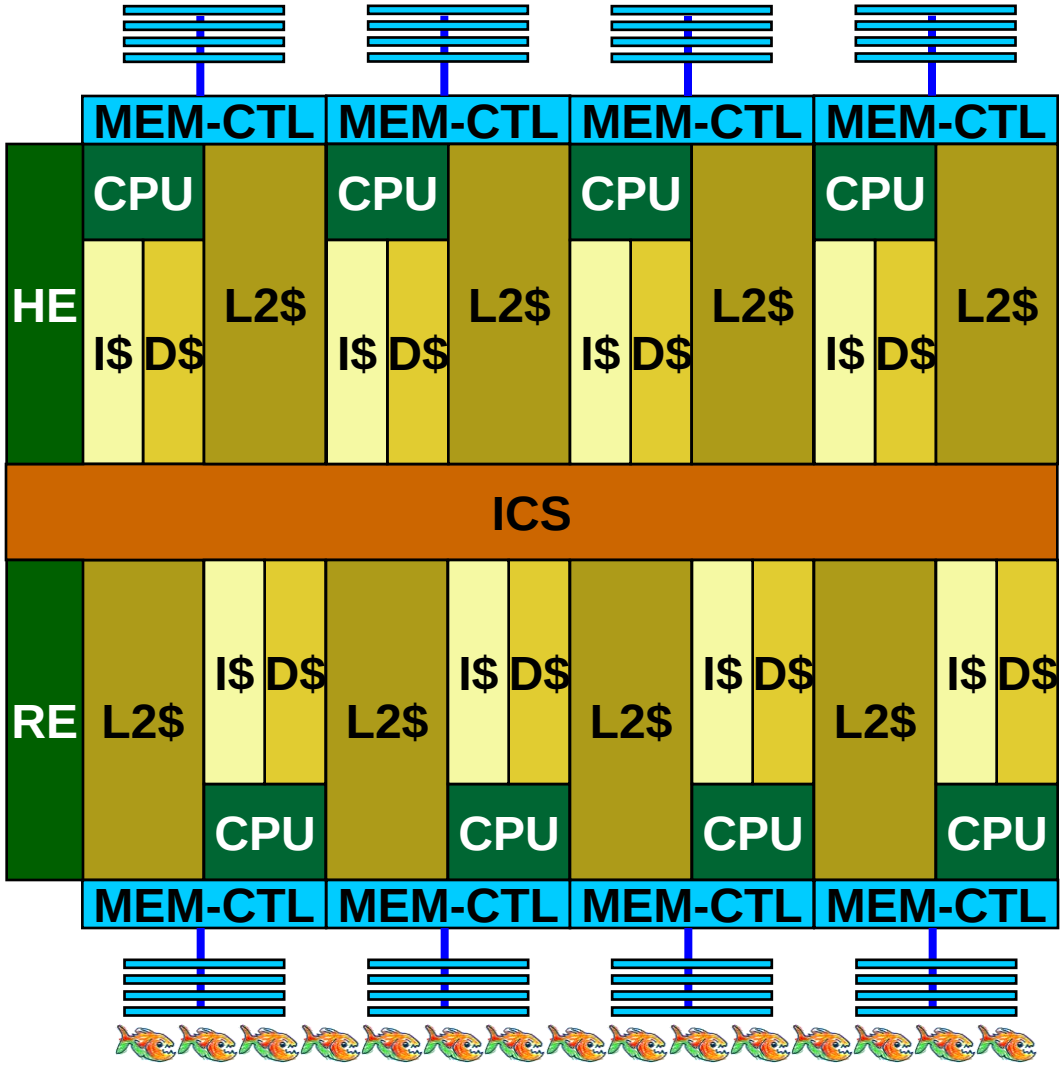
L1 caches:
- I&D, 64KB, 2-way

Intra-chip switch (ICS)
- 32GB/sec, 1-cycle delay

L2 cache:
- shared, 1MB, 8-way

Memory Controller (MC)
- RDRAM, 12.8GB/sec

Piranha Processing Node



Alpha core:
- 1-issue, in-order, 500MHz

L1 caches:
- I&D, 64KB, 2-way

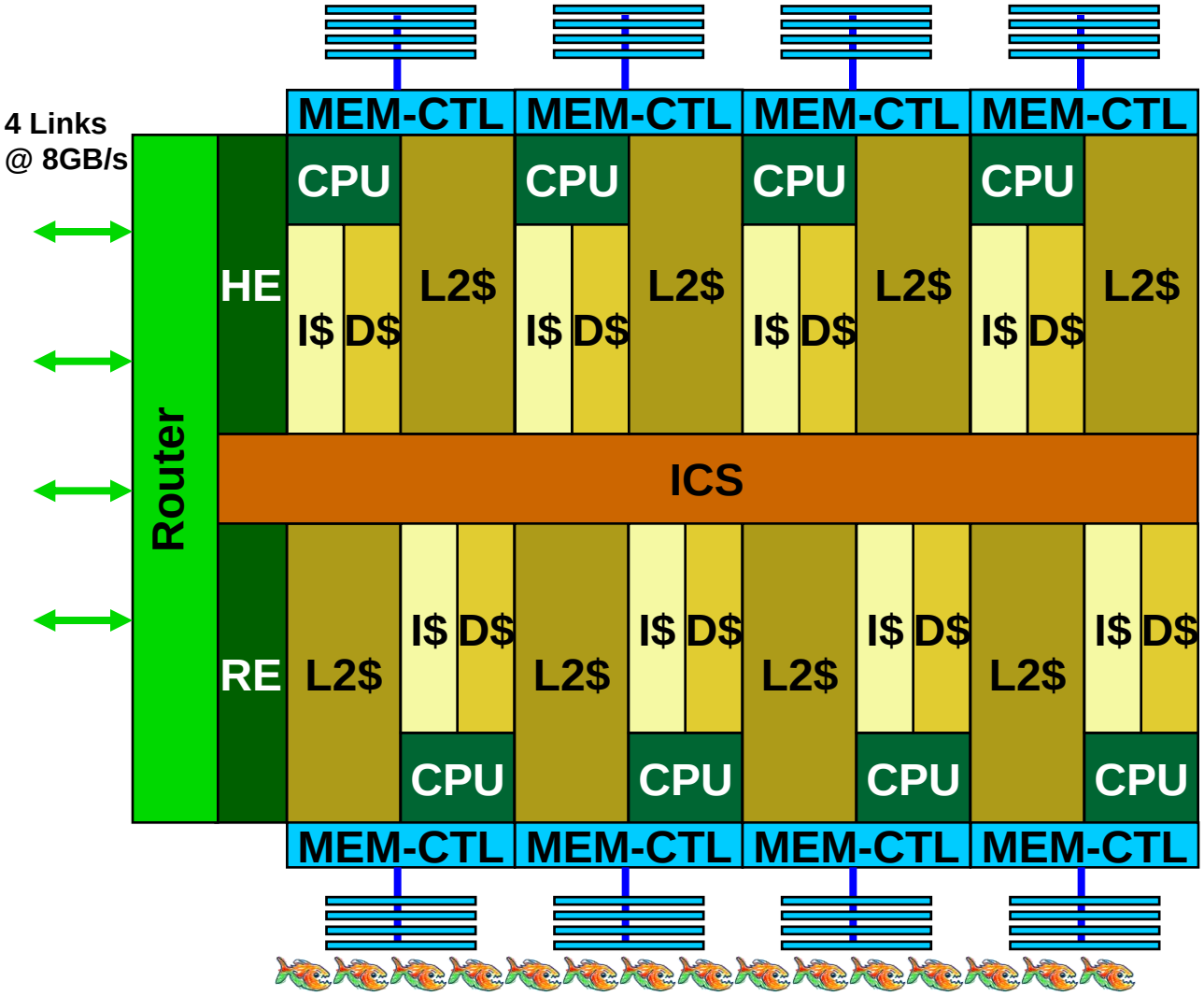
Intra-chip switch (ICS)
- 32GB/sec, 1-cycle delay

L2 cache:
- shared, 1MB, 8-way

Memory Controller (MC)
- RDRAM, 12.8GB/sec

Protocol Engines (HE & RE)
- μprog., 1K μinstr., even/odd interleaving

Piranha Processing Node



Alpha core:
- 1-issue, in-order, 500MHz

L1 caches:
- I&D, 64KB, 2-way

Intra-chip switch (ICS)
- 32GB/sec, 1-cycle delay

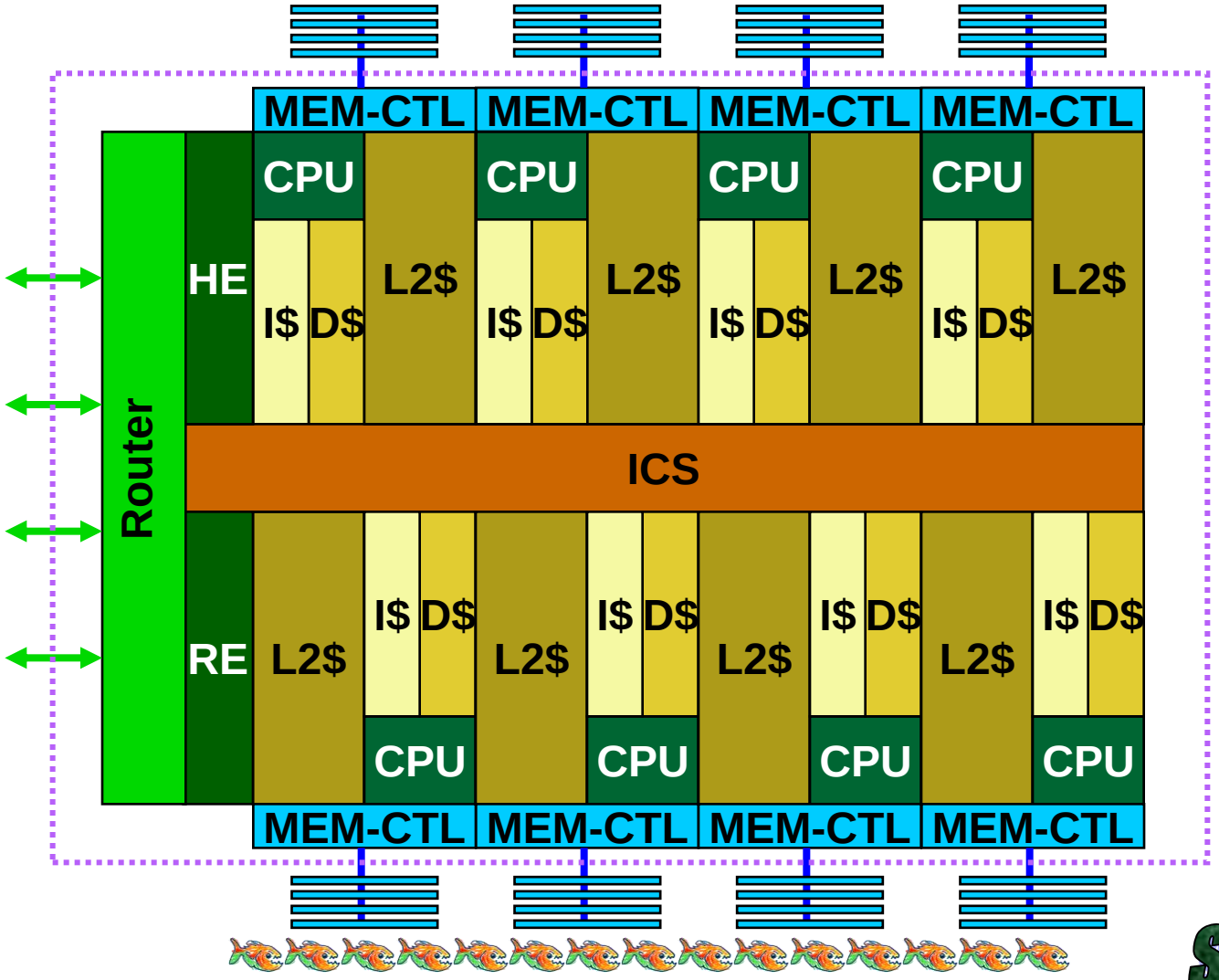
L2 cache:
- shared, 1MB, 8-way

Memory Controller (MC)
- RDRAM, 12.8GB/sec

Protocol Engines (HE & RE):
- µprog., 1K µinstr., even/odd interleaving

System Interconnect:
- 4-port Xbar router
- topology independent
- 32GB/sec total bandwidth

Piranha Processing Node



Alpha core:
- 1-issue, in-order, 500MHz

L1 caches:
- I&D, 64KB, 2-way

Intra-chip switch (ICS)
- 32GB/sec, 1-cycle delay

L2 cache:
- shared, 1MB, 8-way

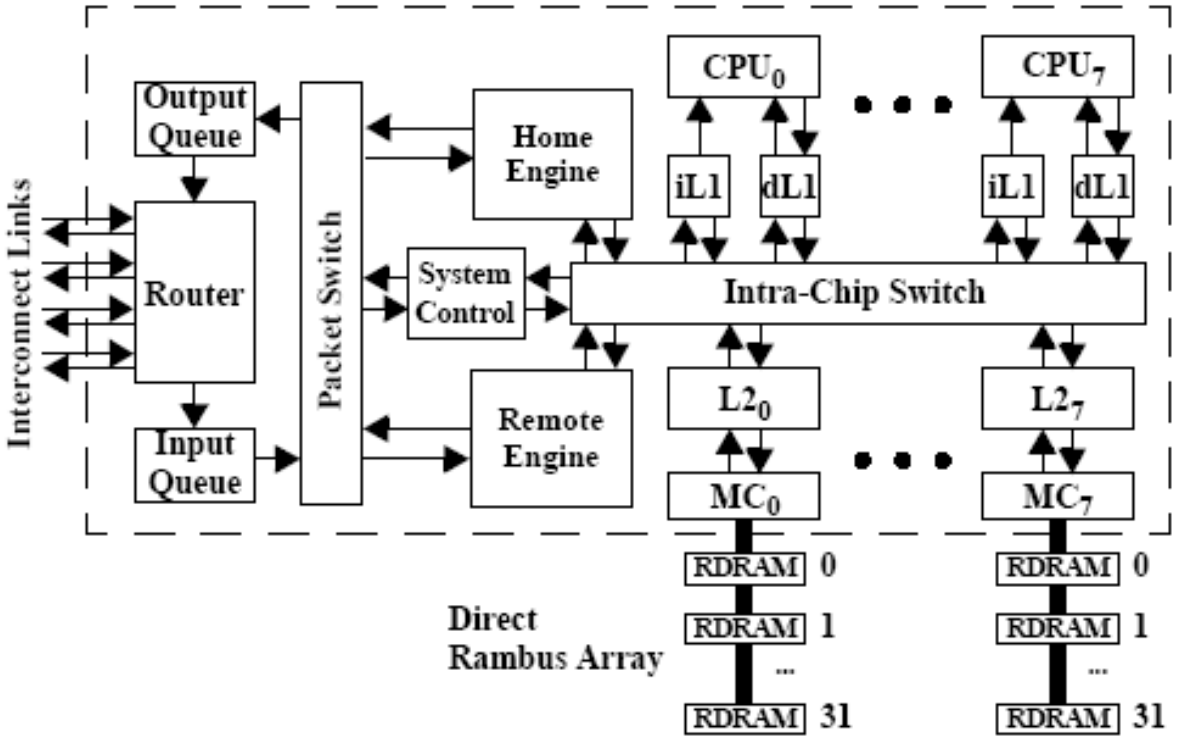
Memory Controller (MC)
- RDRAM, 12.8GB/sec

