Thread-Level Parallelism and Multithreading

- Limitations of Instruction-Level Parallelism (ILP)
- Thread-Level Parallelism and Multithreading
  - Coarse-grained Multithreading
  - Fine-grained Multithreading
  - Simultaneous Multithreading
- Case Studies of Multithreaded Processors

Limits to Instruction-Level Parallelism (ILP)

Assumptions for ideal/perfect machine to start

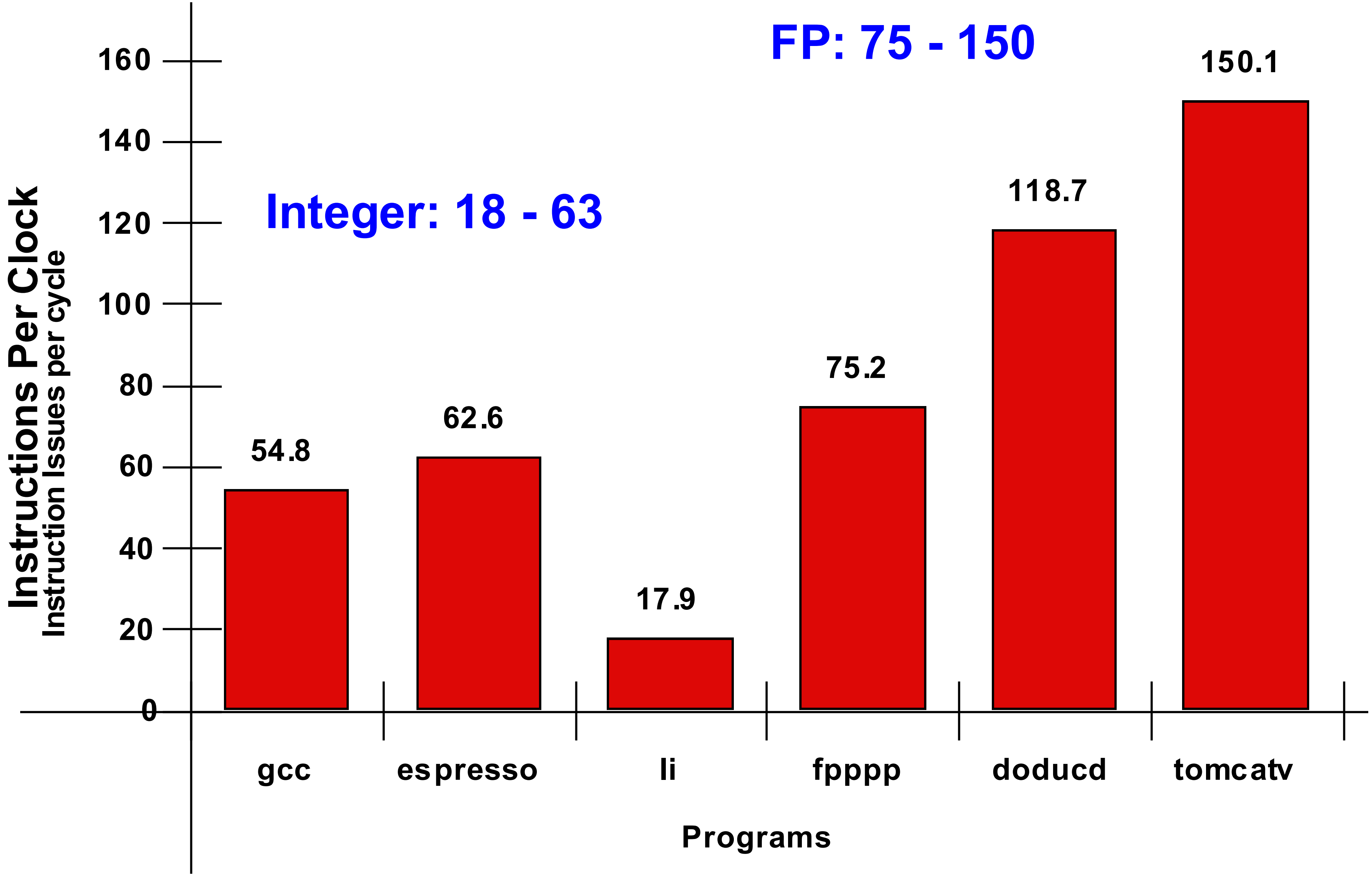
- Register renaming – infinite virtual registers
  - All register WAW & WAR hazards are avoided
- Memory-address alias analysis – addresses known and a load can be moved before a store provided addresses are not equal
  - The above two assumptions eliminate all but RAW
- Branch prediction – perfect; no mispredictions
- Jump prediction – all jump/returns perfectly predicted
  - no control dependencies; perfect speculation & an unbounded buffer of instructions available
- Perfect caches – 1 cycle latency for all instructions
- Unlimited Issue – Unlimited instructions issued per clock cycle

Limits to ILP HW Model Comparison

	Perfect Model	Power 5
Instructions Issued per Clock	Infinite	4
Instruction Window Size	Infinite	200
Renaming Registers	Infinite	48 integer + 40 Floating-point
Branch Prediction	Perfect	2% to 6% misprediction (Tournament Branch Predictor)
Cache	Perfect	64KI, 32KD, 1.92M L2, 36M L3
Memory Alias Analysis	Perfect	??

Paper: Limits of Instruction-Level Parallelism, David Wall, 1990

Upper Limit to ILP: Ideal Machine



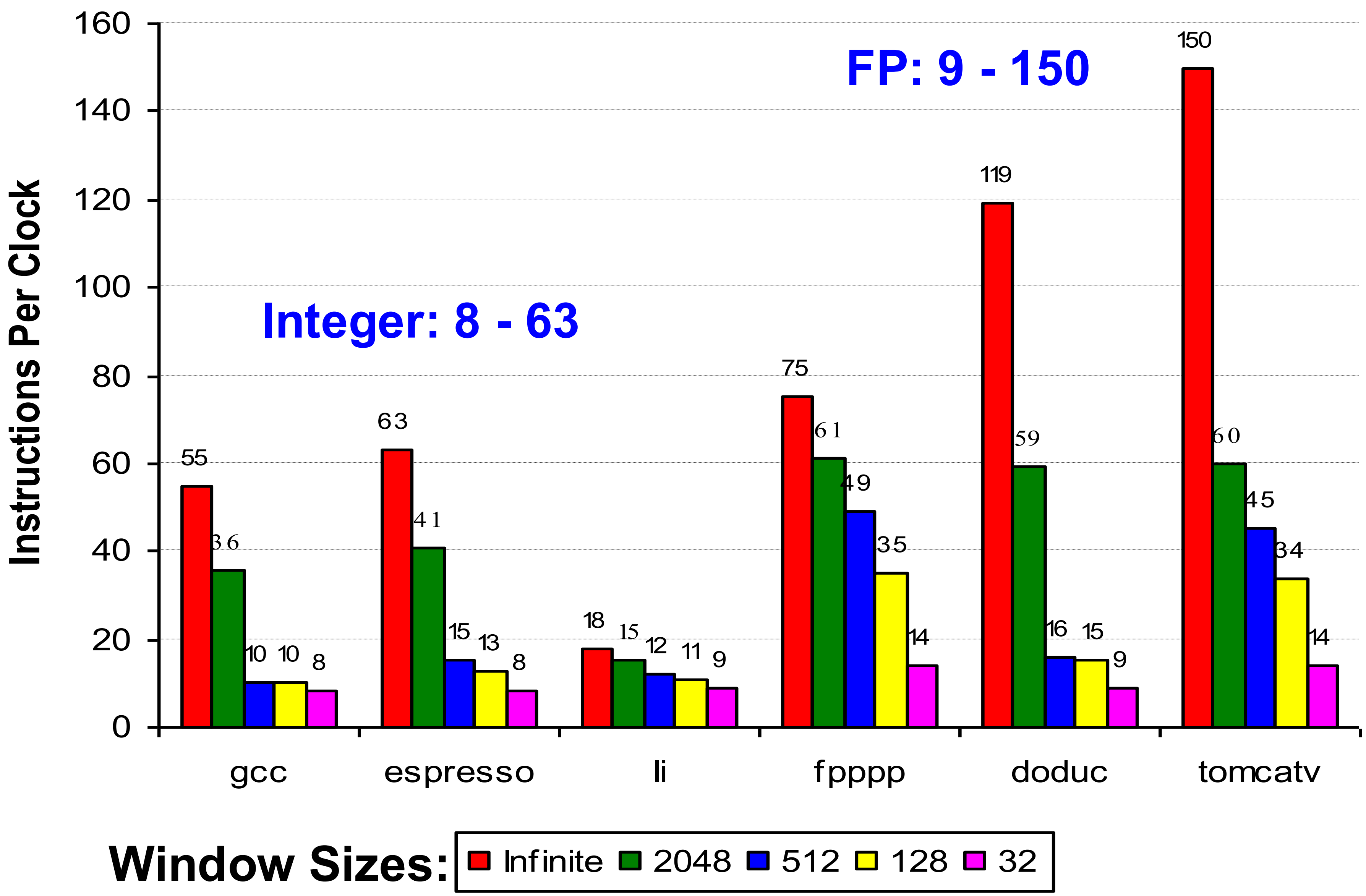
Practical Check: Maximum Issue Count

- How close a real dynamically scheduled, speculative processor come to the ideal one?
  - Look arbitrarily far ahead predicting all branches
  - Rename all register uses to avoid WAR/WAW
  - Determine data dependencies
  - Determine memory dependencies
  - Enough parallel unis
- Determine if n issuing instructions have any dependencies
  - **RAW:** $2n - 2 + 2n - 4 + \ldots + 2 = 2 \sum_{i=1}^{n-1} i = 2.n(n-1)/2 = n^2 - n$
  - **WAR:** $n^2 - n$
  - **WAW:** $(n-1) + (n-2) + \ldots + 1 = n(n-1)/2 = (n^2 - n)/2$
  - Issuing 50 instructions require **6125** (2450 + 2450 + 1225) comparisons

Limits to ILP HW Model Comparison

	New Model	Perfect Model	Power 5
Instructions Issued per Clock	Infinite	Infinite	4
Instruction Window Size	Infinite, 2K, 512, 128, 32	Infinite	200
Renaming Registers	Infinite	Infinite	48 integer + 40 Floating-point
Branch Prediction	Perfect	Perfect	2% to 6% misprediction (Tournament Branch Predictor)
Cache	Perfect	Perfect	64KI, 32KD, 1.92B L2, 36M L3
Memory Alias	Perfect	Perfect	??

More Realistic HW: Window Impact



Limits to ILP HW Model Comparison

	New Model	Perfect Model	Power 5
Instructions Issued per Clock	64	Infinite	4
Instruction Window Size	2048	Infinite	200
Renaming Registers	Infinite	Infinite	48 integer + 40 Floating-point
Branch Prediction	Perfect, 8K Tournament, 512 2-bit, profile, none	Perfect	2% to 6% misprediction (Tournament Branch Predictor)
Cache	Perfect	Perfect	64KI, 32KD, 1.92B L2, 36M L3
Memory Alias	Perfect	Perfect	??

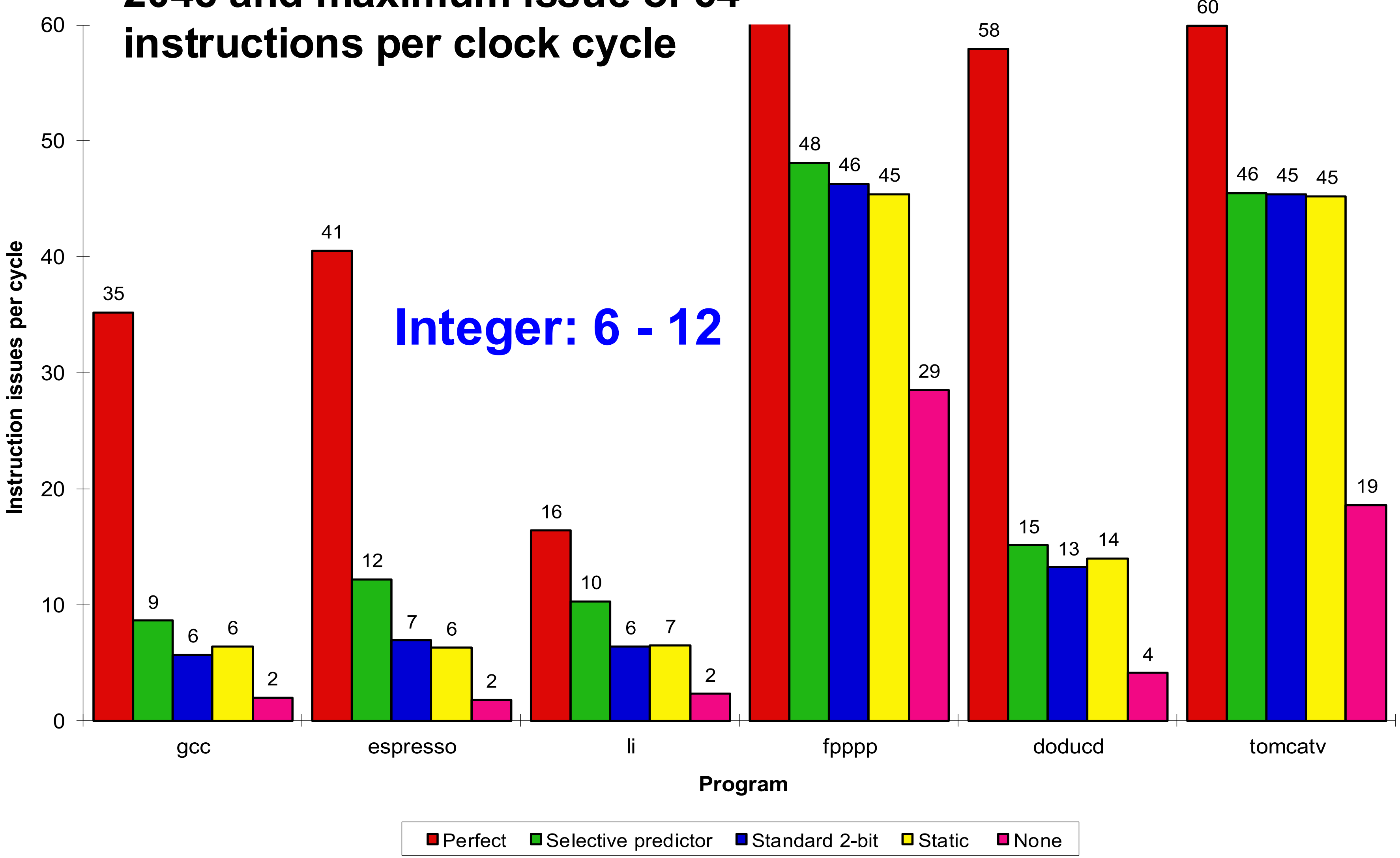
More Realistic HW: Branch Impact

Change from Infinite window to 2048 and maximum issue of 64 instructions per clock cycle

