The Limitations of Instruction-Level Parallelism and Thread-Level Parallelism

Computer Architecture

Wei Wang

Text Book Chapters

- “Computer Architecture” Hennessy and Patterson, Chapter 3.10 “Studies of the Limitations of ILP”.

Road Map

The Limitations of ILP

Thread-level Parallelism

Acknowledgment

Instruction-Level Parallelism

- We have learned many CPU design techniques to optimize performance
  - Pipelining, superscalar, speculative execution, out-of-order execution
- The common goal is to execution instructions in parallel (pipelining), as many instructions as possible (superscalar, speculation and OoO).
- Instruction-level parallelism (ILP) is a measurement of how many of the instructions in a computer program can be executed in parallel.

The Limitations of ILP

- Applications (algorithms)
  - Different applications (algorithms) have different numbers of instructions that can run simultaneously.
- Compiler sophistication
  - Good compilers can generate and/or schedule instructions that run in parallel
  - E.g., VLIW compilers (in some degree)
- Hardware sophistication
  - Complex hardware usually can find more instructions to run
    - E.g., superscalar, speculation and OoO
    - E.g., SIMD instructions
- In this lecture, we focus on the hardware limitations and the application limitations.

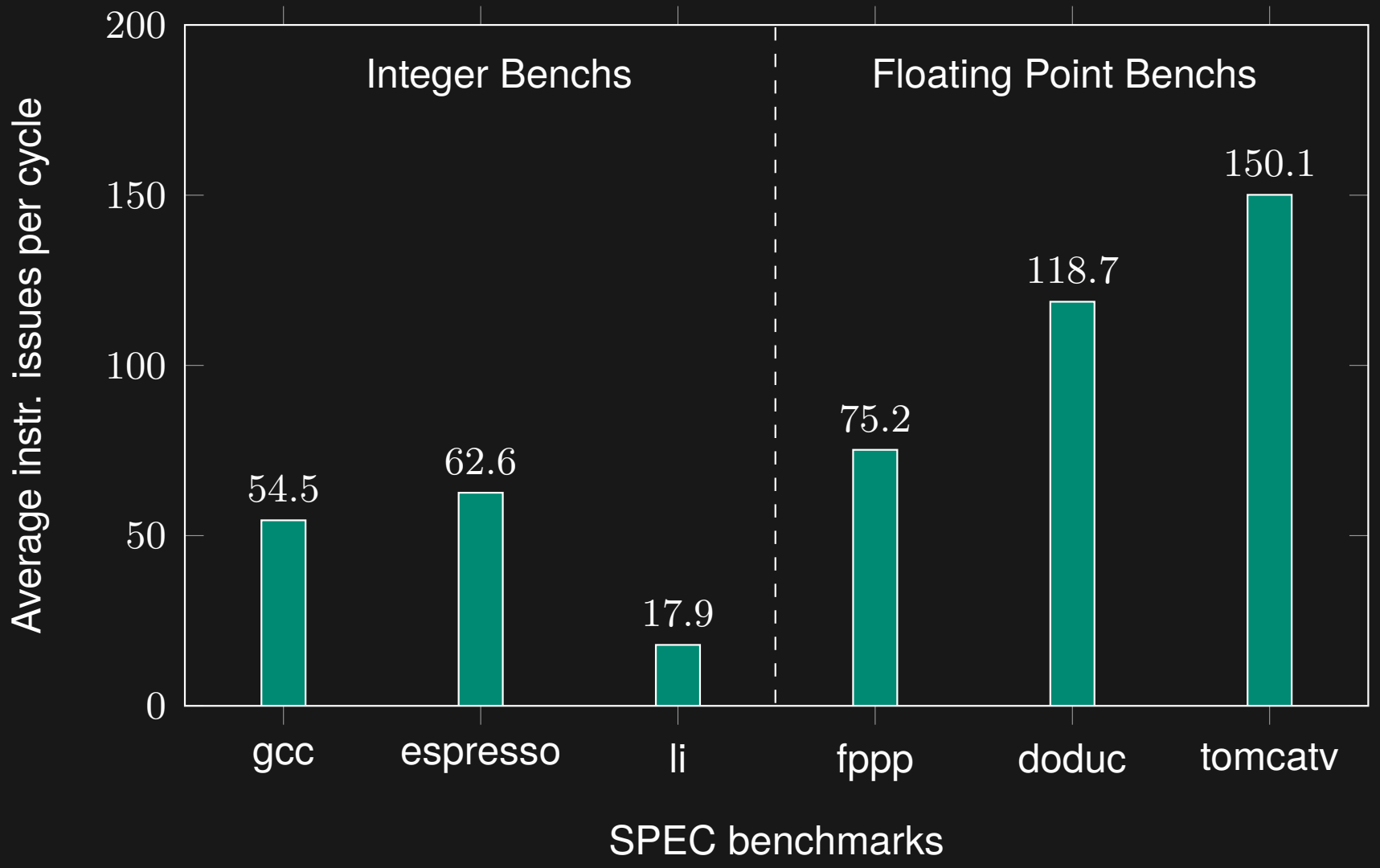
Hardware Limitations of ILP

- The number of registers for renaming
  - Essentially the size of the ROB
  - The more ROB entries the more instructions can be examined for parallel execution
- Branch (outcome and branch target) prediction accuracy
  - Better branch prediction => fewer stalls and pipeline flushes
- Memory address alias analysis (disambiguation)
  - Better memory aliasing analysis => more accurate dependencies detection => fewer pipeline flushes from incorrect speculation, fewer instructions stalled due to falsely/conservatively assumed dependencies
- Memory/cache latency
  - Faster memory/cache => fewer stalls due to slow memory accesses

The Perfect Hardware Model

- How much ILP can a perfect CPU find?
  - Infinite register rename (infinite ROB/RS) – all WAW/WAR hazards avoided, no structural hazards from ROB/RS, infinite number of instructions can be issued in parallel
  - Infinite functional units – infinite number of instructions can execute in parallel
  - Fast functional units – one cycle execution latency, no stalls due to RAW on slow computations
  - Perfect branch prediction – branch outcomes and targets are 100% accurately predicted
  - Perfect memory address alias analysis – all memory addresses are known
  - Perfect memory/cache – all memory accesses take one cycle.
  - Only true (RAW) dependencies are left (that limits ILP).
- Impossible in practice
  - But can be simulated using past execution traces.