Protocol Engines (HE & RE):
- µprog., 1K µinstr., even/odd interleaving

System Interconnect:
- 4-port Xbar router
- topology independent
- bandwidth

Single Chip

Piranha Processing Node



Inter-Node Coherence Protocol Engine

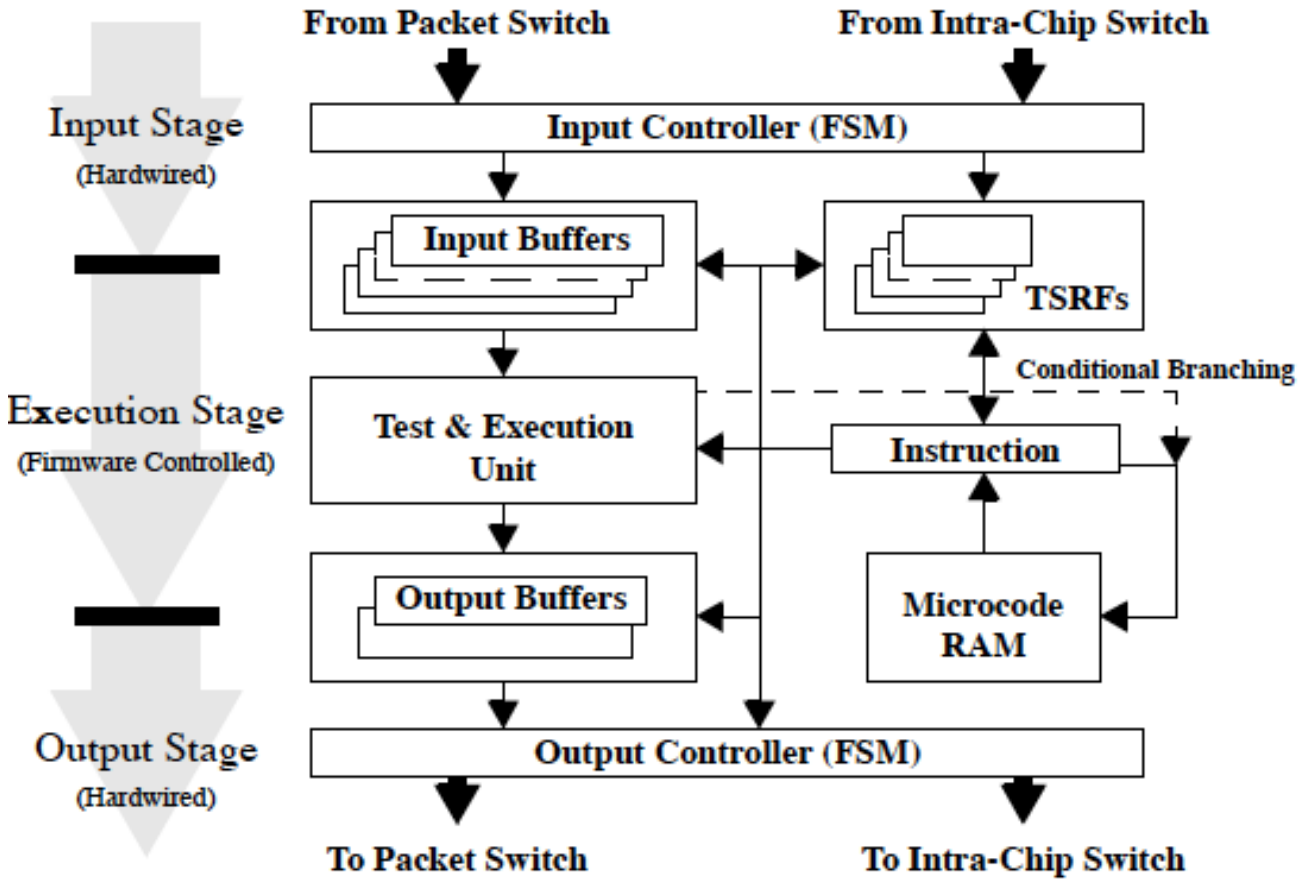


Figure 4. Block diagram of a protocol engine.

Piranha System

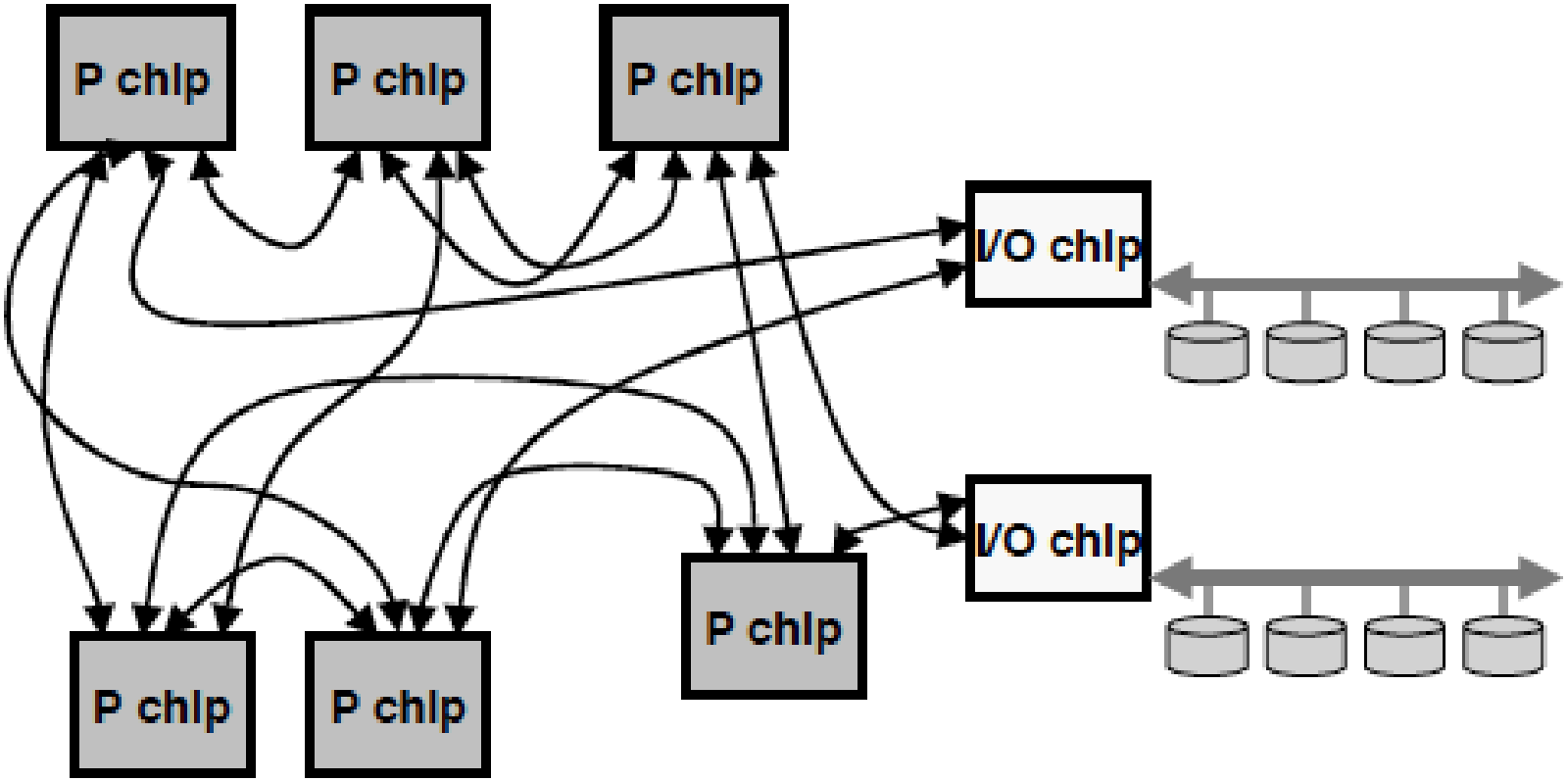


Figure 3. Example configuration for a Piranha system with six processing (8 CPUs each) and two I/O chips.

Piranha I/O Node

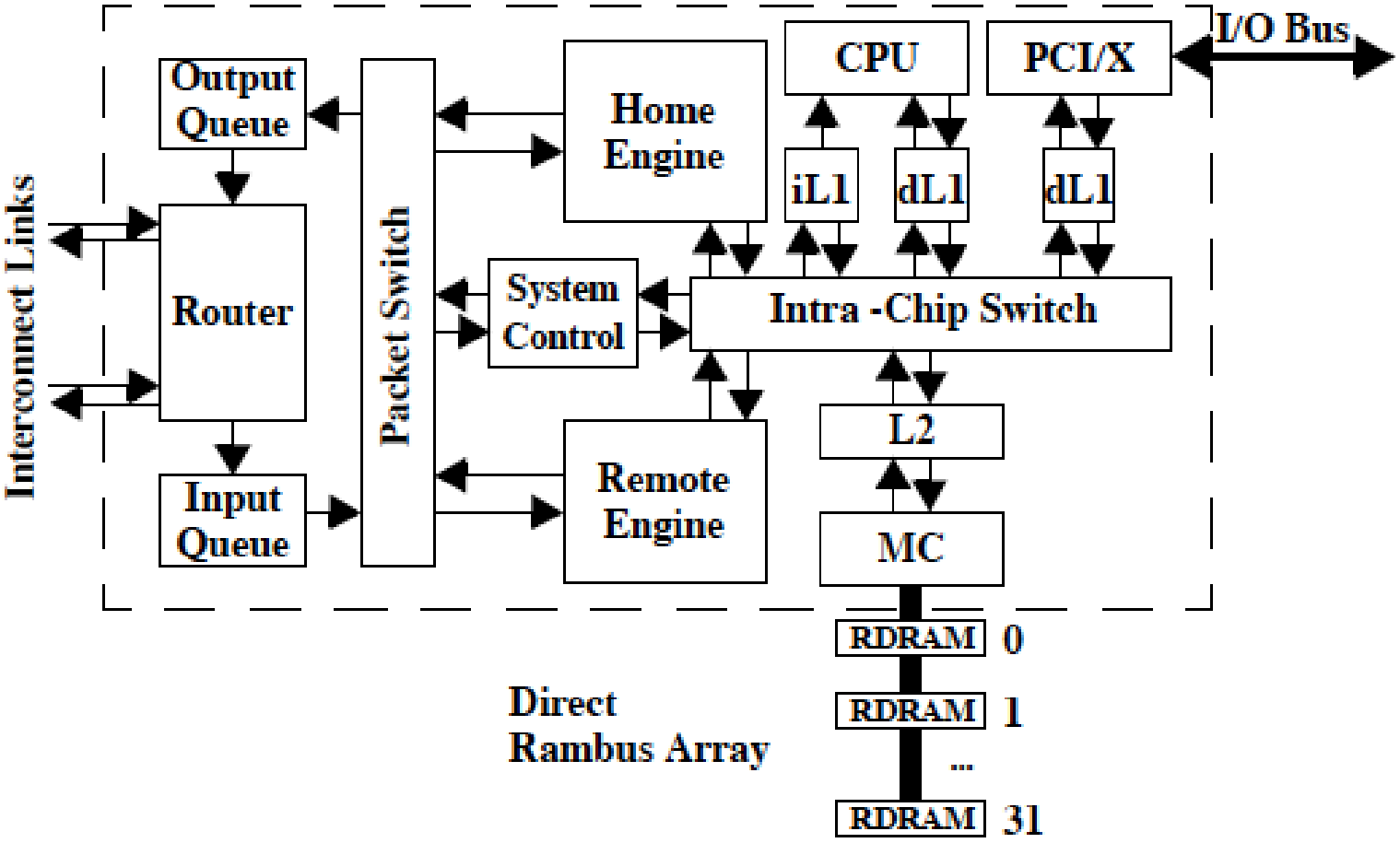
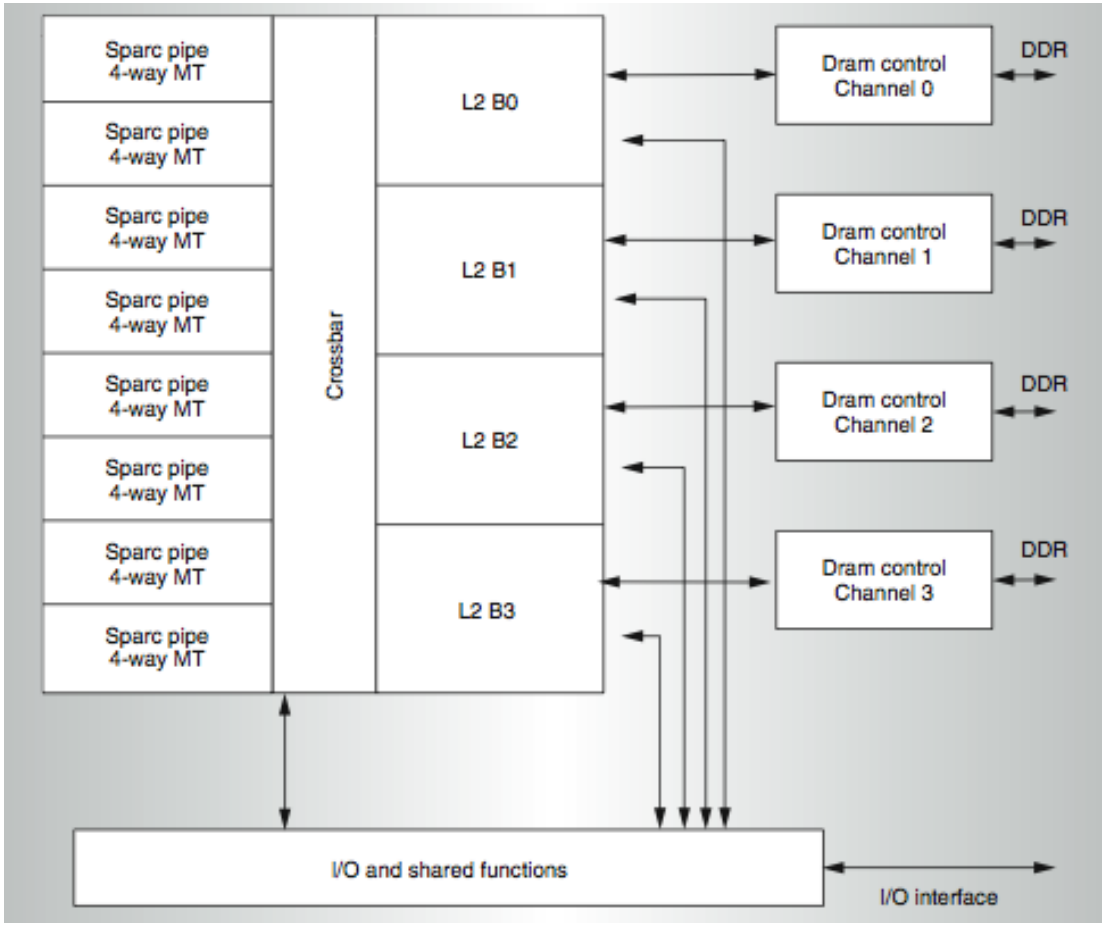


Figure 2. Block diagram of a single-chip Piranha I/O node.