FP: 13 - 48

Integer: 6 - 12

IPC



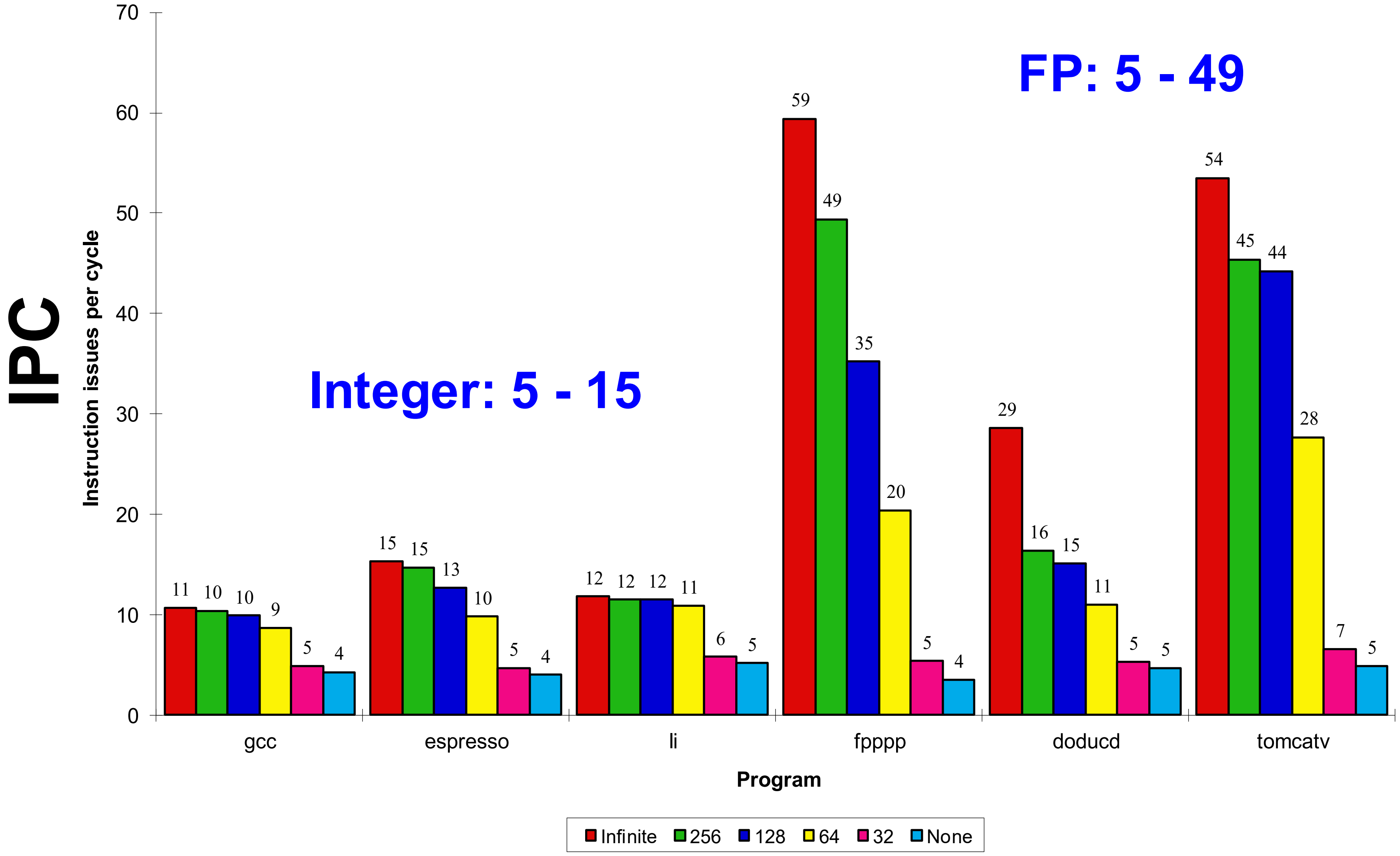
Perfect Tournament BHT (512) Profile No prediction

Limits to ILP HW Model Comparison

	New Model	Perfect Model	Power 5
Instructions Issued per Clock	64	Infinite	4
Instruction Window Size	2048	Infinite	200
Renaming Registers	Infinite, 256, 128, 64, 32, none	Infinite	48 integer + 40 Floating-point
Branch Prediction	8K 2-bit	Perfect	2% to 6% misprediction (Tournament Branch Predictor)
Cache	Perfect	Perfect	64KI, 32KD, 1.92B L2, 36M L3
Memory Alias	Perfect	Perfect	??

Renaming Register Impact (N int + N fp)

Change 2048 instr window, 64 instr issue, 8K 2 level prediction



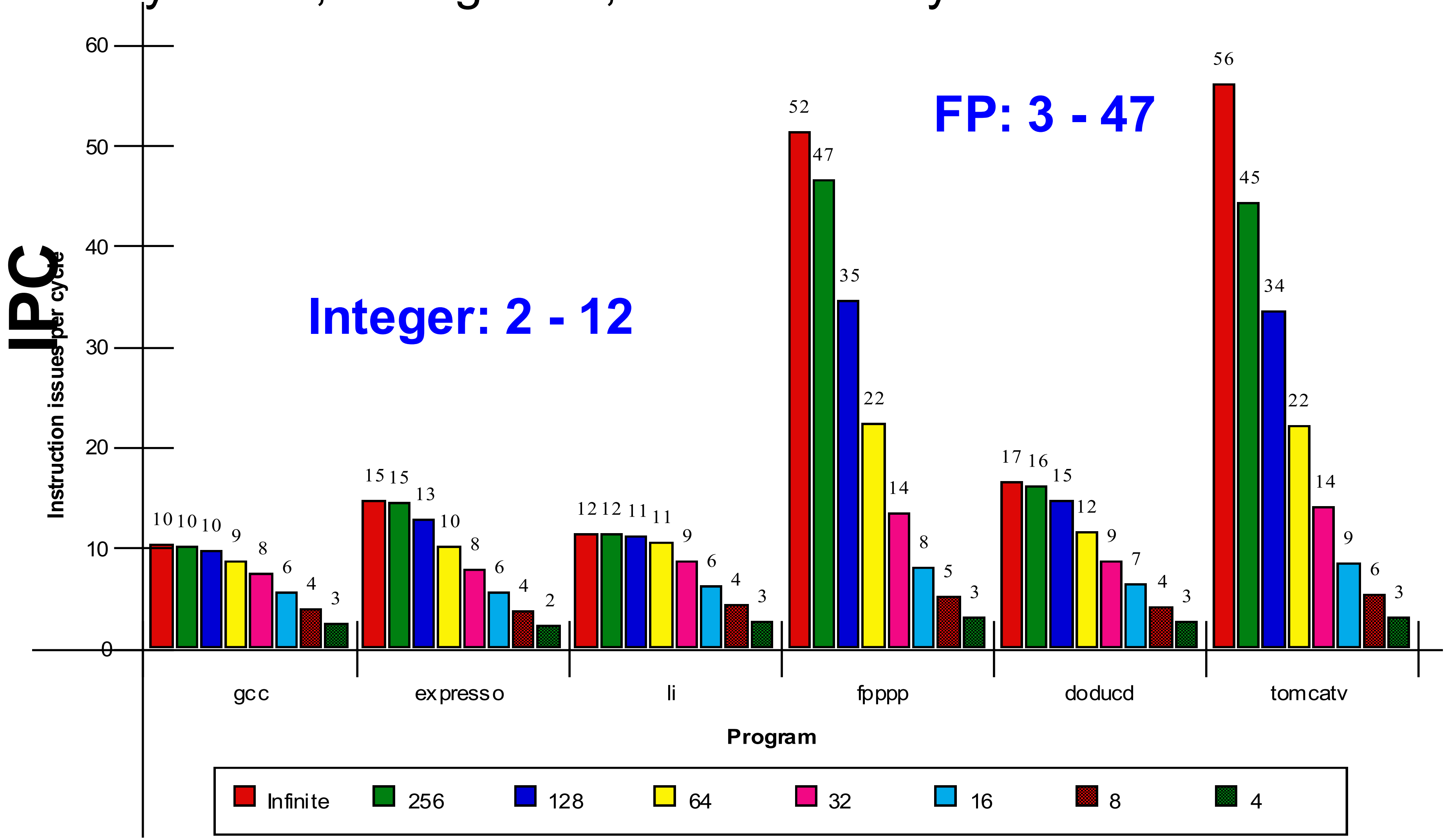
Infinite 256 128 64 32 None

Limits to ILP HW Model Comparison

	New Model	Perfect Model	Power 5
Instructions Issued per Clock	64	Infinite	4
Instruction Window Size	Infinite, 256, 128, 64, 32	Infinite	200
Renaming Registers	64 Int + 64 FP	Infinite	48 integer + 40 Floating-point
Branch Prediction	1K 2-bit	Perfect	2% to 6% misprediction (Tournament Branch Predictor)
Cache	Perfect	Perfect	64KI, 32KD, 1.92B L2, 36M L3
Memory Alias	HW dis-ambiguation	Perfect	??

Realistic HW: Window Impact

Perfect disambiguation (HW), 1K Selective Prediction, 16 entry return, 64 registers, issue as many as window



Integer: 2 - 12

FP: 3 - 47

How to Exceed ILP Limits?

- These are not laws of physics; just practical limits for today, and perhaps overcome via research
- Compiler & ISA advances can change results
- WAR and WAW hazards through memory: eliminated WAW and WAR hazards through register renaming, but not in memory usage
  - Can get conflicts via allocation of stack frames as a called procedure reuses the memory addresses of a previous frame on the stack
- Unnecessary dependencies e.g., loop variable can be removed by loop unrolling
- The data flow limit can be improved by value prediction

Thread-Level Parallelism and Multithreading

- Limitations of Instruction-Level Parallelism (ILP)
- Thread-Level Parallelism and Multithreading
  - Coarse-grained Multithreading
  - Fine-grained Multithreading
  - Simultaneous Multithreading
- Case Studies of Multithreaded Processors

Performance beyond Single Thread ILP

- There can be higher natural parallelism in some applications
  - Database or scientific applications
  - Explicit Data Level Parallelism or Thread Level Parallelism
- Data-Level Parallelism: Perform identical operations on data
  - We will discuss this later.
- Thread-Level Parallelism
  - Thread is a process with own instructions and data
  - Thread may be a process 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

Thread Level Parallelism (TLP)

- ILP exploits implicit parallel operations within a loop or straight-line code segment

- TLP are represented using multiple threads of execution that are inherently parallel

- Goal: Use multiple instruction streams to improve

  1. Throughput of computers that run many programs

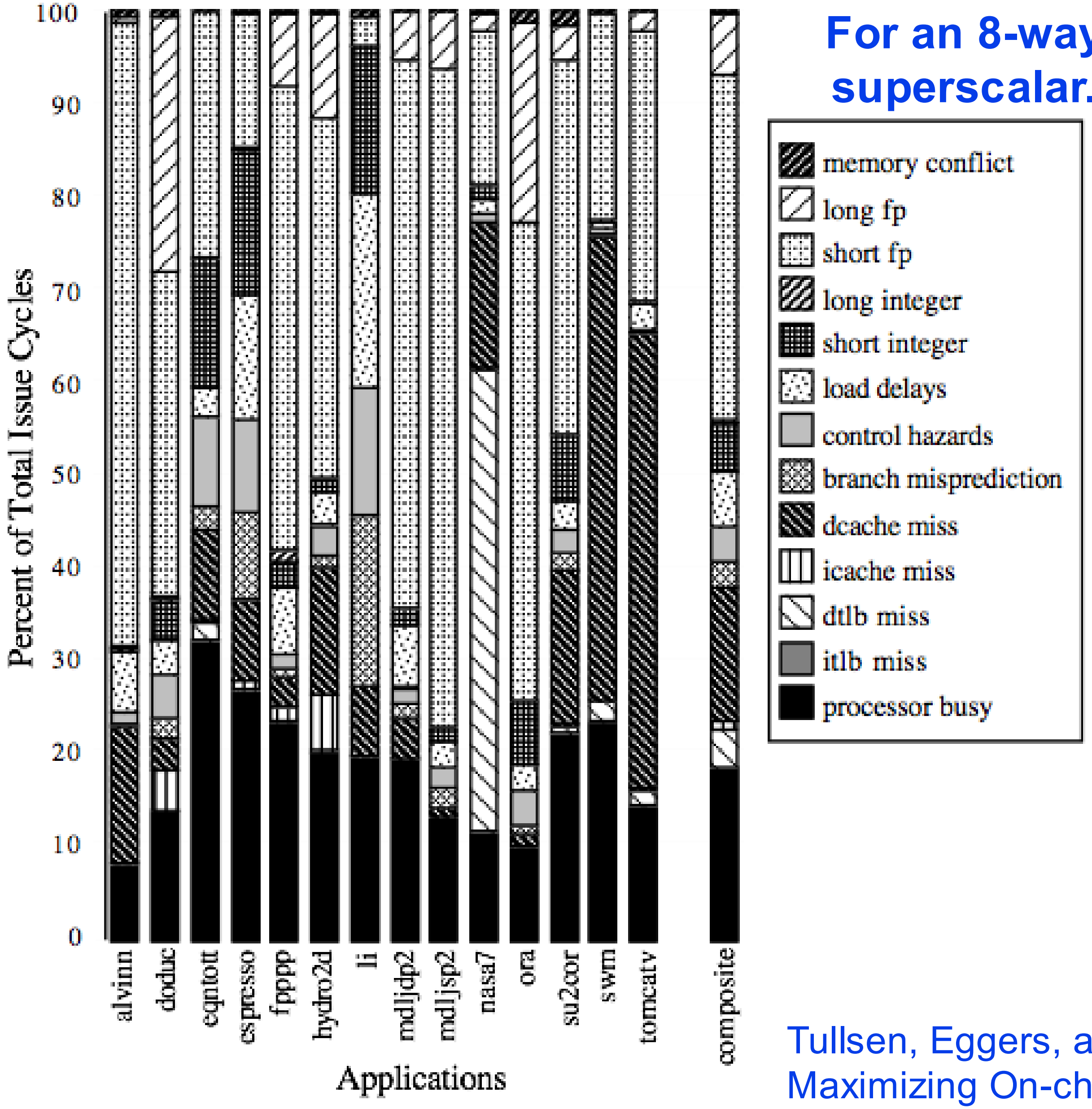
  2. Execution time of multi-threaded programs

- TLP is more cost-effective to exploit than ILP

Do both ILP and TLP?

- Functional units are often idle in data path designed for ILP because of either stalls or dependences
- TLP and ILP exploit two different kinds of parallel structure in a program
- Can a processor oriented at ILP exploit TLP?
- 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?

For Most Applications, Most Execution Units are Idle



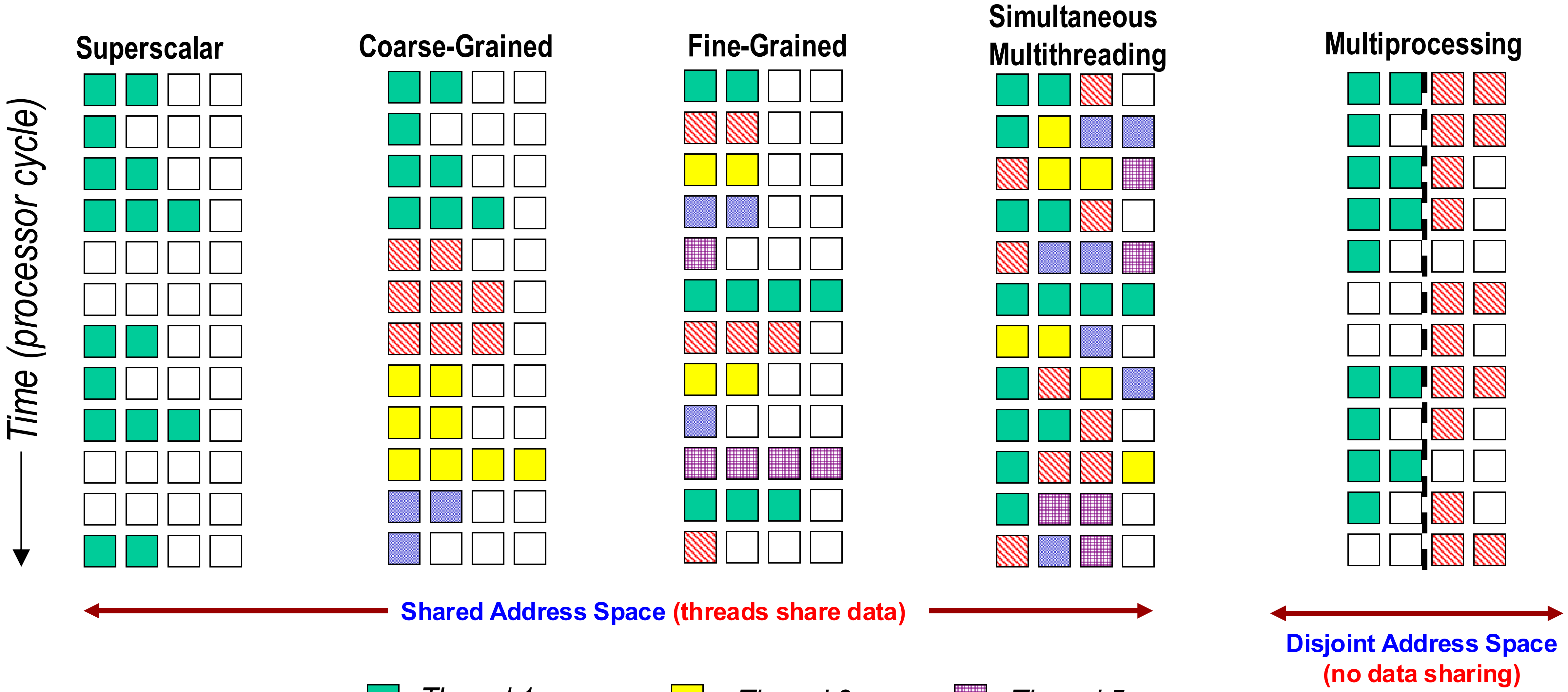
For an 8-way superscalar.

Tullsen, Eggers, and Levy, “Simultaneous Multithreading: Maximizing On-chip Parallelism”, ISCA 1995.

New Approach: Mulithreaded Execution

- Multithreading: multiple threads to share the functional units of one processor via overlapping
  - processor must duplicate independent state of each thread e.g., a separate copy of register file, a separate PC, and for running independent programs, a separate page table
  - memory shared through the virtual memory mechanisms, which already support multiple processes
  - HW for fast thread switch; much faster than full process switch ≈ 100s to 1000s of clocks
- When to switch?
  - Alternate instruction per thread (fine grain)
  - When a thread is stalled, perhaps for a cache miss, another thread can be executed (coarse grain)

Multithreaded Categories


Superscalar
Coarse-Grained
Fine-Grained
Simultaneous Multithreading
Multiprocessing
Time (processor cycle)
Shared Address Space (threads share data)
Disjoint Address Space (no data sharing)

🟩 *Thread 1*

▨ *Thread 2*

🟨 *Thread 3*

▦ *Thread 4*

▦ *Thread 5*

□ *Idle slot*

Coarse-Grained Multithreading

- Switches threads only on costly stalls, such as L2 cache miss
- Advantages
  - Relieves need to have very fast thread-switching
  - Doesn't slow down thread, since instructions from other threads issued only when the thread reaches a costly stall
- Disadvantage is hard to overcome throughput losses from shorter stalls, due to pipeline start-up costs
  - Since CPU issues instructions from 1 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 better for reducing penalty of high cost stalls, where pipeline refill << stall time
- Used in IBM AS/400

Fine-Grained Multithreading

- Switches between threads on each instruction, causing the execution of multiple threads to be interleaved
  - Usually done in a round-robin fashion, skipping any stalled threads
- CPU must be able to switch threads each clock
- Advantage – it can hide both short and long stalls, since instructions from other threads execute when one thread stalls
- Disadvantage – it slows down execution of individual threads, since a thread ready to execute without stalls will be delayed by instructions from other threads
- Used in Sun’s Niagara (will see later)

Simultaneous Multithreading (SMT)

- 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 hardware
- 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

Simultaneous Multi-Threading (SMT)

One thread, 8 units

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

Two threads, 8 units

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

M = Load/Store, FX = Fixed Point, FP = Floating Point, BR = Branch, CC = Condition Codes

Design Challenges in SMT

- Since SMT makes sense only with fine-grained implementation, impact of fine-grained scheduling on single thread performance.
  - A preferred thread approach sacrifices neither throughput nor single-thread performance.
  - Unfortunately, with a preferred thread, the processor is likely to sacrifice some throughput, when preferred thread stalls
- Larger register file needed to hold multiple contexts
- Not affecting 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

Thread-Level Parallelism and Multithreading

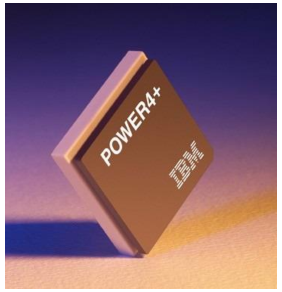
- Limitations of Instruction-Level Parallelism (ILP)
- Thread-Level Parallelism and Multithreading
- Case Studies of Multithreaded Processors
  - IBM Power 4 (one thread) versus Power 5 (two threads)
  - Fine-grained Multithreading on Sun T1 (no SMT)
  - Simultaneous Multithreading on Superscalar Processors
  - Putting It All Together: Intel Core i7 versus ARM Cortex-A53

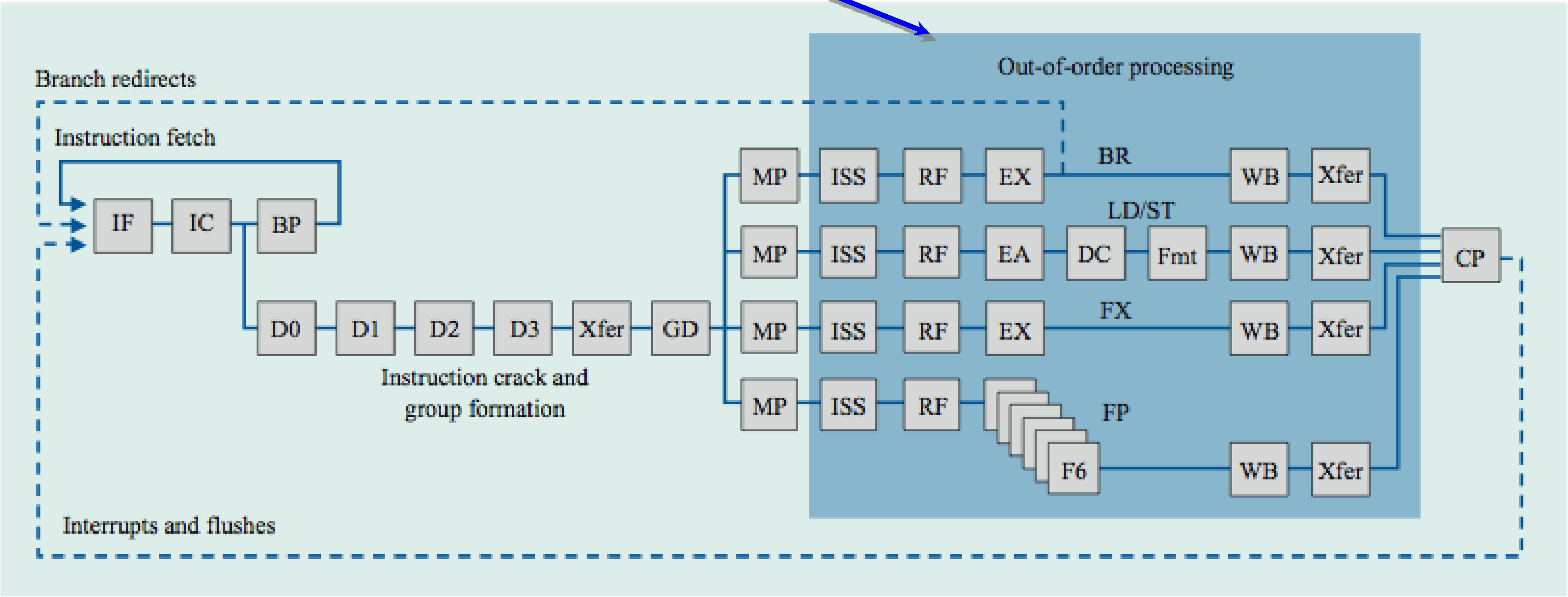
Five Generations of IBM Processors

	Power4	Power5	Power6	Power7	Power8
Introduced	2001	2004	2007	2010	2014
Initial clock rate (GHz)	1.3	1.9	4.7	3.6	3.3 GHz
Transistor count (M)	174	276	790	1200	4200
Issues per clock	5	5	7	6	8
Functional units per core	8	8	9	12	16
SMT threads per core	0	2	2	4	8
Cores/chip	2	2	2	8	12
SMT threads per core	0	2	2	4	8
Total on-chip cache (MiB)	1.5	2	4.1	32.3	103.0