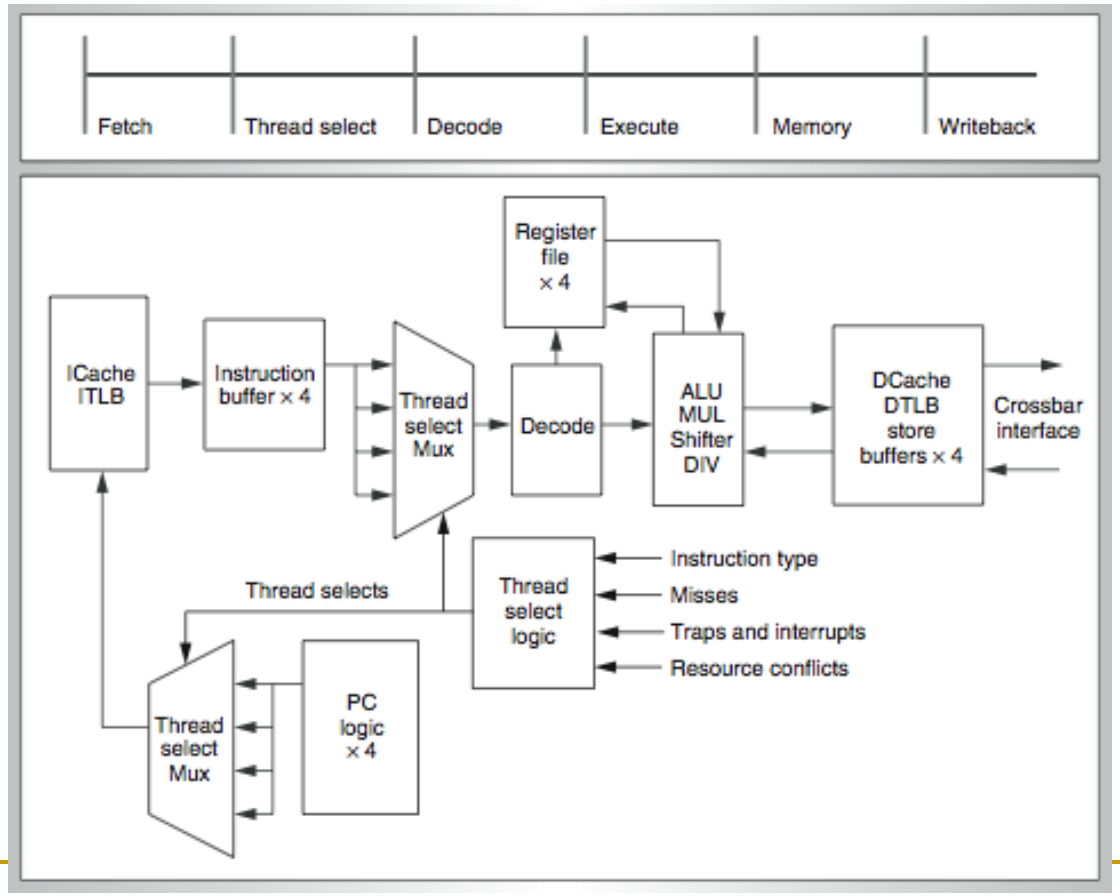
Sun Niagara (UltraSPARC T1)

- Kongetira et al., "Niagara: A 32-Way Multithreaded SPARC Processor," IEEE Micro 2005.



Niagara Core

- 4-way fine-grain multithreaded, 6-stage, dual-issue in-order
- Round robin thread selection (unless cache miss)
- Shared FP unit among cores

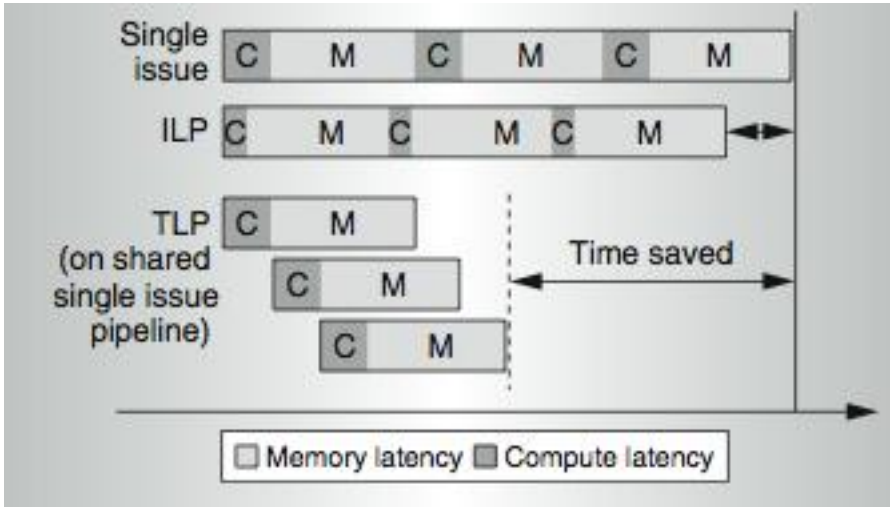


Niagara Design Point

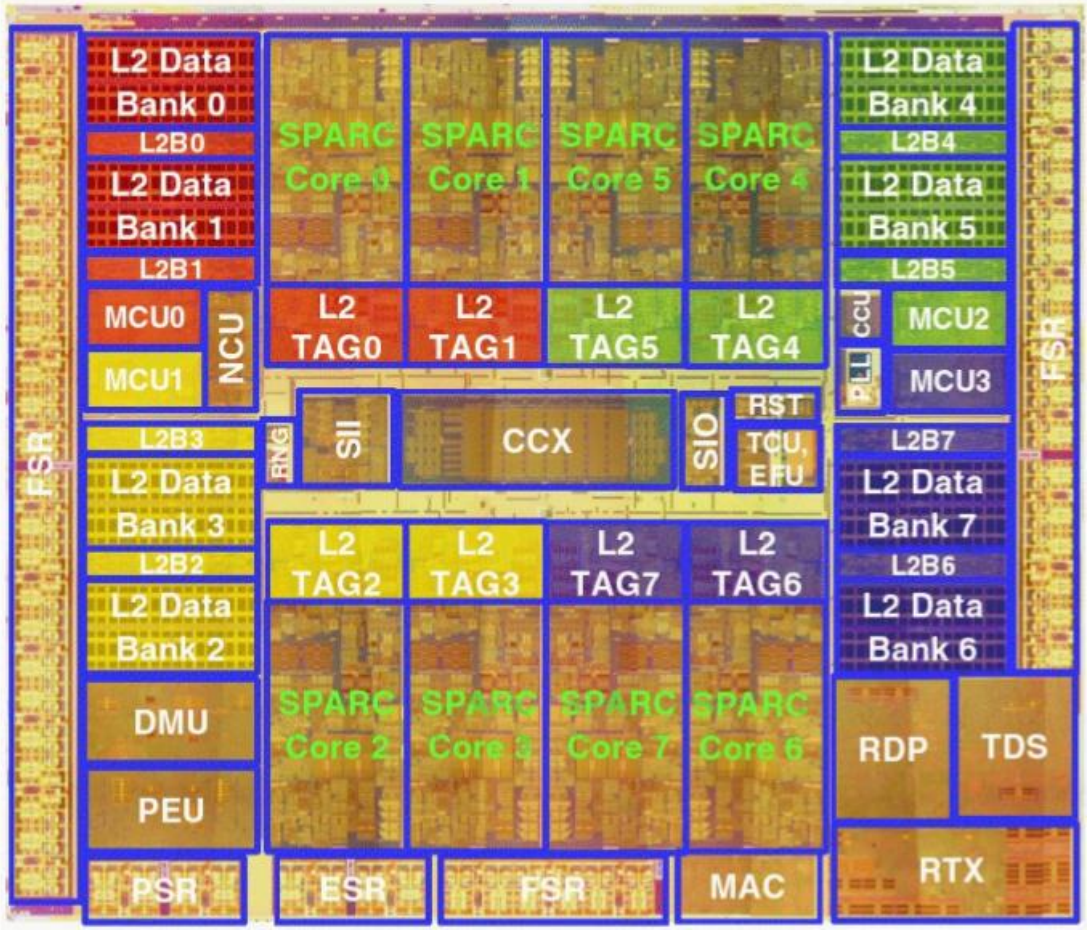
- Also designed for commercial applications

Table 1. Commercial server applications.

Benchmark	Application category	Instruction-level parallelism	Thread-level parallelism	Working set	Data sharing
Web99	Web server	Low	High	Large	Low
JBB	Java application server	Low	High	Large	Medium
TPC-C	Transaction processing	Low	High	Large	High
SAP-2T	Enterprise resource planning	Medium	High	Medium	Medium
SAP-3T	Enterprise resource planning	Low	High	Large	High
TPC-H	Decision support system	High	High	Large	Medium



Sun Niagara II (UltraSPARC T2)



- 8 SPARC cores, 8 threads/core. 8 stages. 16 KB I$ per Core. 8 KB D$ per Core. FP, Graphics, Crypto, units per Core.
- 4 MB Shared L2, 8 banks, 16-way set associative.
- 4 dual-channel FBDIMM memory controllers.
- X8 PCI-Express @ 2.5 Gb/s.
- Two 10G Ethernet ports @ 3.125 Gb/s.

Chip Multithreading (CMT)

- Spracklen and Abraham, "Chip Multithreading: Opportunities and Challenges," HPCA Industrial Session, 2005.

- Idea: Chip multiprocessor where each core is multithreaded
  - Niagara 1/2: fine grained multithreading
  - IBM POWER5: simultaneous multithreading

- Motivation: Tolerate memory latency better
  - A simple core stays idle on a cache miss
  - Multithreading enables tolerating cache miss latency when there is TLP

CMT (CMP + MT) vs. CMP

- Advantages of adding multithreading to each core
  - + Better memory latency tolerance when there are enough threads
  - + Fine grained multithreading can simplify core design (no need for branch prediction, dependency checking)
  - + Potentially better utilization of core, cache, memory resources
    - + Shared instructions and data among threads not replicated
    - + When one thread is not using a resource, another can

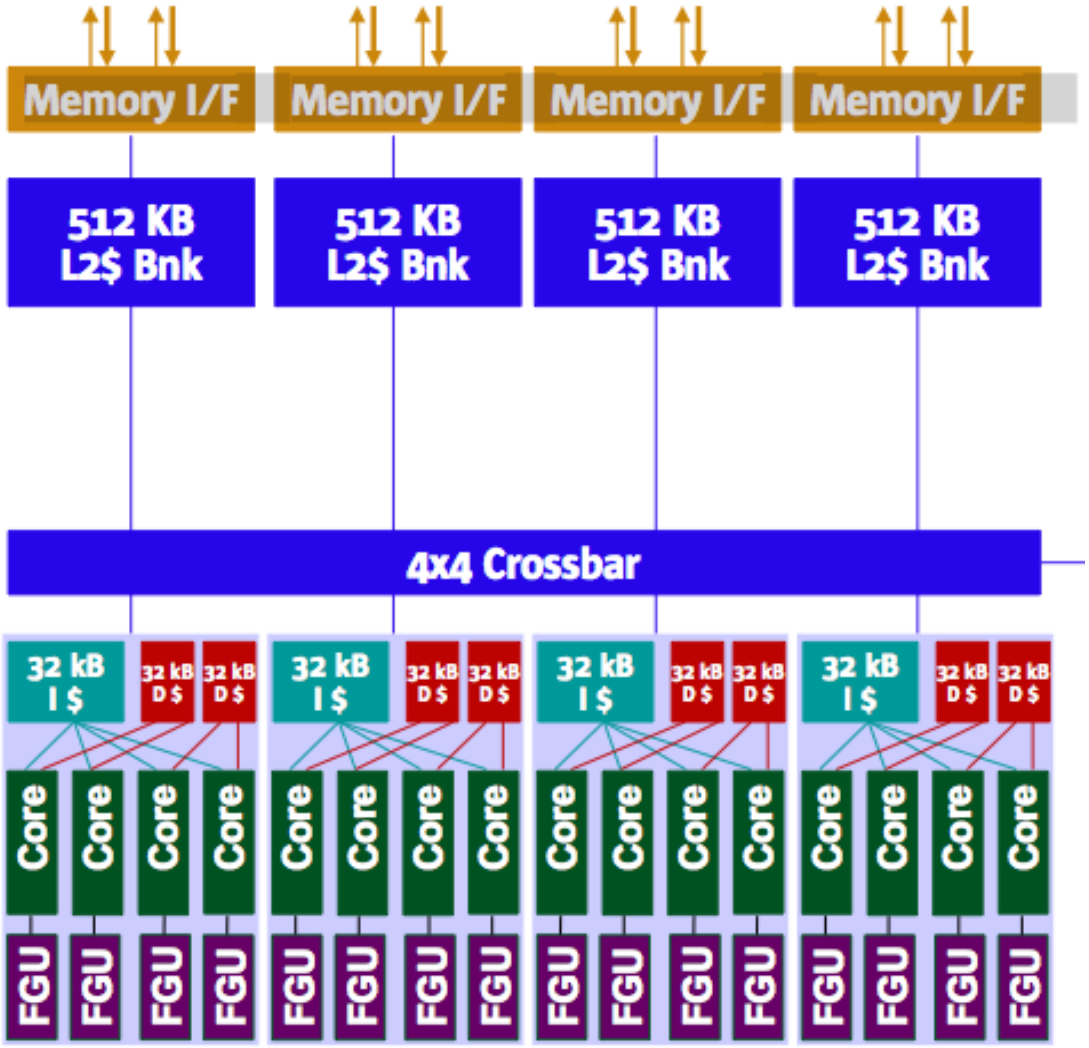
- Disadvantages
  - - Reduced single-thread performance (a thread does not have the core and L1 caches to itself)
  - - More pressure on the shared resources (cache, off-chip bandwidth) → more resource contention
  - - Applications with limited TLP do not benefit