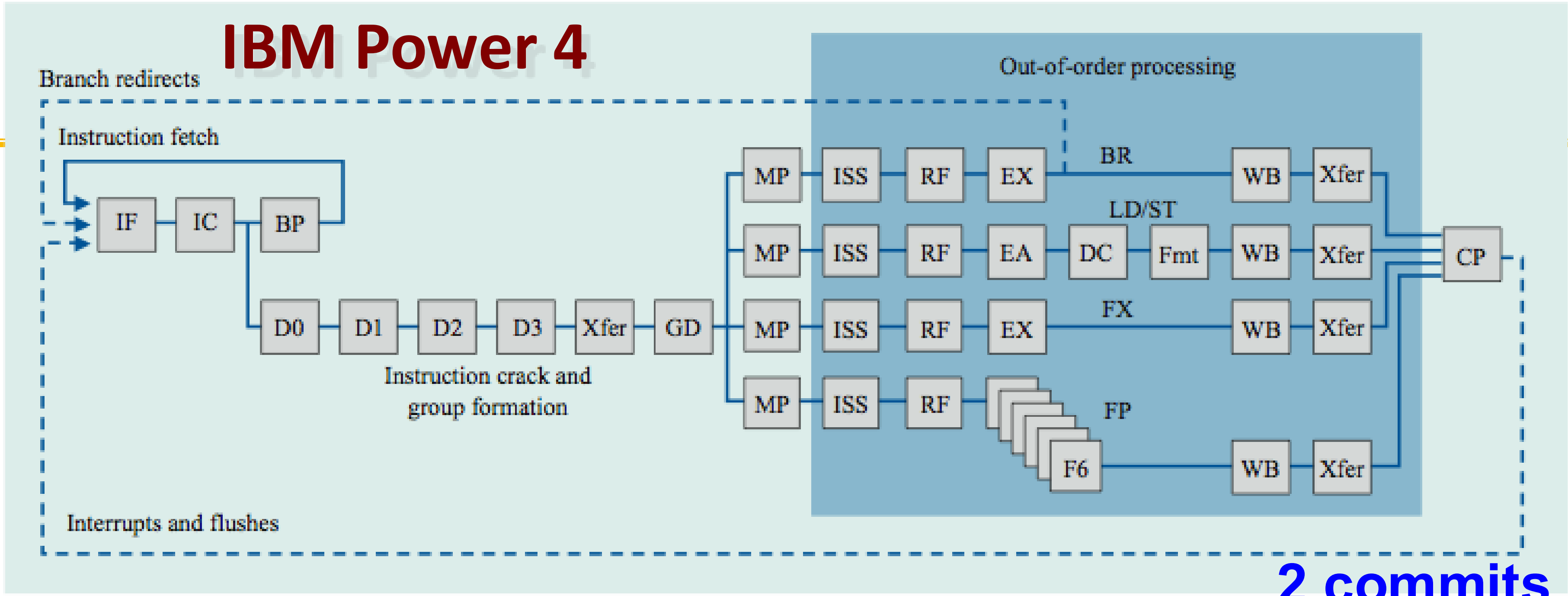
IBM Power 4

Single-threaded predecessor to Power 5. Eight execution units in out-of-order engine, each may issue an instruction each cycle.

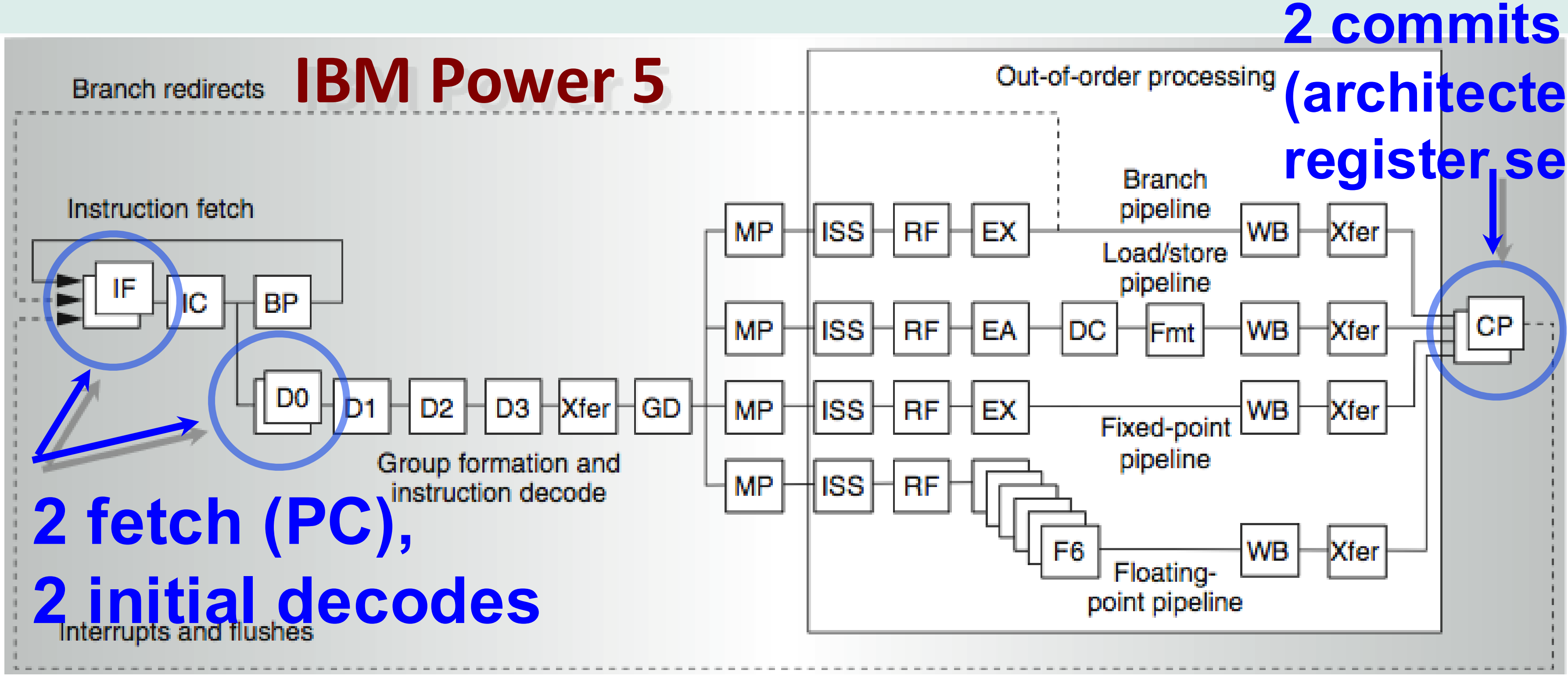




IBM Power 4



IBM Power 5



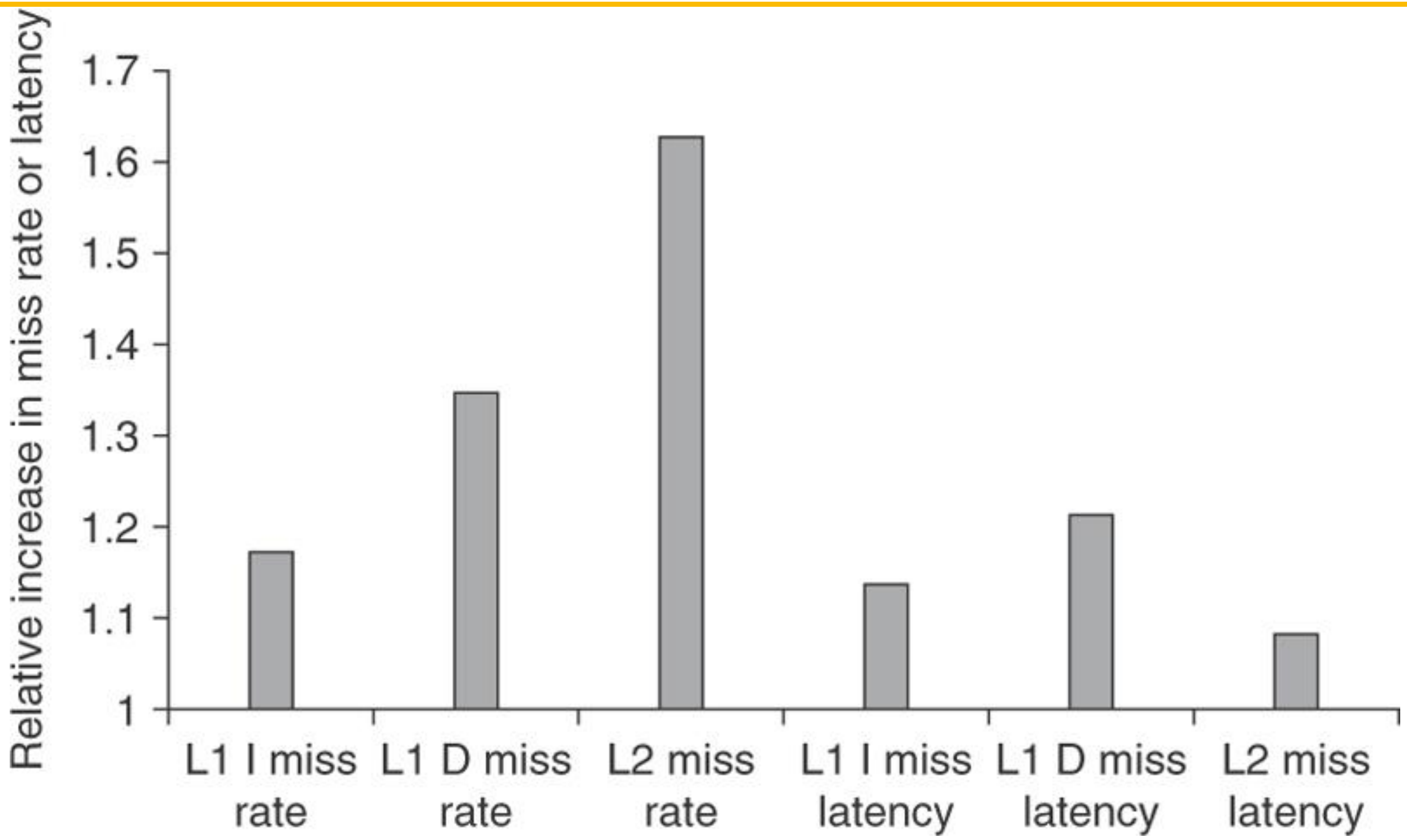
2 commits (architected register sets)

2 fetch (PC), 2 initial decodes

Thread-Level Parallelism and Multithreading

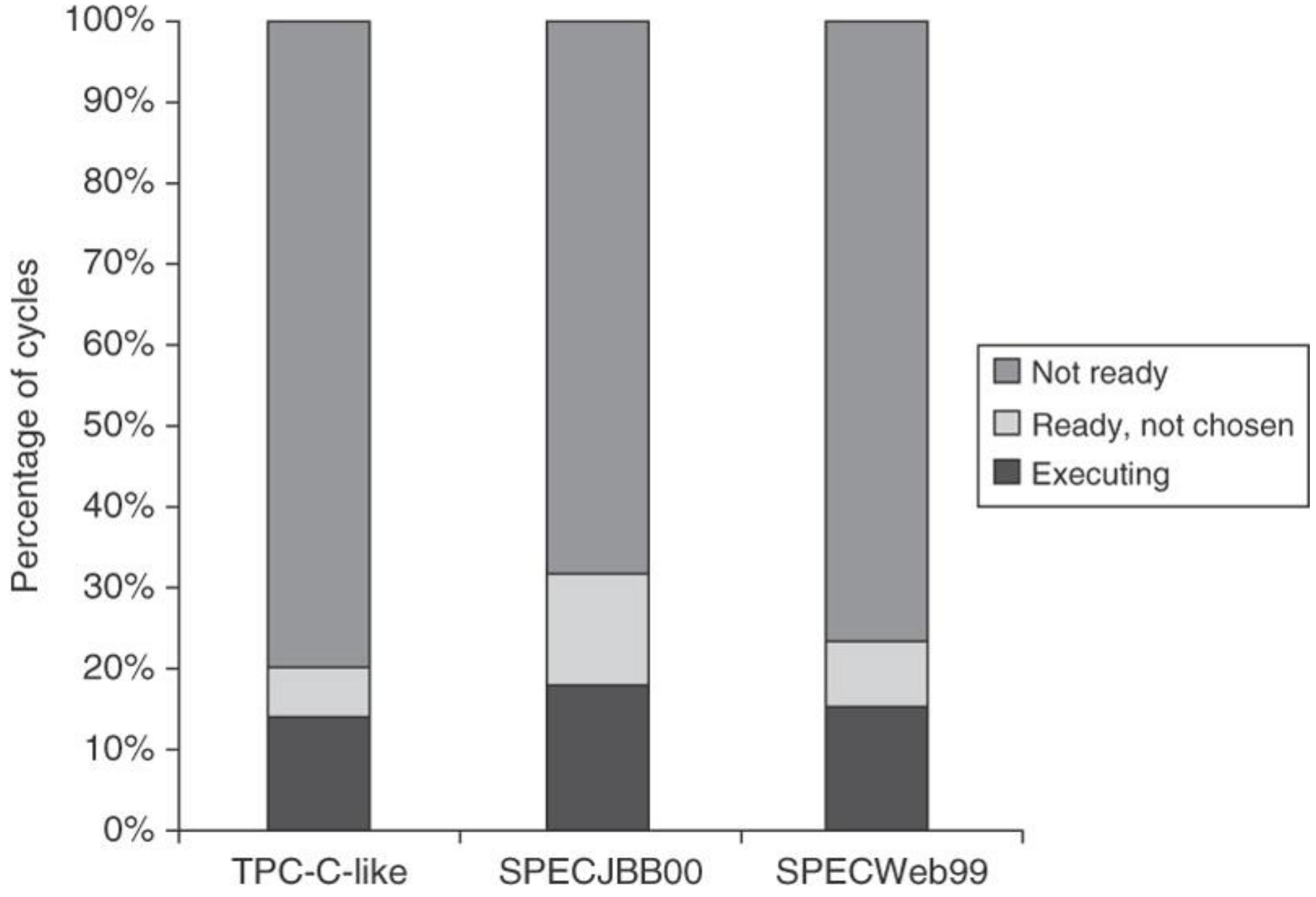
- Limitations of Instruction-Level Parallelism (ILP)
- Thread-Level Parallelism and Multithreading
- Case Studies of Multithreaded Processors
  - IBM Power 4 (one thread) versus Power 5 (two threads)
  - Fine-grained Multithreading on Sun T1 (no SMT)
  - Simultaneous Multithreading on Superscalar Processors
  - Putting It All Together: Intel Core i7 versus ARM Cortex-A53