Sun ROCK

- Chaudhry et al., "Rock: A High-Performance Sparc CMT Processor," IEEE Micro, 2009.
- Chaudhry et al., "Simultaneous Speculative Threading: A Novel Pipeline Architecture Implemented in Sun's ROCK Processor," ISCA 2009

- Goals:
  - Maximize throughput when threads are available
  - Boost single-thread performance when threads are not available and on cache misses
- Ideas:
  - Runahead on a cache miss → ahead thread executes miss-independent instructions, behind thread executes dependent instructions
  - Branch prediction (gshare)

Sun ROCK



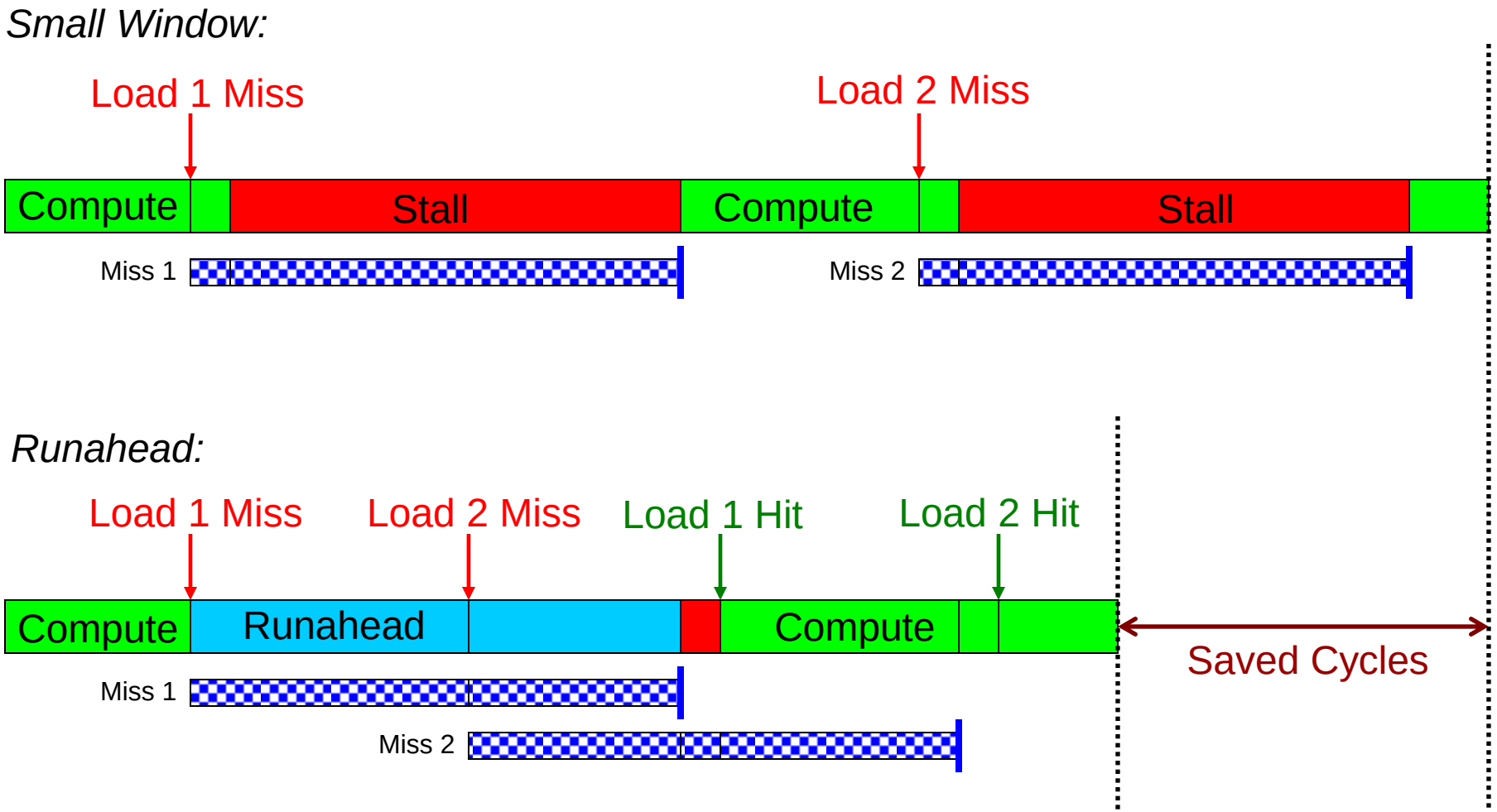
- 16 cores, 2 threads per core (fewer threads than Niagara 2)
- 4 cores share a 32KB instruction cache
- 2 cores share a 32KB data cache
- 2MB L2 cache (smaller than Niagara 2)

Runahead Execution (I)

- A simple pre-execution method for prefetching purposes
- Mutlu et al., "Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-order Processors," HPCA 2003.

- When the oldest instruction is a long-latency cache miss:
  - Checkpoint architectural state and enter runahead mode
- In runahead mode:
  - Speculatively pre-execute instructions
  - The purpose of pre-execution is to generate prefetches
  - L2-miss dependent instructions are marked INV and dropped
- Runahead mode ends when the original miss returns
  - Checkpoint is restored and normal execution resumes

Runahead Execution (II)

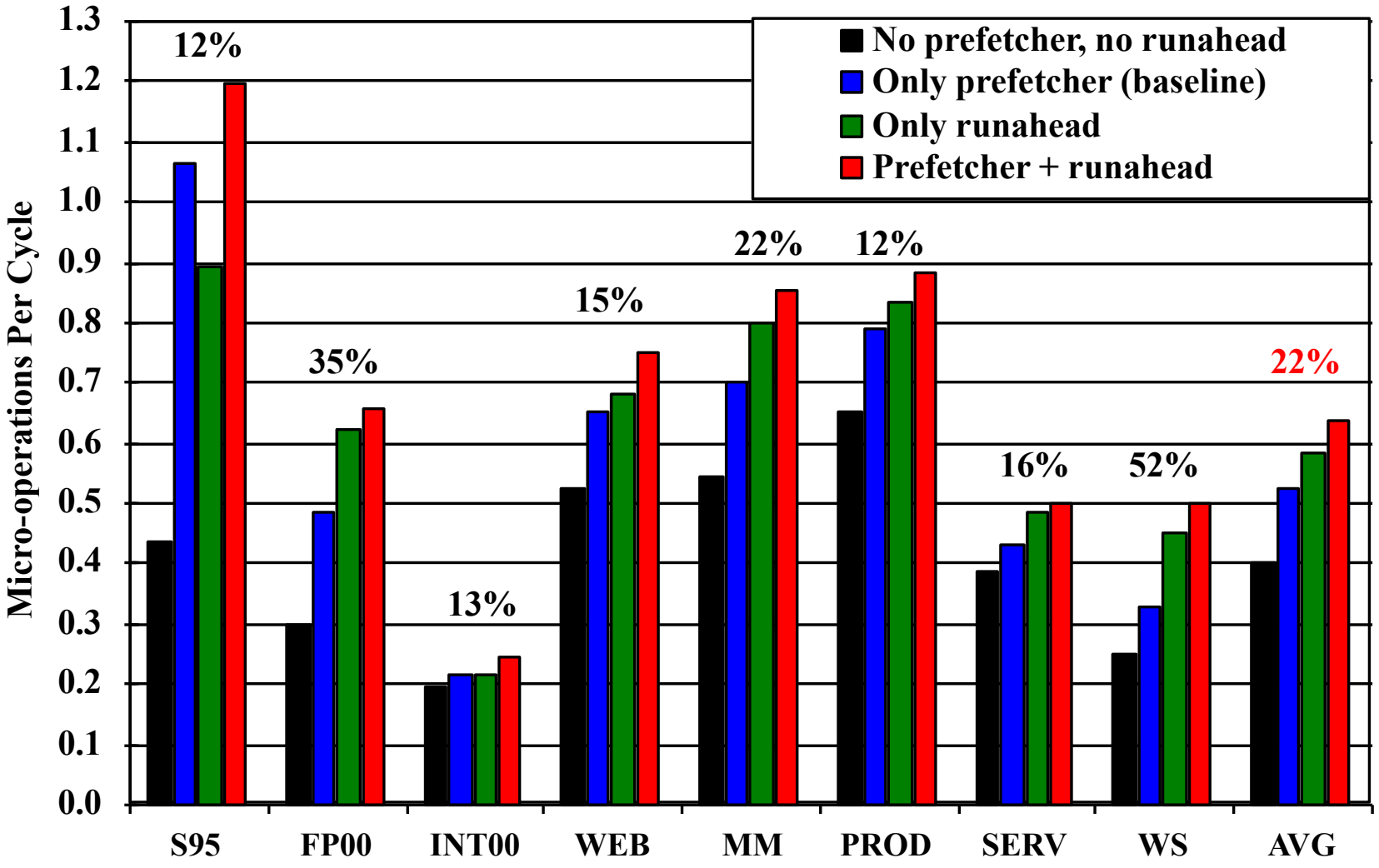


Runahead Execution (III)

- Advantages
  + Very accurate prefetches for data/instructions (all cache levels)
    + Follows the program path
  + Simple to implement, most of the hardware is already built in
- Disadvantages
  -- Extra executed instructions
- Limitations
  -- Limited by branch prediction accuracy
  -- Cannot prefetch dependent cache misses. Solution?
  -- Effectiveness limited by available Memory Level Parallelism

- Mutlu et al., "Efficient Runahead Execution: Power-Efficient Memory Latency Tolerance," IEEE Micro Jan/Feb 2006.

Performance of Runahead Execution

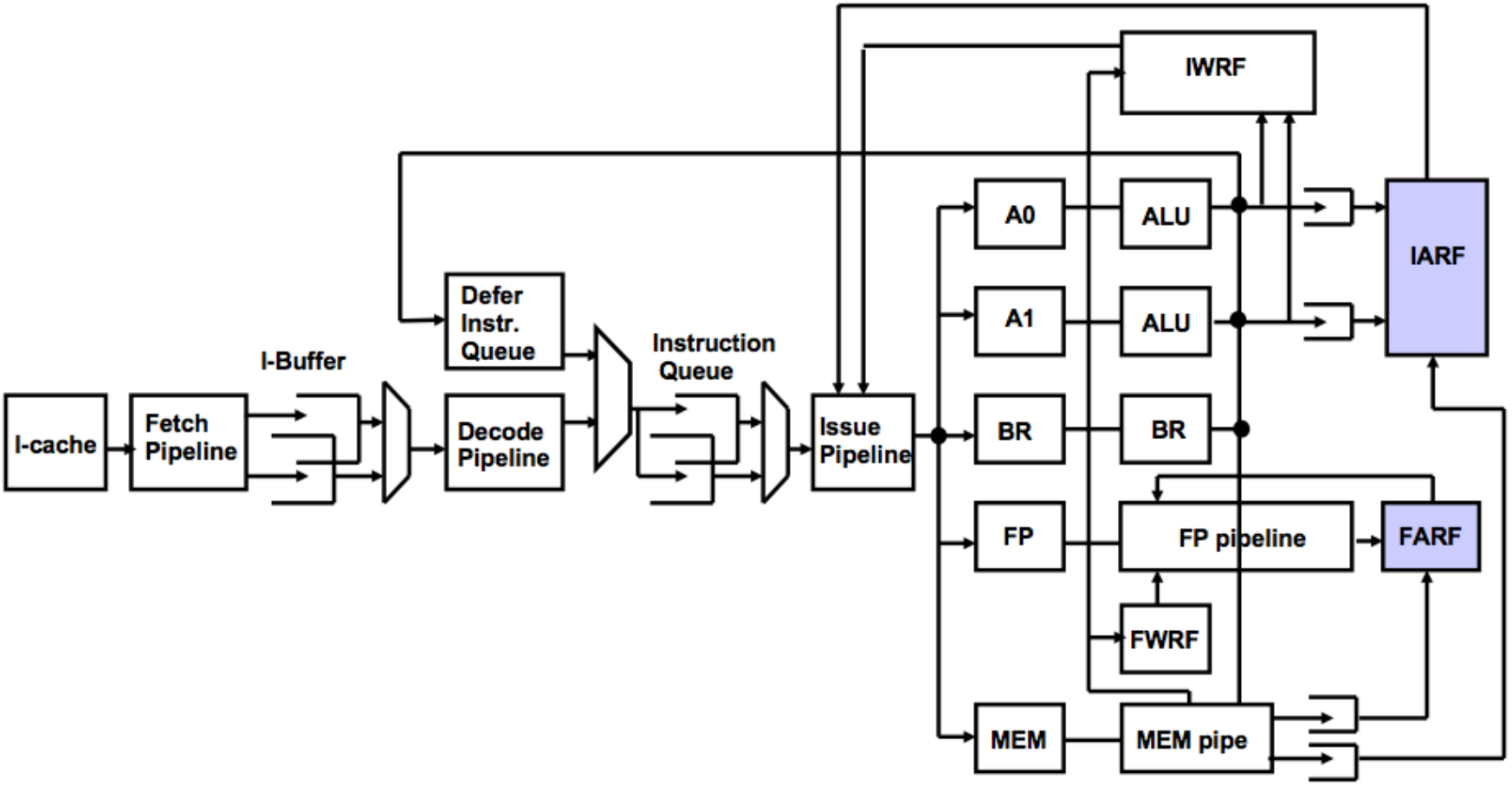


Sun ROCK Cores

- Load miss in L1 cache starts parallelization using 2 HW threads
- Ahead thread
  - Checkpoints state and executes speculatively
  - Instructions independent of load miss are speculatively executed
  - Load miss(es) and dependent instructions are deferred to behind thread
- Behind thread
  - Executes deferred instructions and re-defers them if necessary

- Memory-Level Parallelism (MLP)
  - Run ahead on load miss and generate additional load misses
- Instruction-Level Parallelism (ILP)
  - Ahead and behind threads execute independent instructions from different points in program in parallel

ROCK Pipeline



More Powerful Cores in Sun ROCK

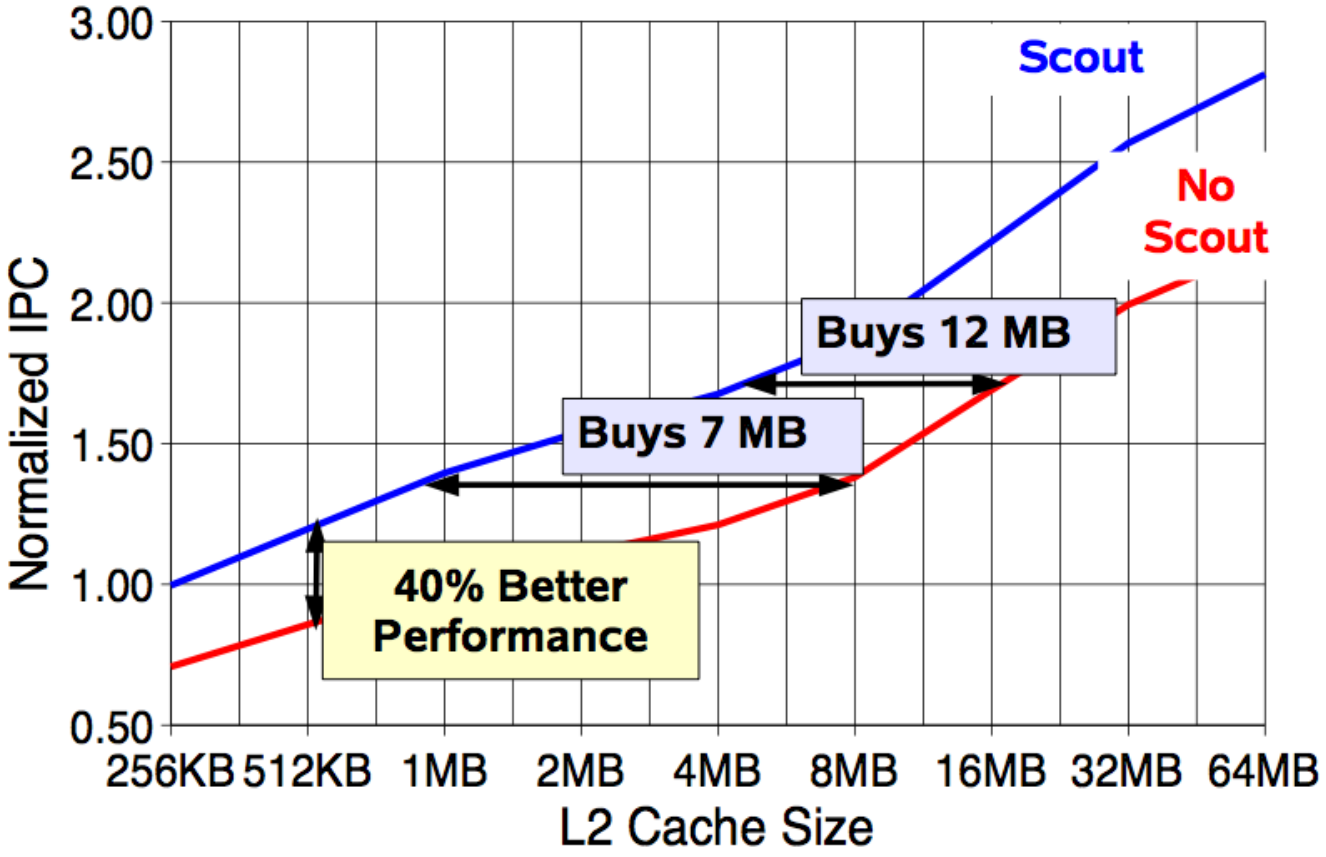
- Advantages
  + Higher single-thread performance (MLP + ILP)
  + Better cache miss tolerance → Can reduce on-chip cache sizes

- Disadvantages
  - Bigger cores → Fewer cores → Lower parallel throughput (in terms of threads).

    How about each thread's response time?
  - More complex than Niagara cores (but simpler than conventional out-of-order execution) → Longer design time?

More Powerful Cores in Sun ROCK

- Chaudhry talk, Aug 2008.



More Powerful Cores in Sun ROCK

- Chaudhry et al., "Simultaneous Speculative Threading: A Novel Pipeline Architecture Implemented in Sun's ROCK Processor," ISCA 2009

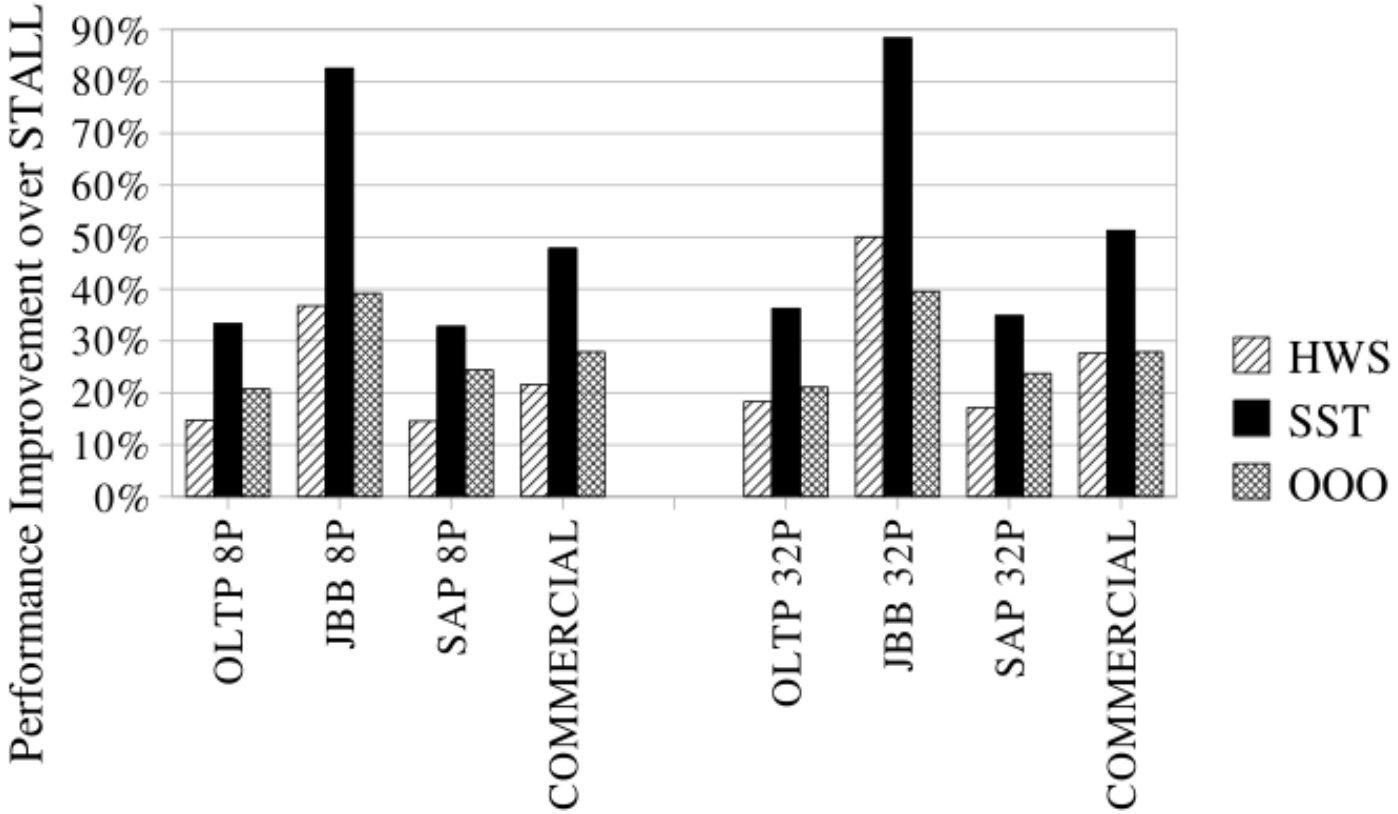
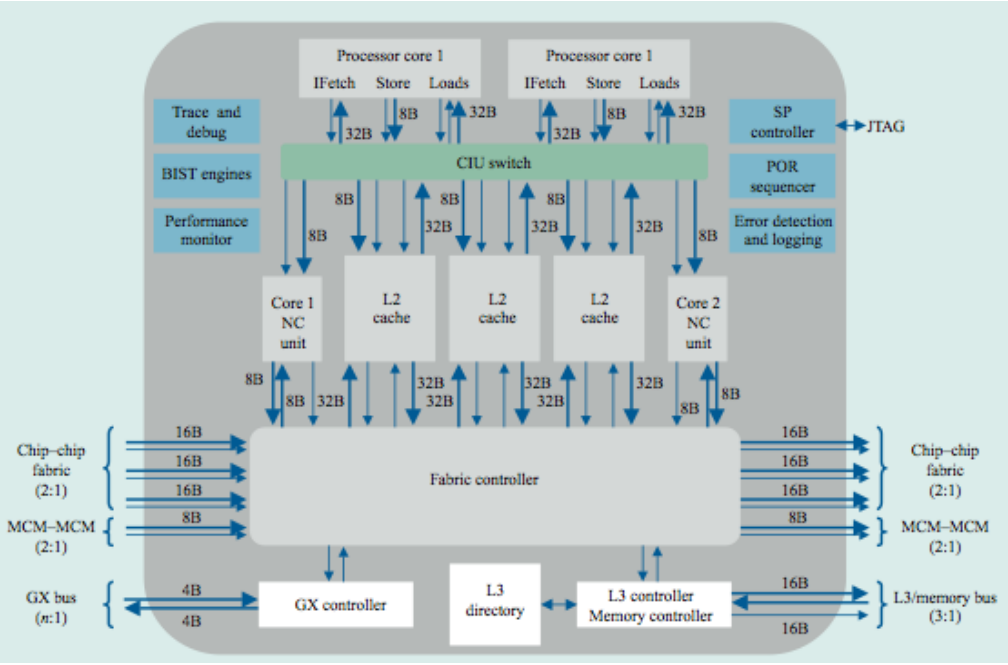


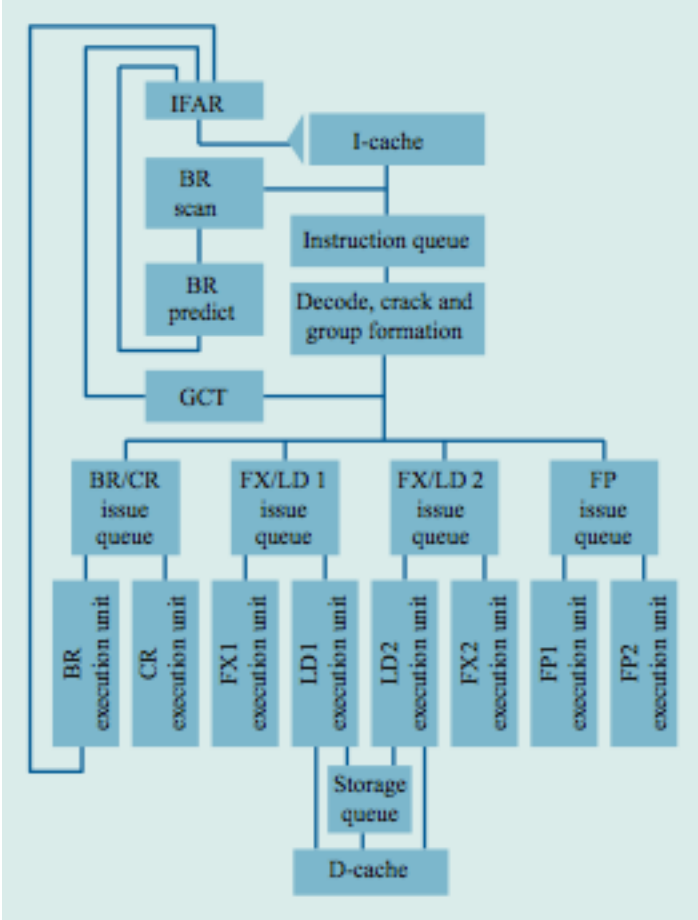
Figure 9: Commercial Performance.

IBM POWER4

- Tendler et al., "POWER4 system microarchitecture," IBM J R&D, 2002.

- Another symmetric multi-core chip…
- But, fewer and more powerful cores





IBM POWER4

- 2 cores, out-of-order execution
- 100-entry instruction window in each core
- 8-wide instruction fetch, issue, execute
- Large, local+global hybrid branch predictor
- 1.5MB, 8-way L2 cache
- Aggressive stream based prefetching

IBM POWER5

- Kalla et al., "IBM Power5 Chip: A Dual-Core Multithreaded Processor," IEEE Micro 2004.