Change in Miss Rates and Latencies



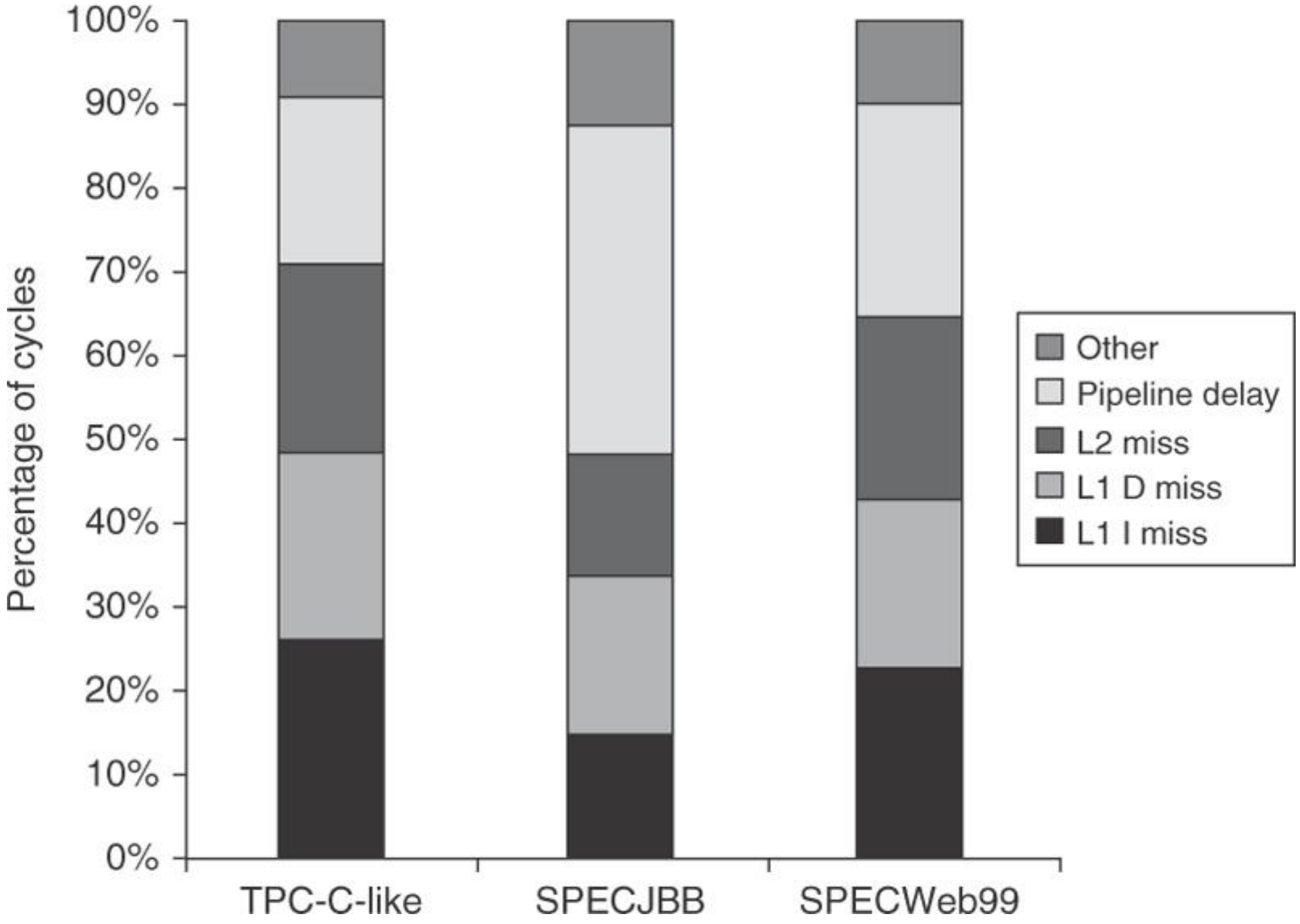
The relative change in the miss rates and miss latencies when executing with one thread per core versus four threads per core on the TPC-C benchmark on Sun UltraSPARC T1. The latencies are the actual time to return the requested data after a miss. In the four-thread case, the execution of other threads could potentially hide much of this latency.

Average Thread Statistics



“Executing” indicates the thread issues an instruction in that cycle. “Ready but not chosen” means it could issue but another thread has been chosen, and “not ready” indicates that the thread is awaiting the completion of an event. Three server benchmarks: TPC-C (online transaction processing), SPECJBB00 (SPEC Java Business Benchmark) and SPECWeb99.

Why a Thread is not Ready?

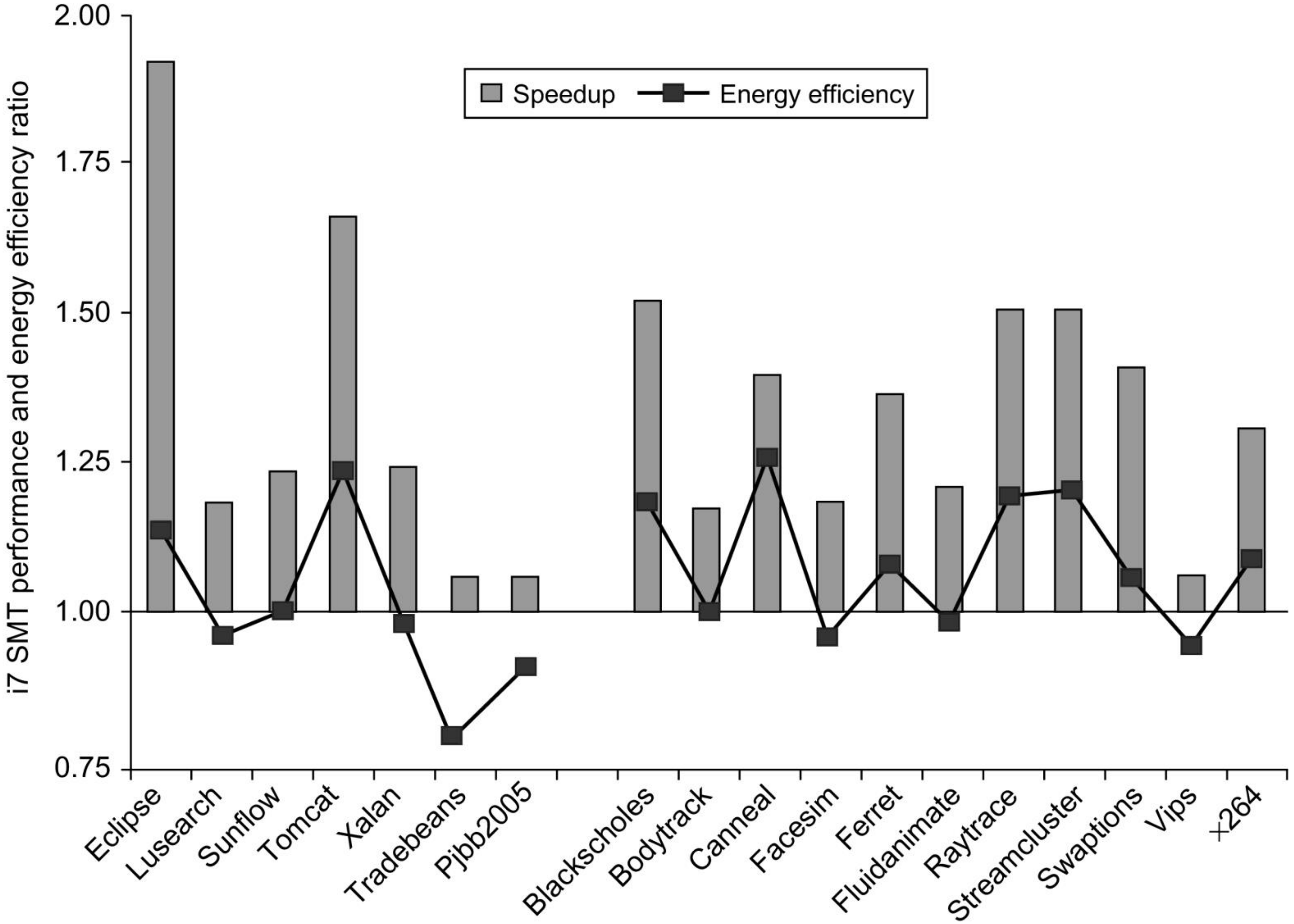


In TPC-C, store buffer full is the largest contributor (L1/L2 misses); in SPECJBB, atomic instructions are the largest contributor (pipeline delay); and in SPECWeb99, both factors contribute.

Instruction-Level Parallelism and Its Exploitation

- Limitations of Instruction-Level Parallelism (ILP)
- Thread-Level Parallelism and Multithreading
- Case Studies of Multithreaded Processors
  - IBM Power 4 (one thread) versus Power 5 (two threads)
  - Fine-grained Multithreading on Sun T1 (no SMT)
  - Simultaneous Multithreading on Superscalar Processors
  - Putting It All Together: Intel Core i7 versus ARM Cortex-A53

Speedup versus Energy Efficiency

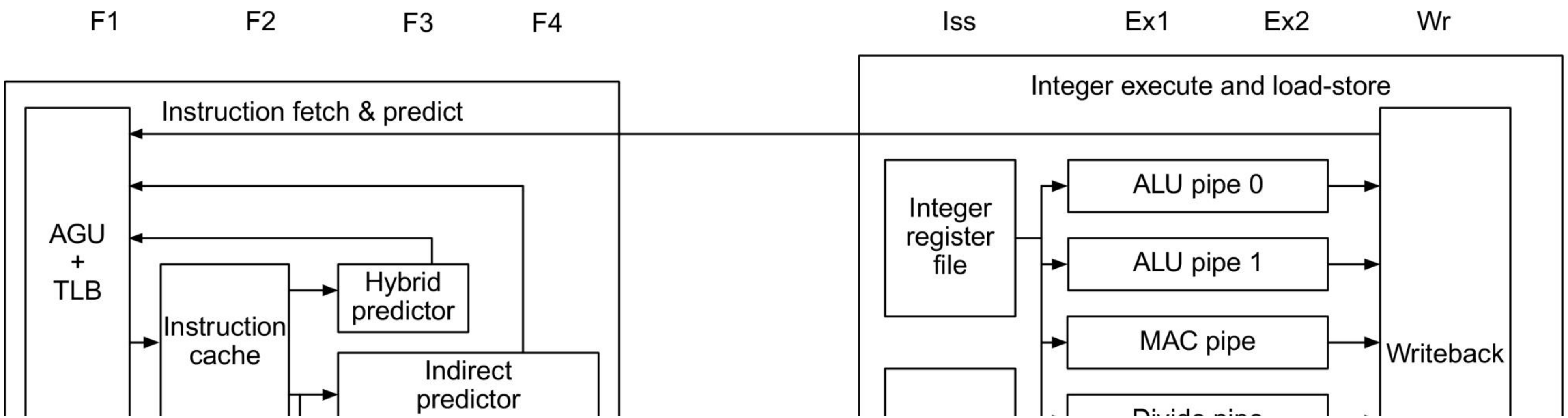


Speedup using multithreading (2 threads per core) on one core on **i7-920** averages 1.28 for the Java benchmarks and 1.31 for the PARSEC benchmarks. The energy efficiency averages 0.99 & 1.07, respectively. Anything above 1.0 for energy efficiency indicates that the feature reduces execution time by more than it increases average power.