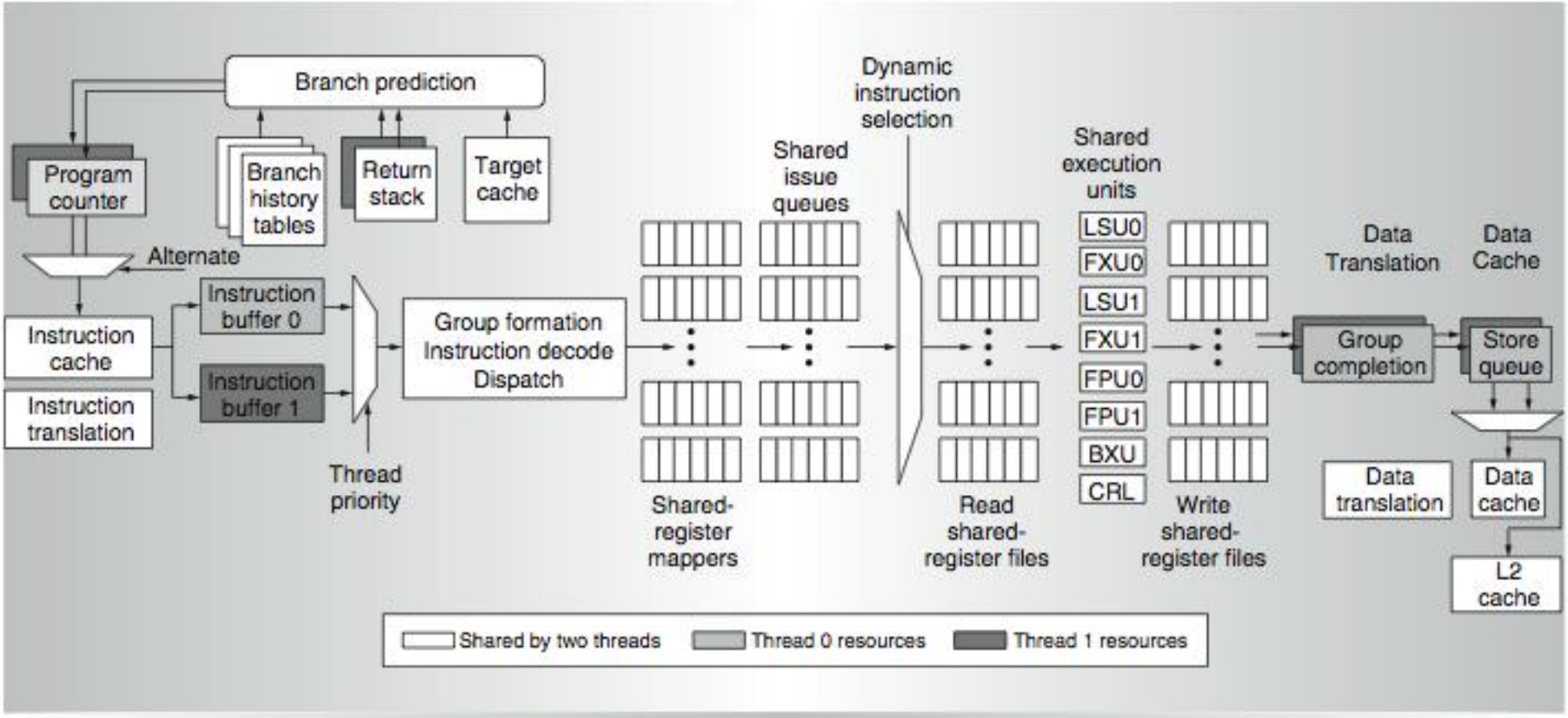
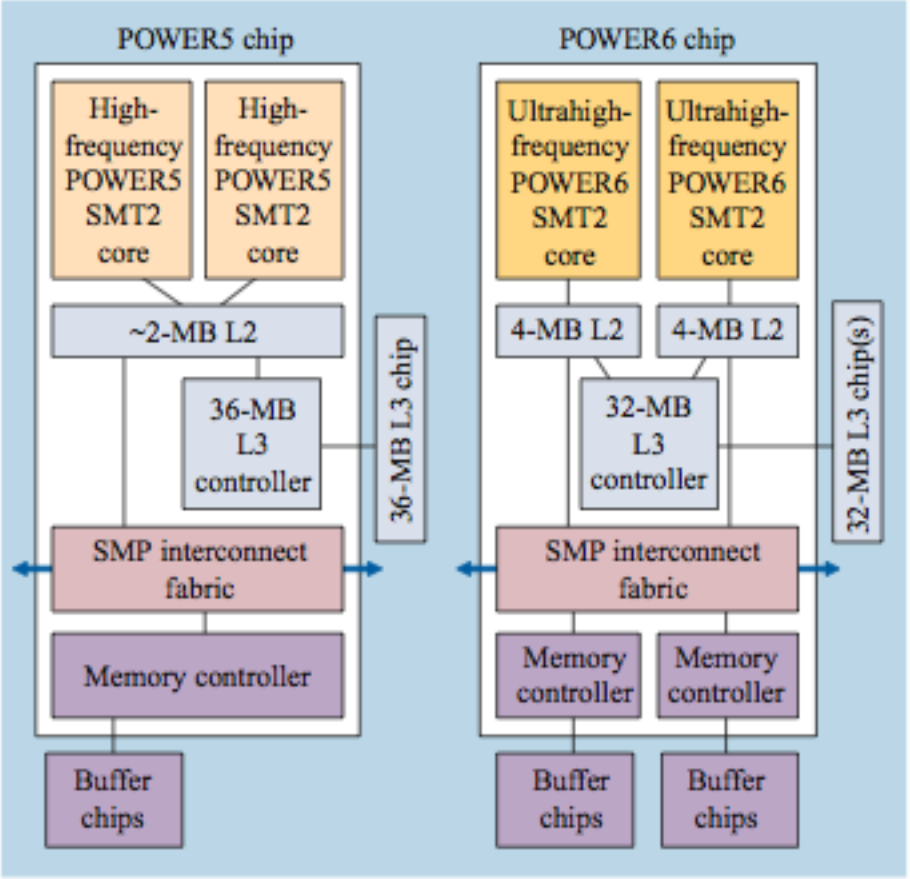


Figure 4. Power5 instruction data flow (BXU = branch execution unit and CRL = condition register logical execution unit).

IBM POWER6

- Le et al., "IBM POWER6 microarchitecture," IBM J R&D, 2007.

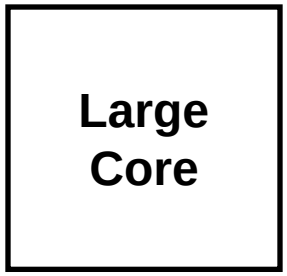
- 2 cores, in order, high frequency (4.7 GHz)
- 8 wide fetch
- Simultaneous multithreading in each core
- Runahead execution in each core
  - Similar to Sun ROCK



IBM POWER7

- Kalla et al., "Power7: IBM's Next-Generation Server Processor," IEEE Micro 2010.
- 8 out-of-order cores, 4-way SMT in each core
- TurboCore mode
  - Can turn off cores so that other cores can be run at higher frequency

Large vs. Small Cores



- *Out-of-order*
- *Wide fetch e.g. 4-wide*
- *Deeper pipeline*
- *Aggressive branch predictor (e.g. hybrid)*
- *Multiple functional units*
- *Trace cache*
- *Memory dependence speculation*

- *In-order*
- *Narrow Fetch e.g. 2-wide*
- *Shallow pipeline*
- *Simple branch predictor (e.g. Gshare)*
- *Few functional units*

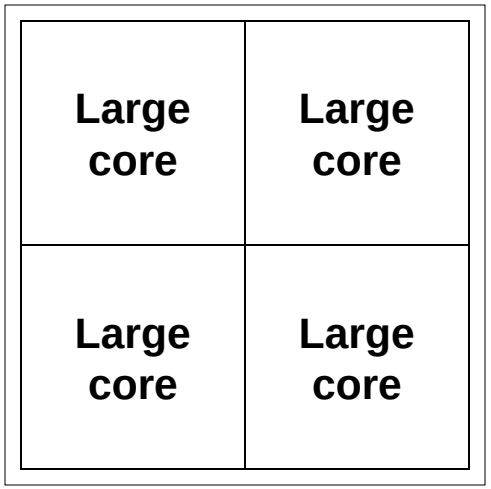
Large Cores are power inefficient:
e.g., 2x performance for 4x area (power)

Large vs. Small Cores

- Grochowski et al., "Best of both Latency and Throughput," ICCD 2004.

	Large core	Small core
Microarchitecture	Out-of-order, 128-256 entry ROB	In-order
Width	3-4	1
Pipeline depth	20-30	5
Normalized performance	5-8x	1x
Normalized power	20-50x	1x
Normalized energy/instruction	4-6x	1x

Tile-Large Approach



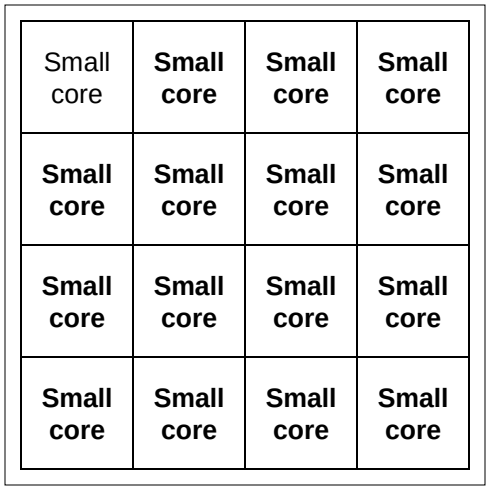
"Tile-Large"

- Tile a few large cores
- IBM Power 5, AMD Barcelona, Intel Core2Quad, Intel Nehalem

+ High performance on single thread, serial code sections (2 units)

- Low throughput on parallel program portions (8 units)

Tile-Small Approach



"Tile-Small"

- Tile many small cores
- Sun Niagara, Intel Larrabee, Tilera TILE (tile ultra-small)

+ High throughput on the parallel part (16 units)

- Low performance on the serial part, single thread (1 unit)

Can We Get the Best of Both worlds?

- Tile Large

  + High performance on single thread, serial code sections (2 units)

  - Low throughput on parallel program portions (8 units)

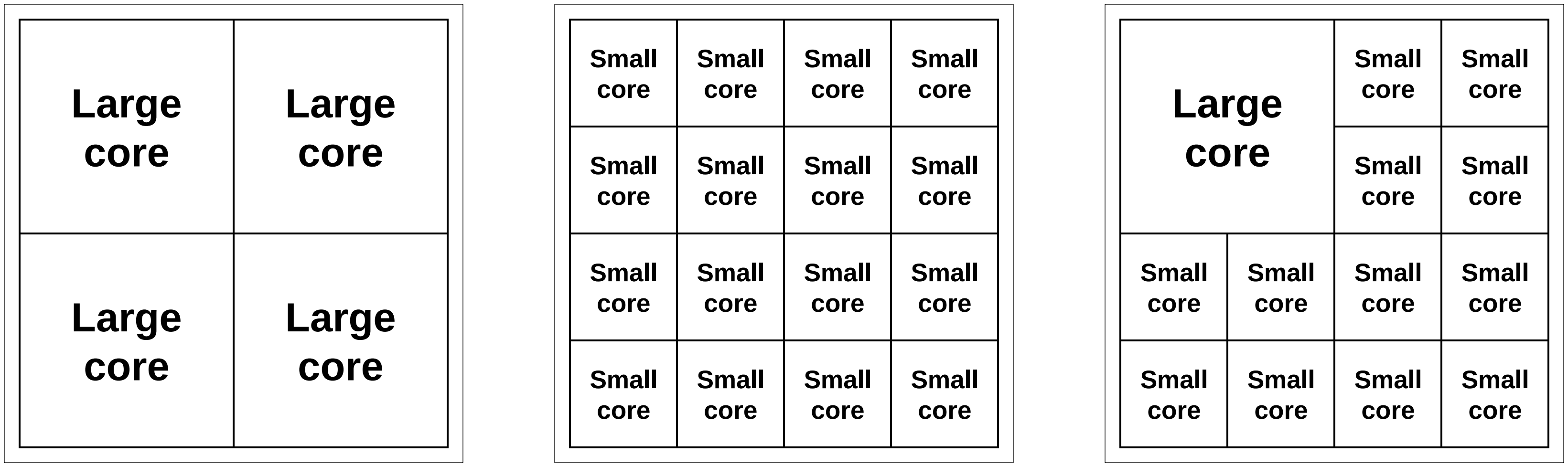
- Tile Small

  + High throughput on the parallel part (16 units)

  - Low performance on the serial part, single thread (1 unit), reduced single-thread performance compared to existing single thread processors

- Idea: Have both large and small on the same chip → Performance asymmetry

Asymmetric Chip Multiprocessor (ACMP)



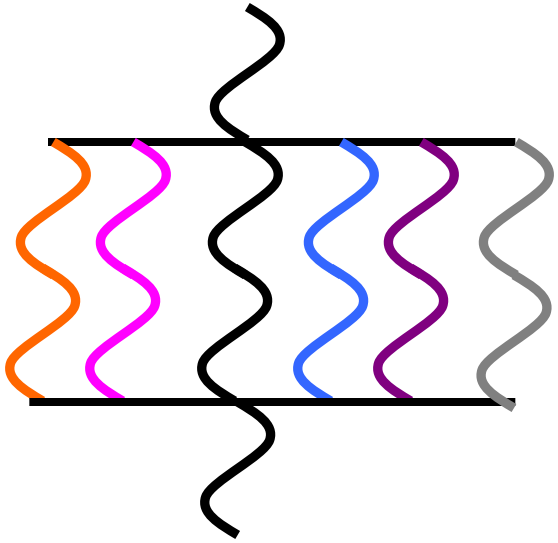
"Tile-Large" "Tile-Small" ACMP

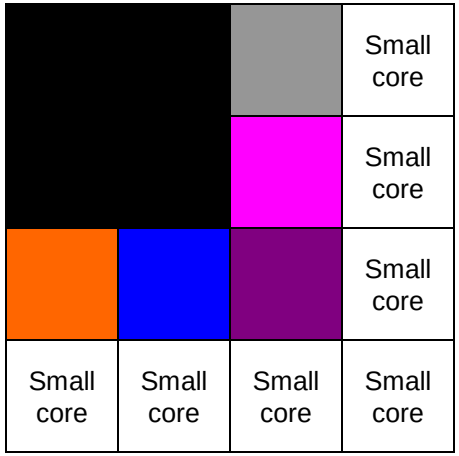
- Provide one large core and many small cores

+ Accelerate serial part using the large core (2 units)
+ Execute parallel part on all cores for high throughput (14 units)

Accelerating Serial Bottlenecks

Single thread → Large core





ACMP Approach

Performance vs. Parallelism

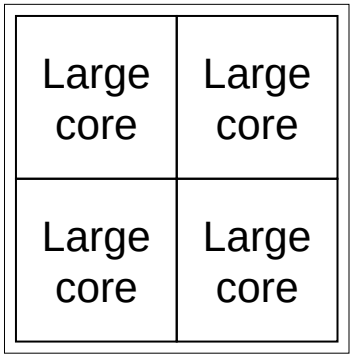
Assumptions:

1. *Small core takes an area budget of 1 and has performance of 1*

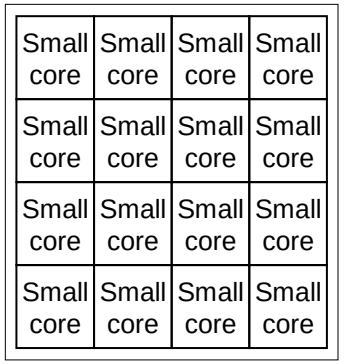
2. *Large core takes an area budget of 4 and has performance of 2*

ACMP Performance vs. Parallelism

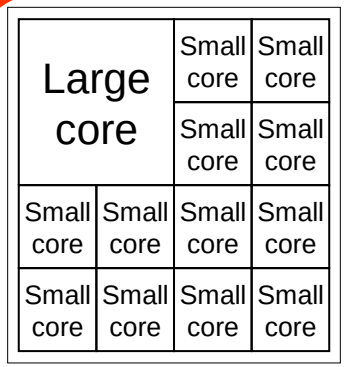
Area-budget = 16 small cores



"Tile-Large"



"Tile-Small"



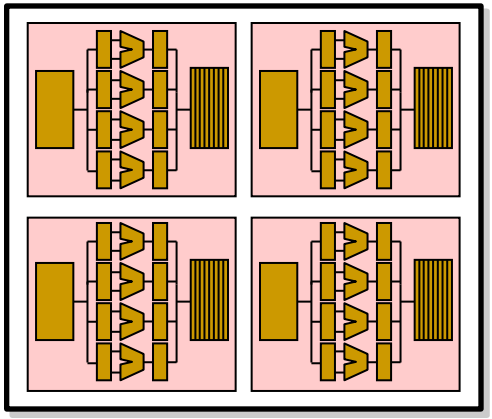
ACMP

	"Tile-Large"	"Tile-Small"	ACMP
Large Cores	4	0	1
Small Cores	0	16	12
Serial Performance	2	1	2
Parallel Throughput	2 x 4 = 8	1 x 16 = 16	1x2 + 1x12 = 14