Thread-Level Parallelism and Multithreading

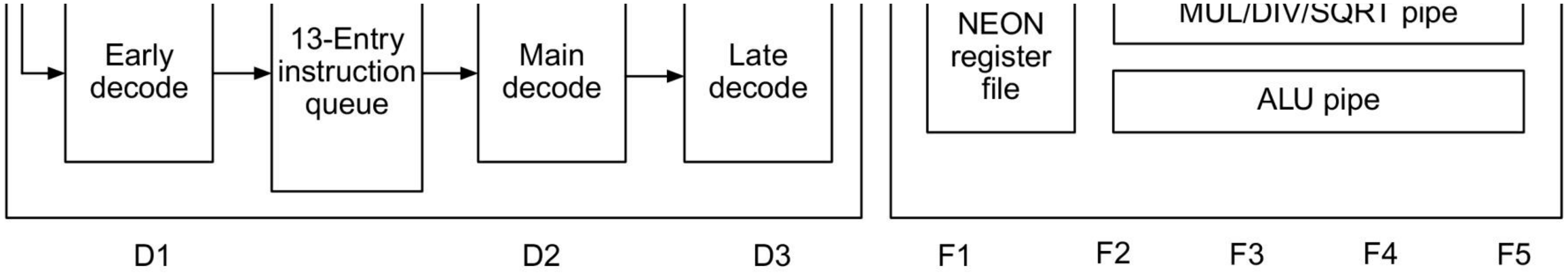
- Limitations of Instruction-Level Parallelism (ILP)
- Thread-Level Parallelism and Multithreading
- Case Studies of Multithreaded Processors
  - IBM Power 4 (one thread) versus Power 5 (two threads)
  - Fine-grained Multithreading on Sun T1 (no SMT)
  - Simultaneous Multithreading on Superscalar Processors
- Putting It All Together: Intel i7 versus ARM Cortex-A53

ARM Cortex-A53

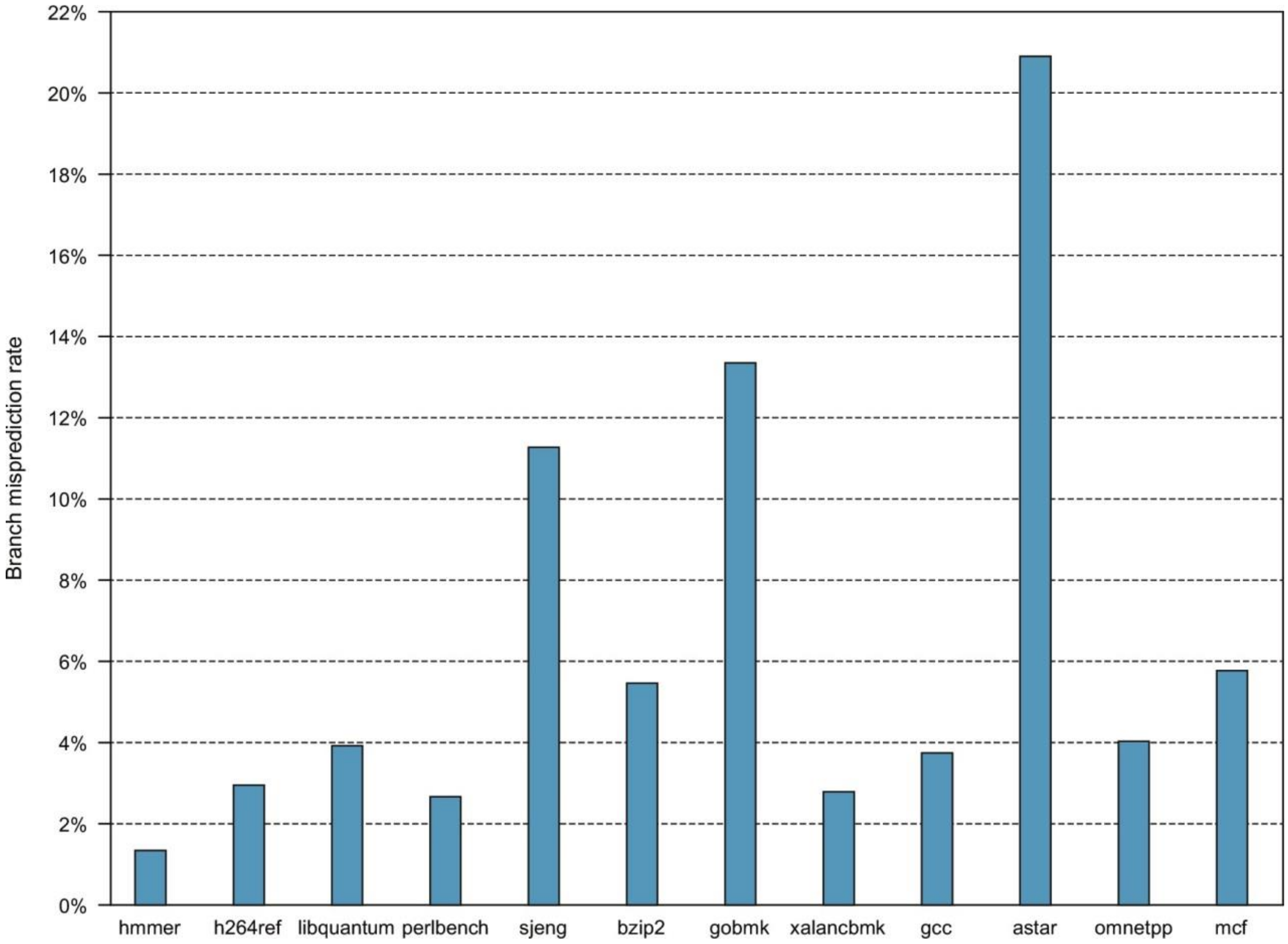


A53 integer pipeline has 8 stages and floating-point pipeline is 10 stages

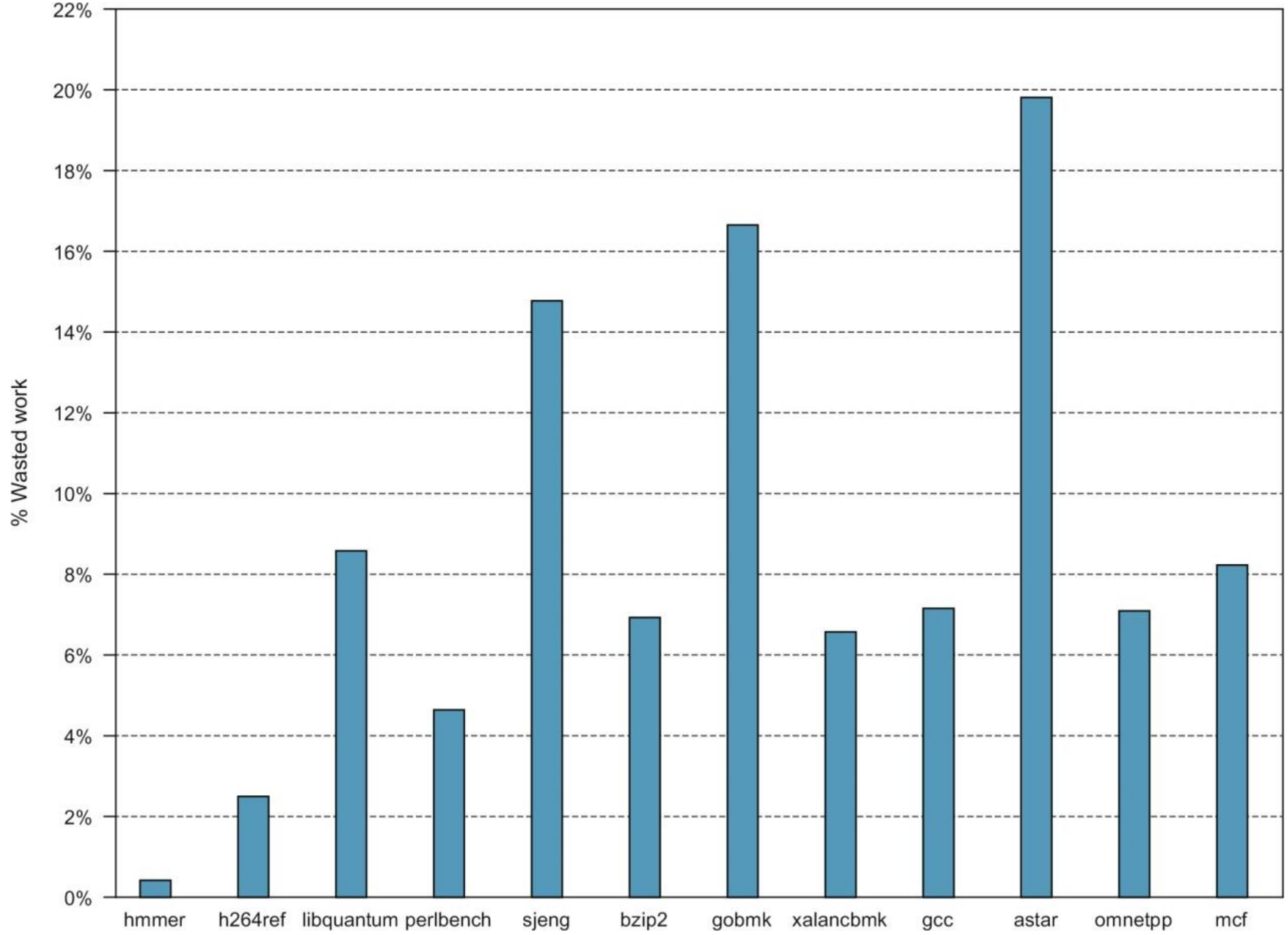
- F1 and F2 fetch the instruction
- D1 and D2 perform basic decoding, D3 for decoding complex instructions
- Issue (Iss), Ex1, Ex2, and Wr complete the integer pipeline
- Floating-point execution pipeline is 5 stages + 5 cycles for fetch and and decode (10 stages in total)



Misprediction Rate in A53 for SPECint2006

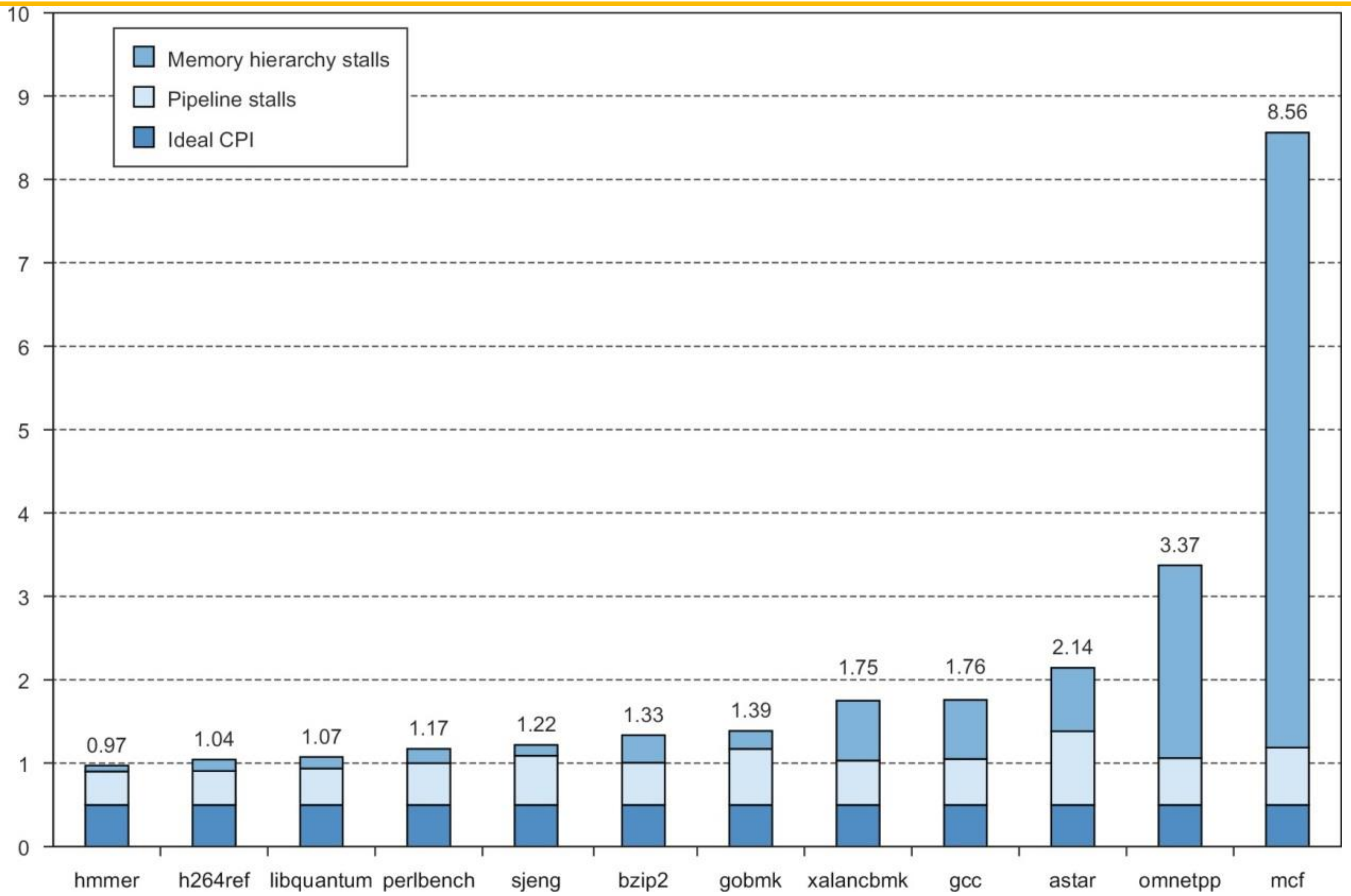


Wasted Work in A53



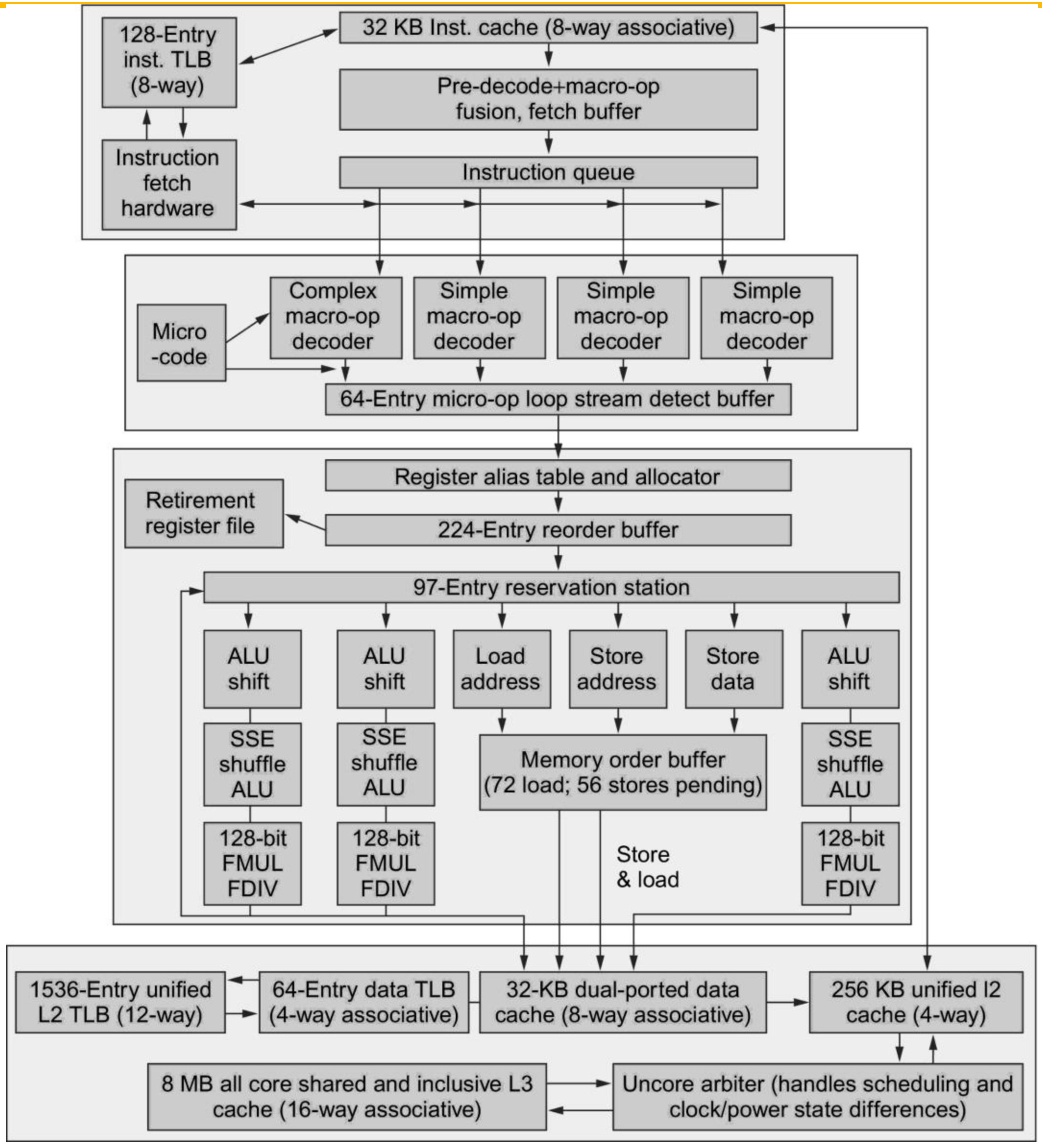
The wasted effort depends on various factors, including branch misprediction, data dependence, and cache misses.

CPI in A53



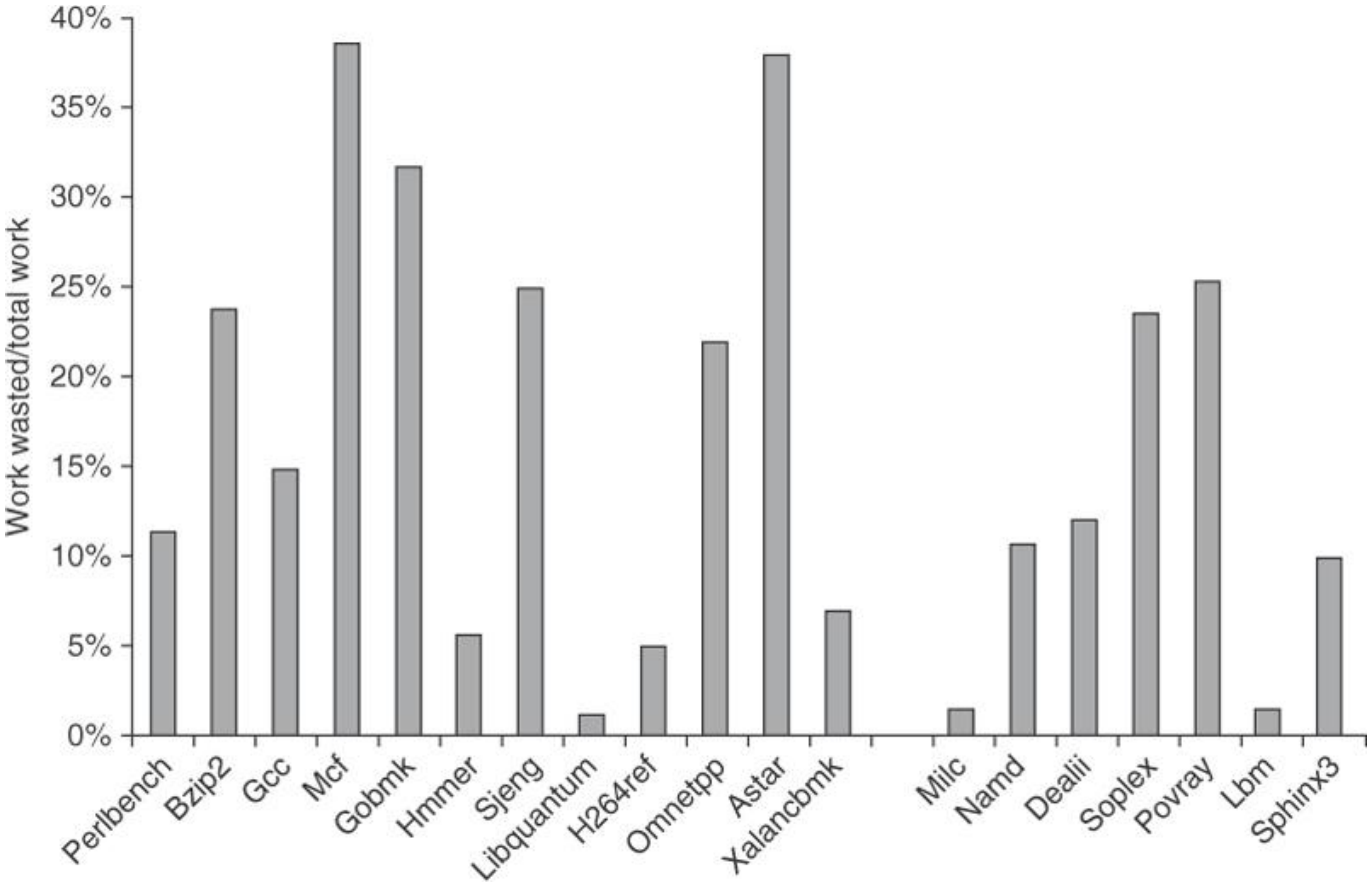
Ideal CPI is 0.5. Pipeline stalls are significant, but outweighed by cache misses in few cases.

Intel Core i7


128-Entry inst. TLB (8-way)
32 KB Inst. cache (8-way associative)
Pre-decode+macro-op fusion, fetch buffer
Instruction fetch hardware
Instruction queue
Micro -code
Complex macro-op decoder
Simple macro-op decoder
Simple macro-op decoder
Simple macro-op decoder
64-Entry micro-op loop stream detect buffer
Register alias table and allocator
Retirement register file
224-Entry reorder buffer
97-Entry reservation station
ALU shift
ALU shift
Load address
Store address
Store data
ALU shift
SSE shuffle ALU
SSE shuffle ALU
Memory order buffer (72 load; 56 stores pending)
SSE shuffle ALU
128-bit FMUL FDIV
128-bit FMUL FDIV
Store & load
128-bit FMUL FDIV
1536-Entry unified L2 TLB (12-way)
64-Entry data TLB (4-way associative)
32-KB dual-ported data cache (8-way associative)
256 KB unified l2 cache (4-way)
8 MB all core shared and inclusive L3 cache (16-way associative)
Uncore arbiter (handles scheduling and clock/power state differences)

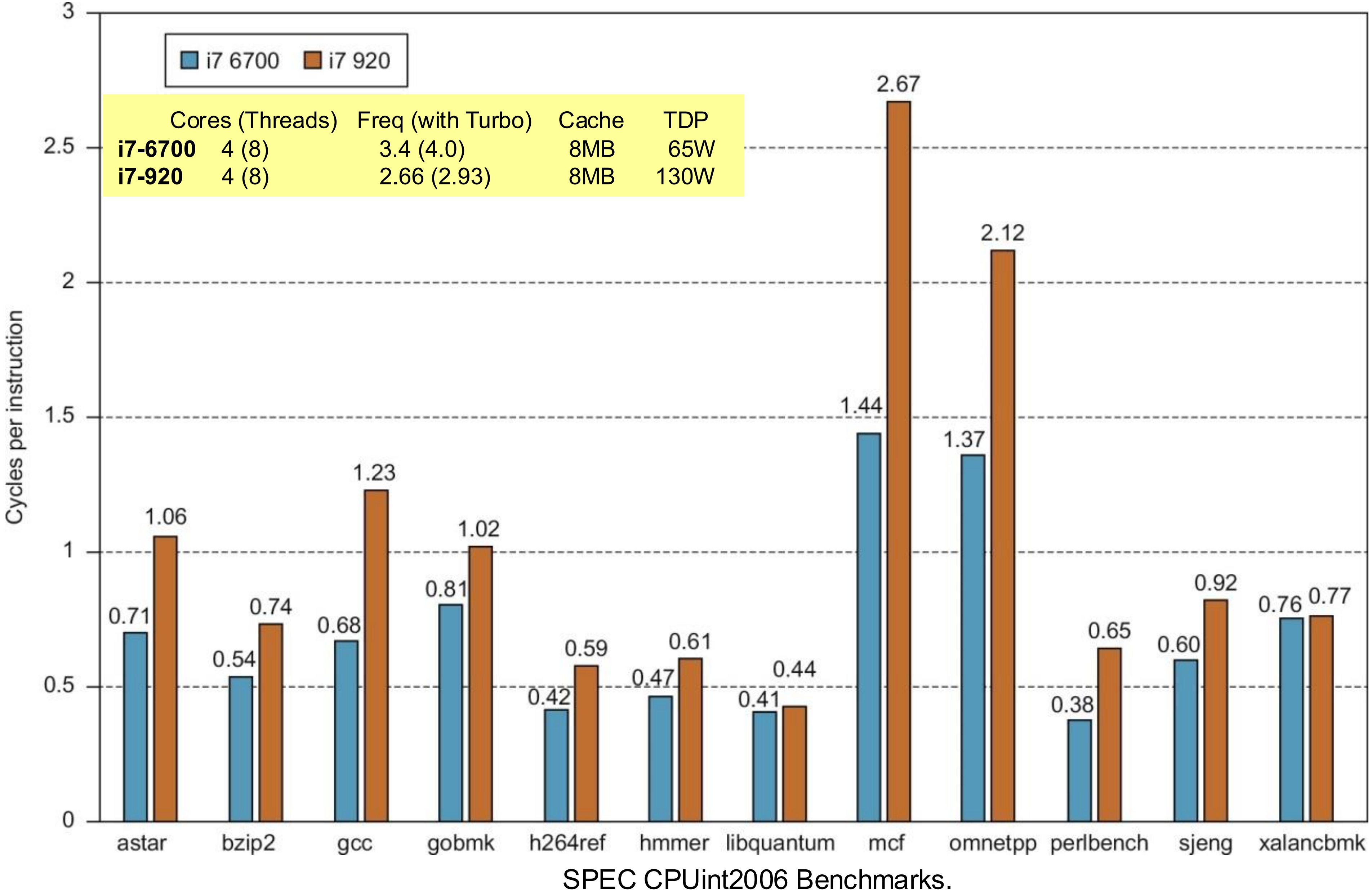
Pipeline depth is 14 stages, with branch mispredictions costing 17 cycles. There are 48 load and 32 store buffers. Six independent functional units can each begin execution of a ready micro-op in the same cycle.