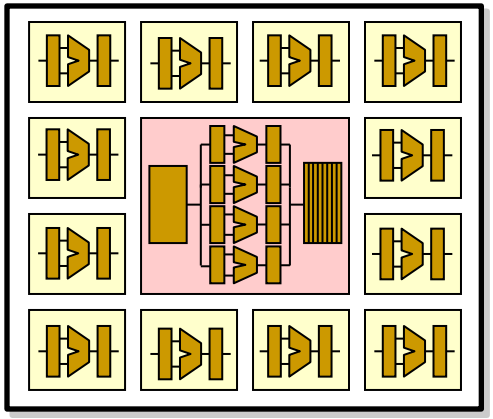
50

Some Analysis

- Hill and Marty, "Amdahl's Law in the Multi-Core Era," IEEE Computer 2008.
- Each Chip Bounded to N BCEs (Base Core Equivalents)
- One R-BCE Core leaves N-R BCEs
- Use N-R BCEs for N-R Base Cores
- Therefore, 1 + N - R Cores per Chip
- For an N = 16 BCE Chip:



Symmetric: Four 4-BCE cores



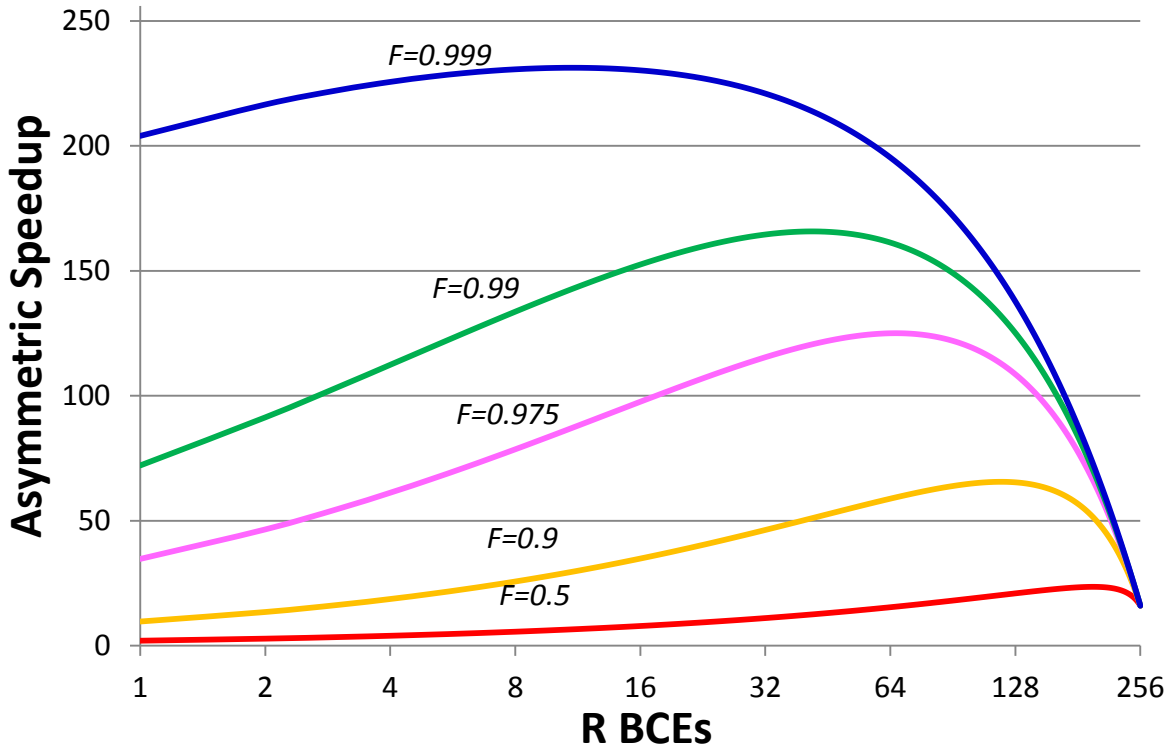
Asymmetric: One 4-BCE core & Twelve 1-BCE base cores

Amdahl's Law Modified

- Serial Fraction 1-F same, so time = (1 – F) / Perf(R)
- Parallel Fraction F
  - One core at rate Perf(R)
  - N-R cores at rate 1
  - Parallel time = F / (Perf(R) + N - R)
- Therefore, w.r.t. one base core:

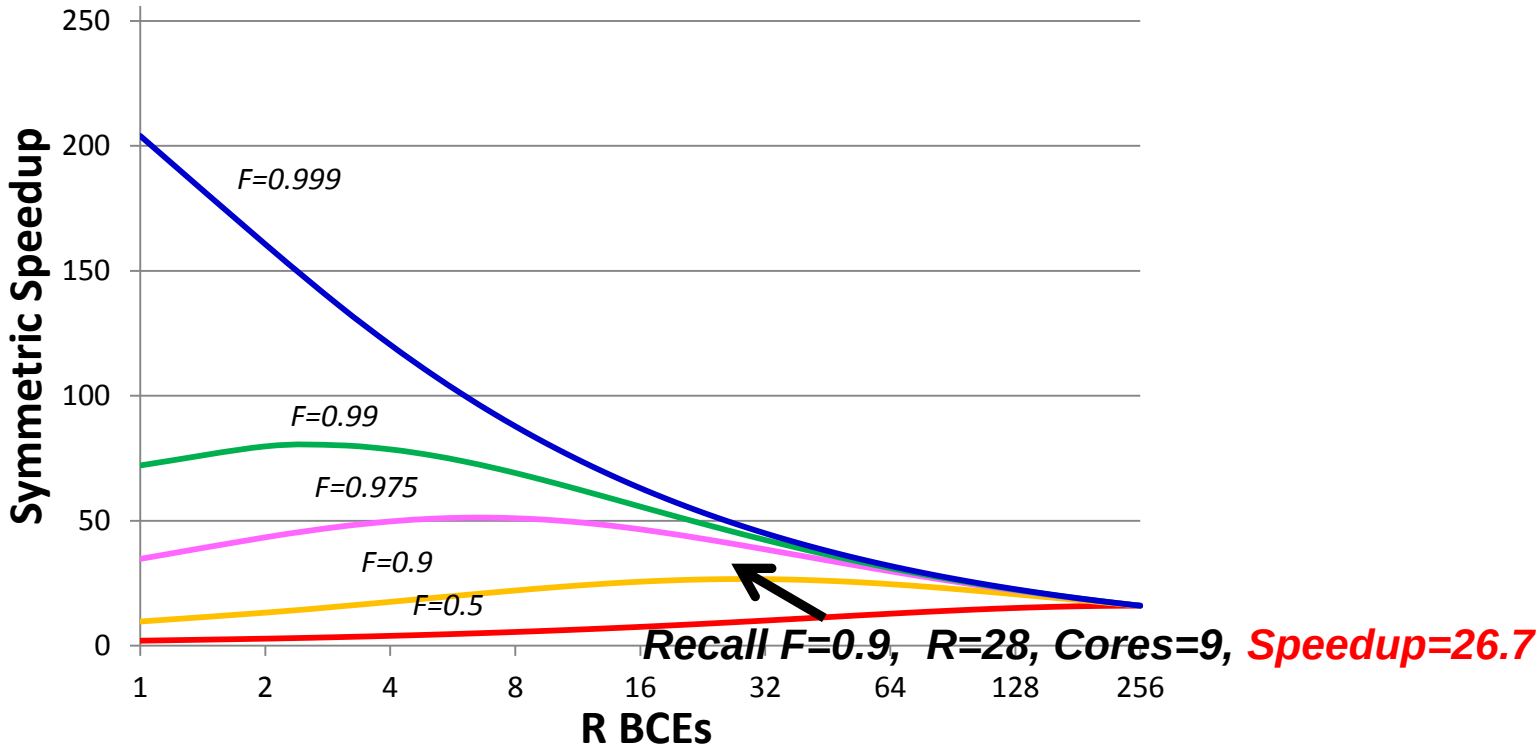
$$\text{Asymmetric Speedup} = \frac{1}{\frac{1 - F}{\text{Perf}(R)} + \frac{F}{\text{Perf}(R) + N - R}}$$

Asymmetric Multicore Chip, N = 256 BCEs

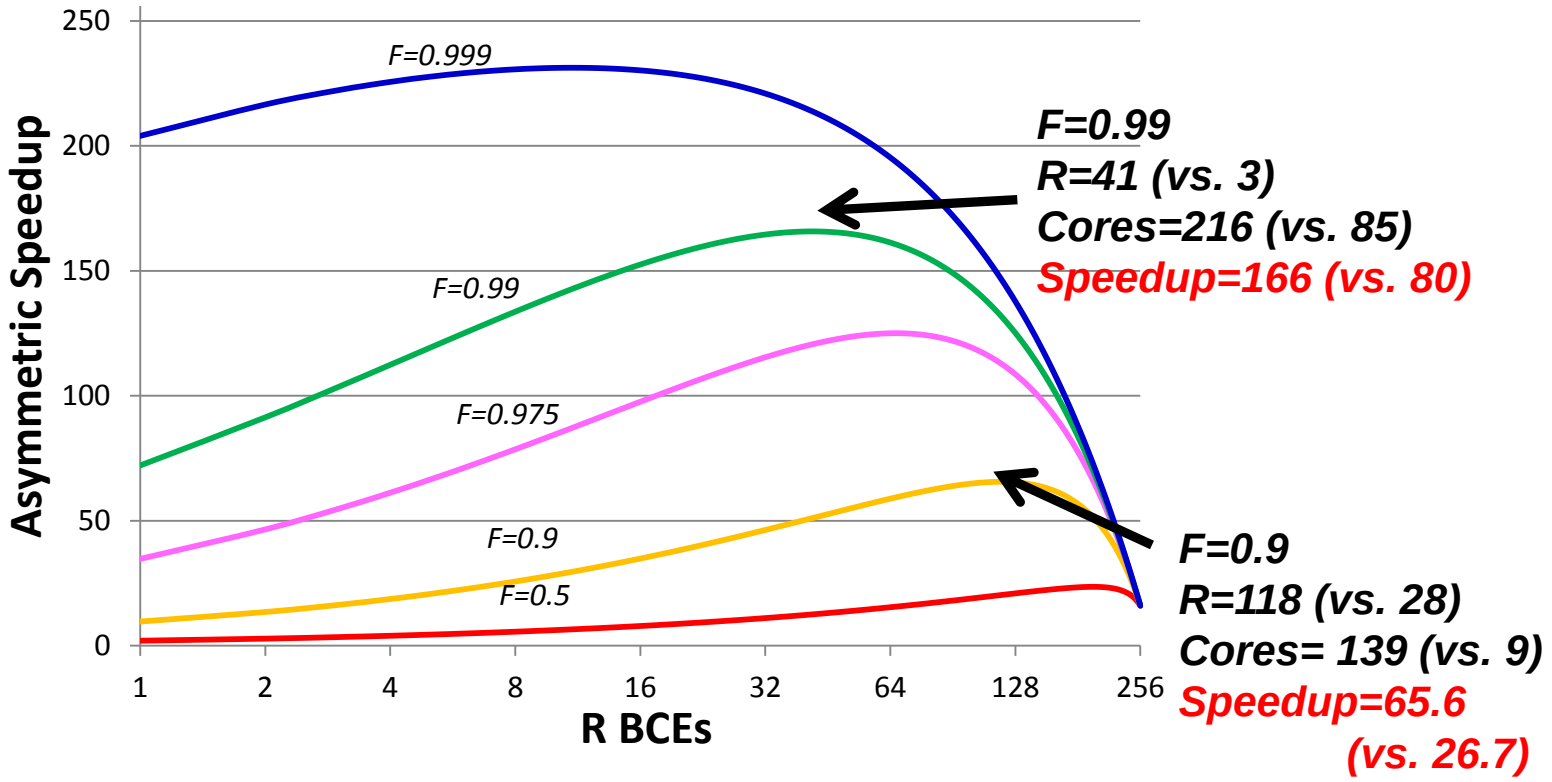


- Number of Cores = 1 (Enhanced) + 256 – R (Base)

Symmetric Multicore Chip, N = 256 BCEs



Asymmetric Multicore Chip, N = 256 BCEs



- Asymmetric multi-core provides better speedup than symmetric multi-core when N is large

Asymmetric vs. Symmetric Cores

- Advantages of Asymmetric
  - + Can provide better performance when thread parallelism is limited
  - + Can be more energy efficient
    - + Schedule computation to the core type that can best execute it
- Disadvantages
  - - Need to design more than one type of core. Always?
  - - Scheduling becomes more complicated
    - - What computation should be scheduled on the large core?
    - - Who should decide? HW vs. SW?
  - - Managing locality and load balancing can become difficult if threads move between cores (transparently to software)
  - - Cores have different demands from shared resources

How to Achieve Asymmetry

- Static
  - Type and power of cores fixed at design time
  - Two approaches to design "faster cores":
    - High frequency
    - Build a more complex, powerful core with entirely different uarch
  - Is static asymmetry natural? (chip-wide variations in frequency)

- Dynamic
  - Type and power of cores change dynamically
  - Two approaches to dynamically create "faster cores":
    - Boost frequency dynamically (limited power budget)
    - Combine small cores to enable a more complex, powerful core
    - Is there a third, fourth, fifth approach?

Asymmetry via Boosting of Frequency

- Static
  - Due to process variations, cores might have different frequency
  - Simply hardwire/design cores to have different frequencies
- Dynamic
  - Annavaram et al., "Mitigating Amdahl's Law Through EPI Throttling," ISCA 2005.
  - Dynamic voltage and frequency scaling

EPI Throttling

- Goal: Minimize execution time of parallel programs while keeping power within a fixed budget
- For best scalar and throughput performance, vary energy expended per instruction (EPI) based on available parallelism
  - P = EPI •IPS
  - P = fixed power budget
  - EPI = energy per instruction
  - IPS = aggregate instructions retired per second
- Idea: For a fixed power budget
  - Run sequential phases on high-EPI processor
  - Run parallel phases on multiple low-EPI processors

EPI Throttling via DVFS

- DVFS: Dynamic voltage frequency scaling
- In phases of low thread parallelism
  - Run a few cores at high supply voltage and high frequency
- In phases of high thread parallelism
  - Run many cores at low supply voltage and low frequency

Possible EPI Throttling Techniques

- Grochowski et al., "Best of both Latency and Throughput," ICCD 2004.

Method	EPI Range	Time to Alter EPI	Throttle Action
Voltage/frequency scaling	1:2 to 1:4	100us (ramp Vcc)	Lower voltage and frequency
Asymmetric cores	1:4 to 1:6	10us (migrate 256KB L2 cache)	Migrate threads from large cores to small cores
Variable-size core	1:1 to 1:2	1us (fill 32KB L1 cache)	Reduce capacity of processor resources
Speculation control	1:1 to 1:1.4	10ns (pipeline latency)	Reduce amount of speculation

Boosting Frequency of a Small Core vs. Large Core

- Frequency boosting implemented on Intel Nehalem, IBM POWER7

- Advantages of Boosting Frequency

  + Very simple to implement; no need to design a new core

  + Parallel throughput does not degrade when TLP is high

  + Preserves locality of boosted thread

- Disadvantages

  - Does not improve performance if thread is memory bound

  - Does not reduce Cycles per Instruction (remember the performance equation?)

  - Changing frequency/voltage can take longer than switching to a large core

We did not cover the following slides in lecture. These are for your preparation for the next lecture.

EPI Throttling (Annavaram et al., ISCA'05)

- Static AMP
  - Duty cycles set once prior to program run
  - Parallel phases run on 3P/1.25GHz
  - Sequential phases run on 1P/2GHz
  - Affinity guarantees sequential on 1P and parallel on 3
  - Benchmarks that rapidly transition between sequential and parallel phases
- Dynamic AMP
  - Duty cycle changes during program run
  - Parallel phases run on all or a subset of four processors
  - Sequential phases of execution on 1P/2GHz
  - Benchmarks with long sequential and parallel phases

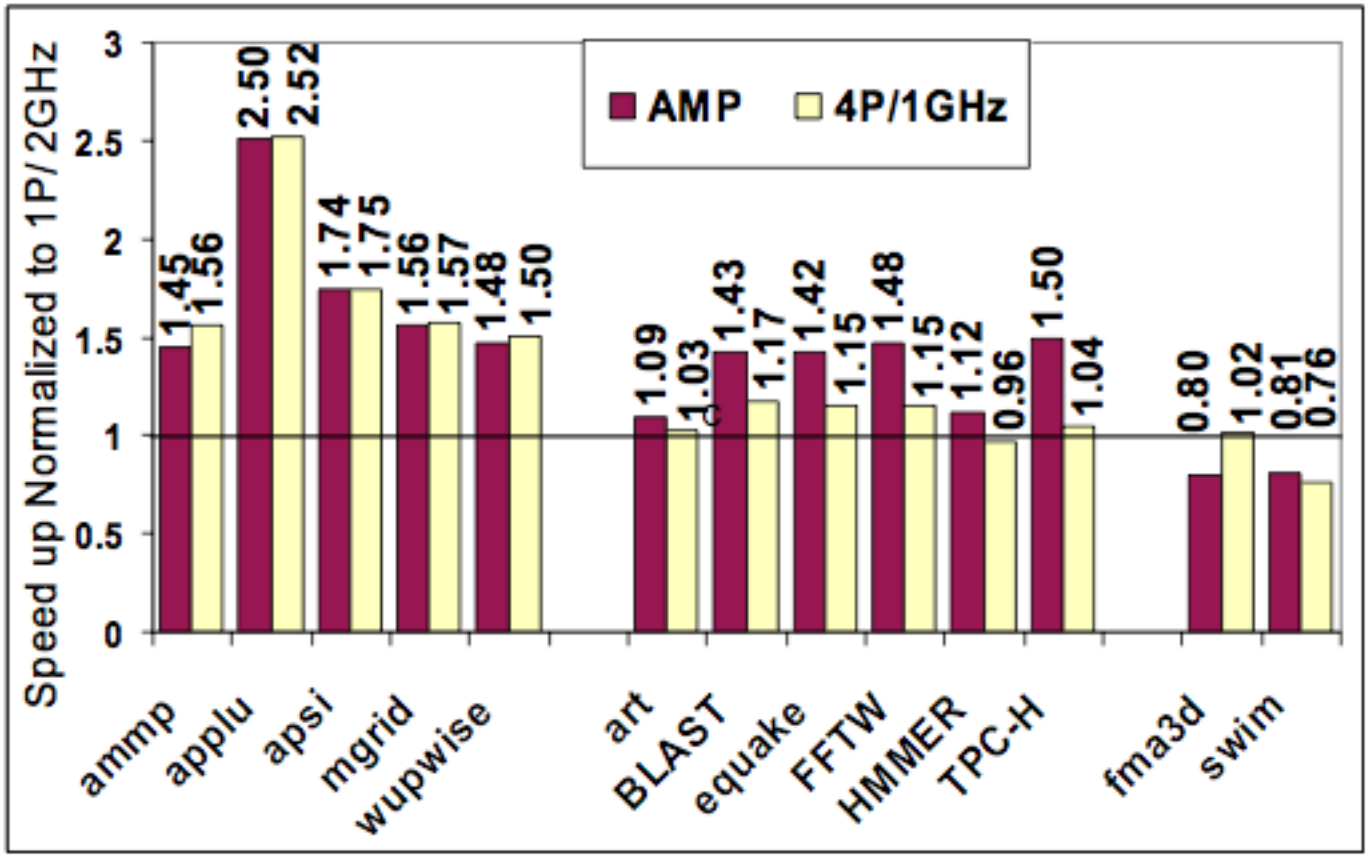
EPI Throttling (Annavaram et al., ISCA'05)

- Evaluation on Base SMP: 4 Base SMP: 4-way 2GHz Xeon, 2MB L3, 4GB Memory

- Hand-modified programs
  - OMP threads set to 3 for static AMP
  - Calls to set affinity in each thread for static AMP
  - Calls to change duty cycle and to set affinity in dynamic AMP

AMP Configuration	Programs
Static AMP: 1P/2GHz or 3P/1.25GHz	wupwise, swim, mgrid, equake, fma3d, art, ammp, BLAST, HMMER
Dynamic AMP: 1P/2GHz to 4P/1GHz	applu, apsi, FFTW, TPC-H

EPI Throttling (Annavaram et al., ISCA'05)



- Frequency boosting AMP improves performance compared to 4-way SMP for many applications

EPI Throttling

- Why does Frequency Boosting (FB) AMP not always improve performance?

- Loss of throughput in static AMP (only 3 processors in parallel portion)
  - Is this really the best way of using FB-AMP?

- Rapid transitions between serial and parallel phases
  - Data/thread migration and throttling overhead

- Boosting frequency does not help memory-bound phases

Review So Far

- Symmetric Multicore
  - Evolution of Sun's and IBM's Multicore systems and design choices
  - Niagara, Niagara 2, ROCK
  - IBM POWERx
- Asymmetric multicore
  - Motivation
  - Functional vs. Performance Asymmetry
  - Static vs. Dynamic Asymmetry
  - EPI Throttling

Design Tradeoffs in ACMP (I)

- Hardware Design Effort vs. Programmer Effort

  - ACMP requires more design effort

  + Performance becomes less dependent on length of the serial part

  + Can reduce programmer effort: Serial portions are not as bad for performance with ACMP

- Migration Overhead vs. Accelerated Serial Bottleneck

  + Performance gain from faster execution of serial portion

  - Performance loss when architectural state is migrated/switched in when the master changes
    - Can be alleviated with multithreading and hidden by long serial portion

  - Serial portion incurs cache misses when it needs data generated by the parallel portion

  - Parallel portion incurs cache misses when it needs data generated by the serial portion

Design Tradeoffs in ACMP (II)

- Fewer threads vs. accelerated serial bottleneck

  + Performance gain from accelerated serial portion

  - Performance loss due to unavailability of L threads in parallel portion

  - This need not be the case → Large core can implement Multithreading to improve parallel throughput
  - As the number of cores (threads) on chip increases, fractional loss in parallel performance decreases

Uses of Asymmetry

- So far:
  - Improvement in serial performance (sequential bottleneck)

- What else can we do with asymmetry?
  - Energy reduction?
  - Energy/performance tradeoff?
  - Improvement in parallel portion?

Use of Asymmetry for Energy Efficiency

- Kumar et al., "Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction," MICRO 2003.

- Idea:
  - Implement multiple types of cores on chip
  - Monitor characteristics of the running thread (e.g., sample energy/perf on each core periodically)
  - Dynamically pick the core that provides the best energy/performance tradeoff for a given phase
    - "Best core" → Depends on optimization metric

Use of Asymmetry for Energy Efficiency

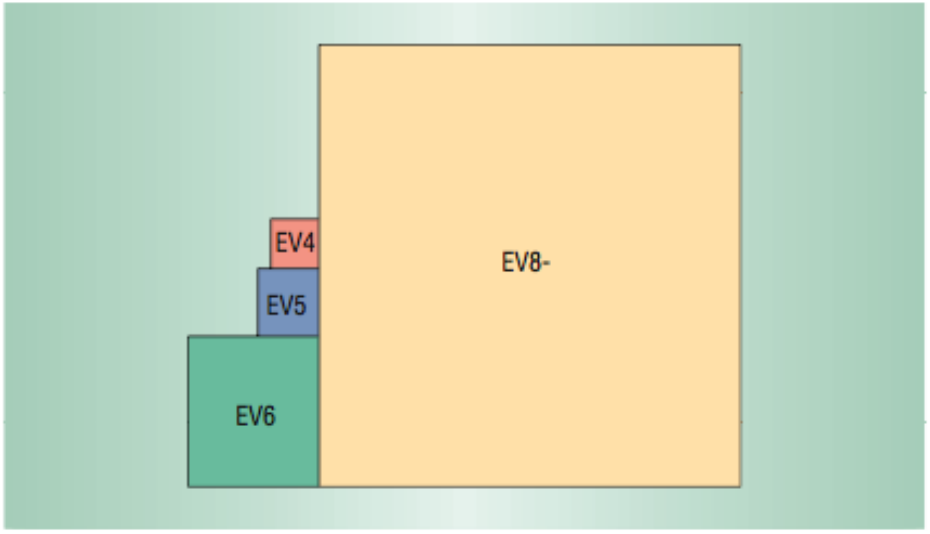
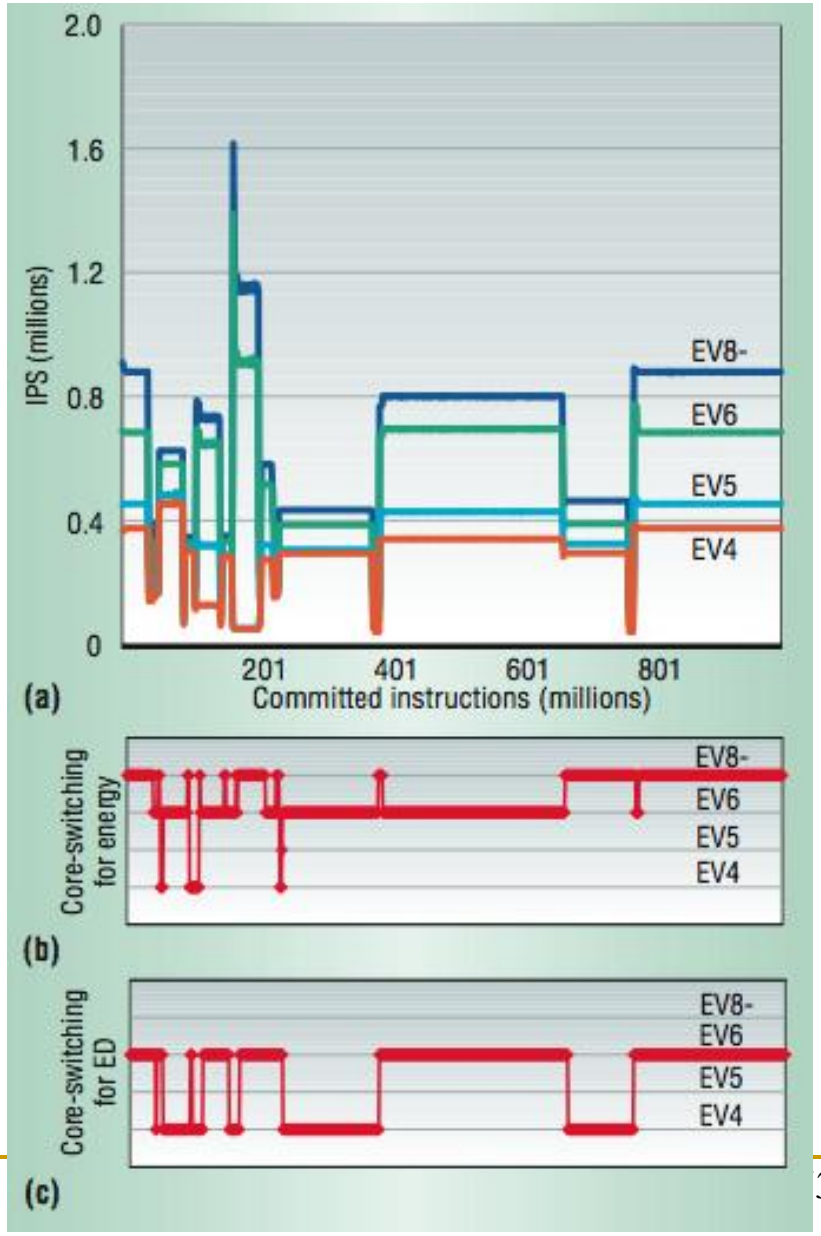


Figure 1. Relative sizes of the Alpha cores scaled to 0.10 µm. EV8 is 80 times bigger but provides only two to three times more single-threaded performance.

Table 1. Power and relative performance of Alpha cores scaled to 0.10 µm. Performance is expressed normalized to EV4 performance.

Core	Peak power (Watts)	Average power (Watts)	Performance (norm. IPC)
EV4	4.97	3.73	1.00
EV5	9.83	6.88	1.30
EV6	17.8	10.68	1.87
EV8	92.88	46.44	2.14



Use of Asymmetry for Energy Efficiency

- Advantages
  + More flexibility in energy-performance tradeoff
  + Can execute computation to the core that is best suited for it (in terms of energy)

- Disadvantages/issues
  - Incorrect predictions/sampling → wrong core → reduced performance or increased energy
  - Overhead of core switching
  - Disadvantages of asymmetric CMP (e.g., design multiple cores)
  - Need phase monitoring and matching algorithms
    - What characteristics should be monitored?
    - Once characteristics known, how do you pick the core?

Use of ACMP to Improve Parallel Portion Performance

- Mutual Exclusion:
  - Threads are not allowed to update shared data concurrently
- Accesses to shared data are encapsulated inside ***critical sections***
- Only one thread can execute a critical section at a given time
- Idea: Ship critical sections to a large core
- Suleman et al., "Accelerating Critical Section Execution with Asymmetric Multi-Core Architectures," ASPLOS 2009, IEEE Micro Top Picks 2010.