Wasted Micro-Ops in i7



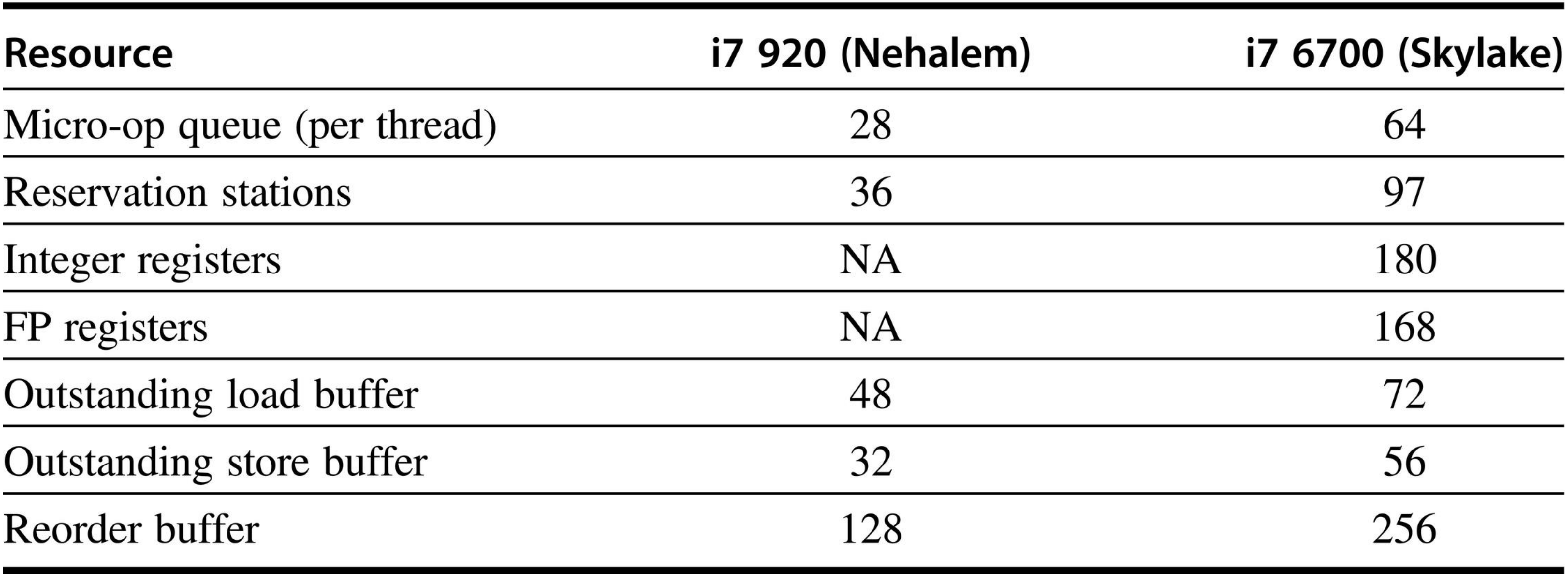
The amount of “**wasted work**” is plotted by taking the ratio of dispatched micro-ops that do not graduate to all dispatched micro-ops. For example, the ratio is 25% for *sjeng*, meaning that 25% of the dispatched and executed micro-ops are thrown away.

CPI of i7 6700 and i7 920



	Cores (Threads)	Freq (with Turbo)	Cache	TDP
i7-6700	4 (8)	3.4 (4.0)	8MB	65W
i7-920	4 (8)	2.66 (2.93)	8MB	130W

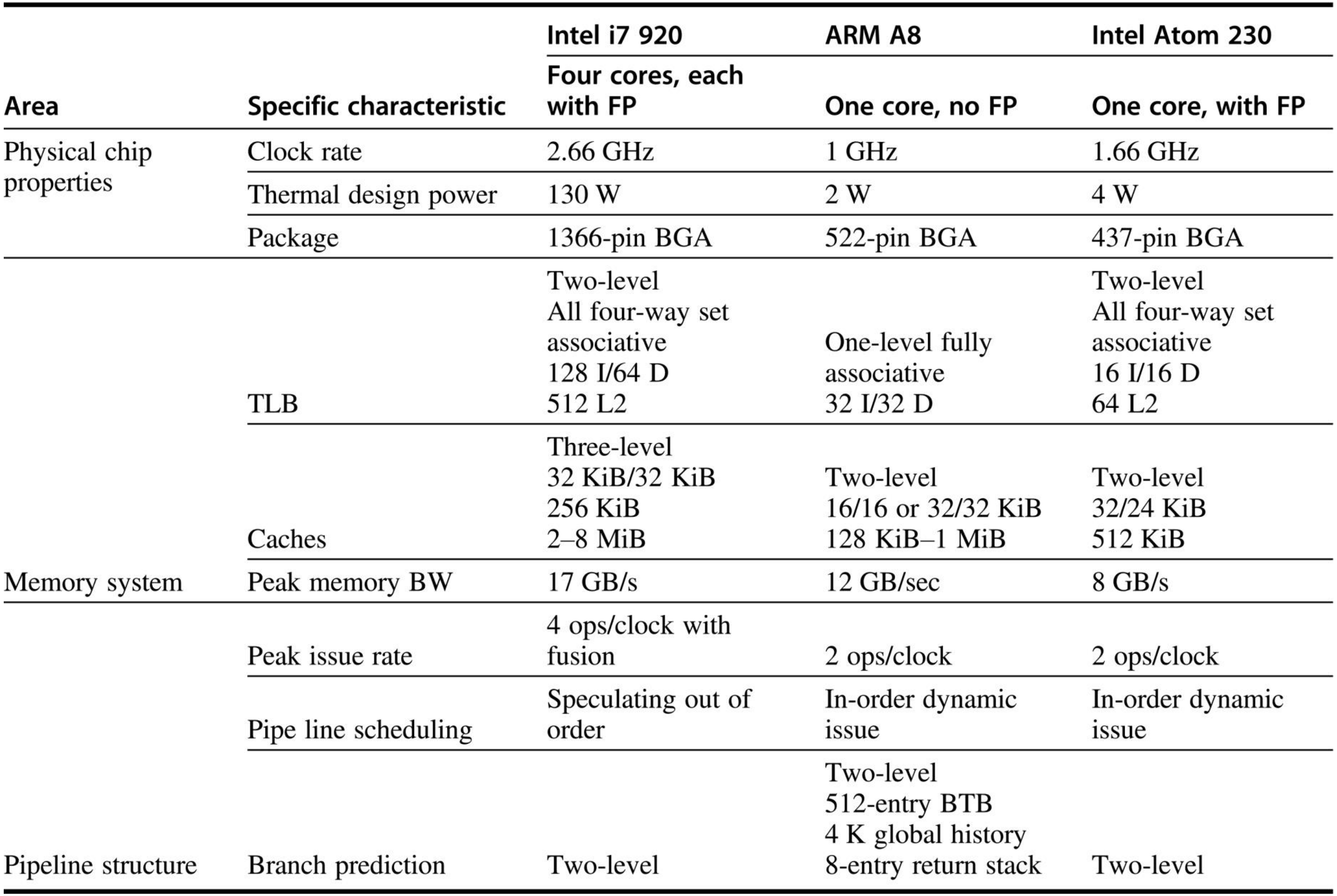
Intel i7 920 versus i7 6700



Resource	i7 920 (Nehalem)	i7 6700 (Skylake)
Micro-op queue (per thread)	28	64
Reservation stations	36	97
Integer registers	NA	180
FP registers	NA	168
Outstanding load buffer	48	72
Outstanding store buffer	32	56
Reorder buffer	128	256

Nehalem (i7 920) used a reservation station plus reorder buffer organization. In later microarchitectures, the reservation stations serve as scheduling resources, and register renaming is used rather than the reorder buffer. The reorder buffer in the **Skylake** (i7 6700) microarchitecture serves only to buffer control information. The choices of the size of various buffers and renaming registers, while appearing sometimes arbitrary, are likely based on extensive simulation.

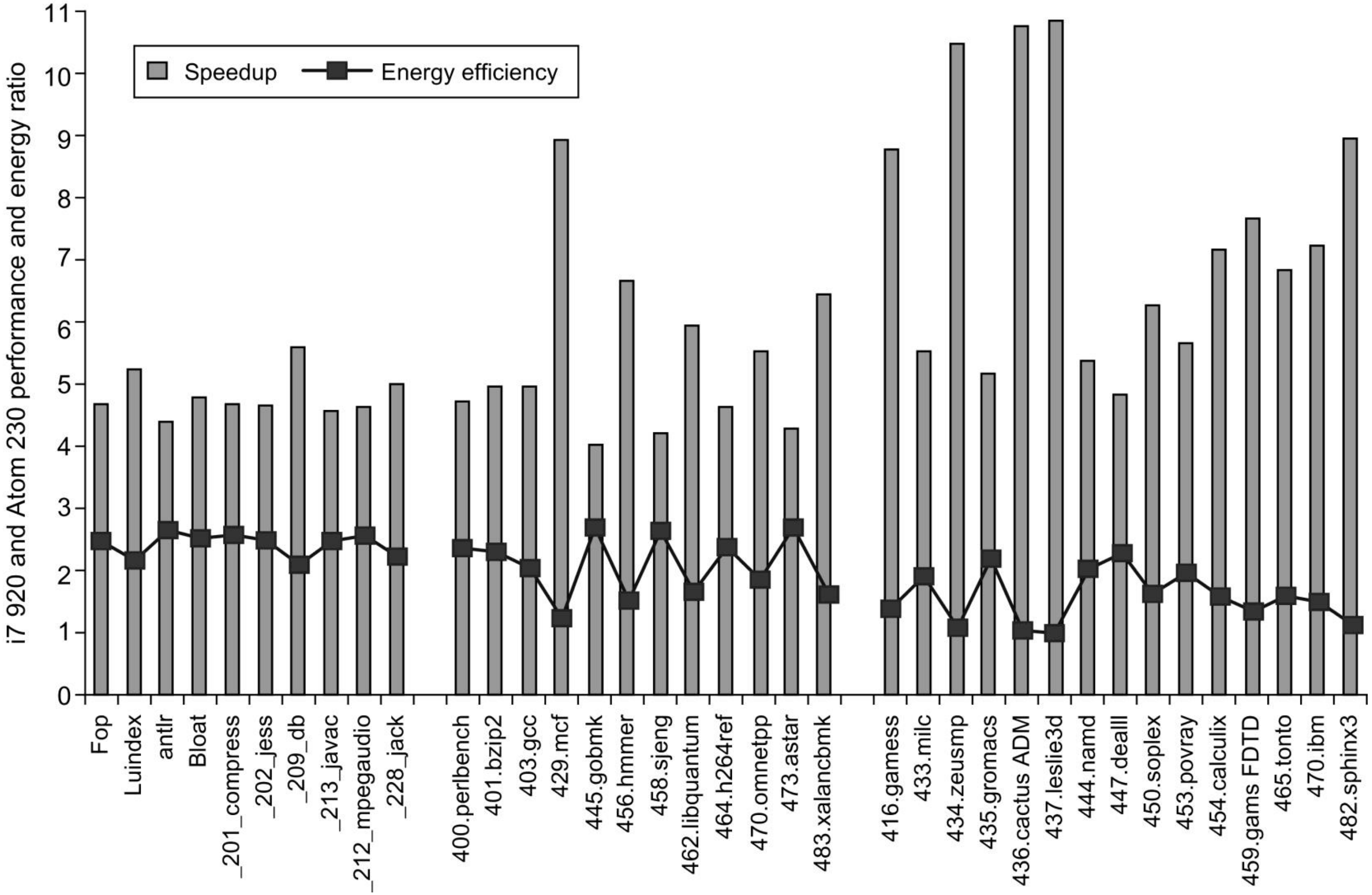
Intel i7 vs ARM A8 vs Atom 230



		Intel i7 920	ARM A8	Intel Atom 230
Area	**Specific characteristic**	**Four cores, each with FP**	**One core, no FP**	**One core, with FP**
Physical chip properties	Clock rate	2.66 GHz	1 GHz	1.66 GHz
	Thermal design power	130 W	2 W	4 W
	Package	1366-pin BGA	522-pin BGA	437-pin BGA
Memory system	TLB	Two-level All four-way set associative 128 I/64 D 512 L2	One-level fully associative 32 I/32 D	Two-level All four-way set associative 16 I/16 D 64 L2
	Caches	Three-level 32 KiB/32 KiB 256 KiB 2–8 MiB	Two-level 16/16 or 32/32 KiB 128 KiB–1 MiB	Two-level 32/24 KiB 512 KiB
	Peak memory BW	17 GB/s	12 GB/sec	8 GB/s
Pipeline structure	Peak issue rate	4 ops/clock with fusion	2 ops/clock	2 ops/clock
	Pipe line scheduling	Speculating out of order	In-order dynamic issue	In-order dynamic issue
	Branch prediction	Two-level	Two-level 512-entry BTB 4 K global history 8-entry return stack	Two-level

Design philosophy: i7 (desktop or servers), low-power mobile devices (A8) and notebook (Atom)

Performance vs. Energy Efficiency



i7 920 is 4 to over 10 times faster than the Atom 230 but that it is about 2 times *less* power efficient on average! One core is active on the i7 (single threaded), and the rest are in deep power saving mode. Turbo Boost is used on the i7, which increases its performance advantage but slightly decreases its relative energy efficiency.

Conclusion

- Instruction-level parallelism (ILP) has its limits
- Exploiting thread-level parallelism to go beyond ILP
  - Coarse-grained Multithreading
  - Fine-grained Multithreading
  - Simultaneous Multithreading
- Case studies of multithreaded processors
- Please read Section 3.10 – 3.13
- We review memory hierarchy next.