Upper Limit of ILP with the Perfect CPU



Upper Limit of ILP with the Perfect CPU cont'd

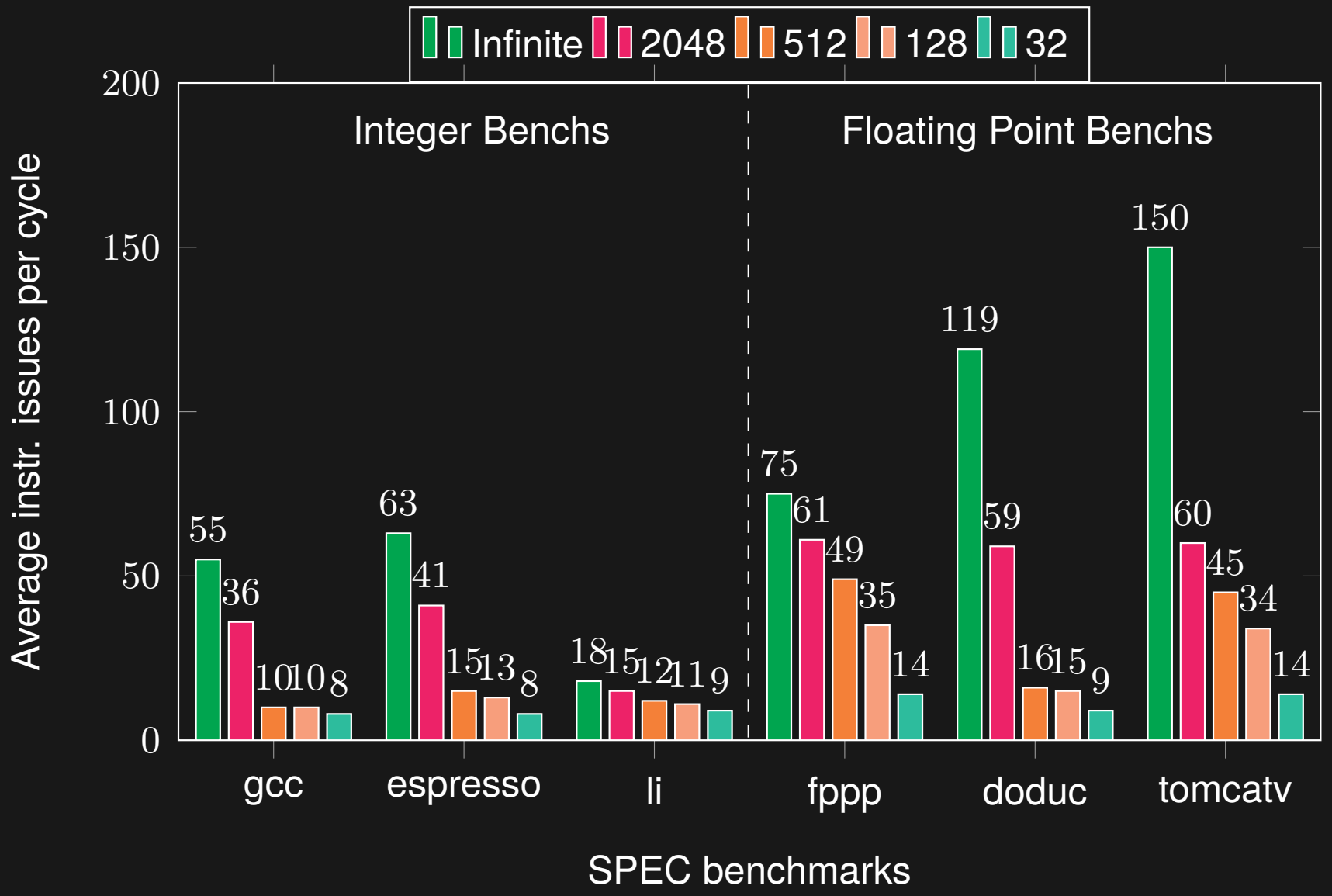
- The maximum ILP is fundamentally limited by the RAW dependencies
  - Cannot issue more instructions if previous computations are not finished
  - RAW dependencies reflect the ILP limitations imposed by the problems, the algorithms, the programs and/or the compiler code generation.
- Floating point benchmarks have higher max ILP
  - Would also benefit from SIMD
  - Highest (so far observed) is 500 from *swm256* (Wall. 1993)
- Integer benchmarks have lower max ILP mostly due to their step-by-step behaviors.

Realistic Instruction Windows

- Realistic CPUs do not have unlimited ROB, so their instruction window is typically less than 500.
- What is the max ILP if we reduce the instruction windows?

	Perfect Model	Current Model
Max Issues per Cycle	Infinite	Infinite
Instruction Window Size	Infinite	2K, 512, 128, 32
Renaming Regisers	Infinite	Infinite
Branch Prediction	Perfect	Perfect
Memory Alias	Perfect	Perfect
Memory/ Cache	Perfect	Perfect

Realistic Instruction Windows cont'd

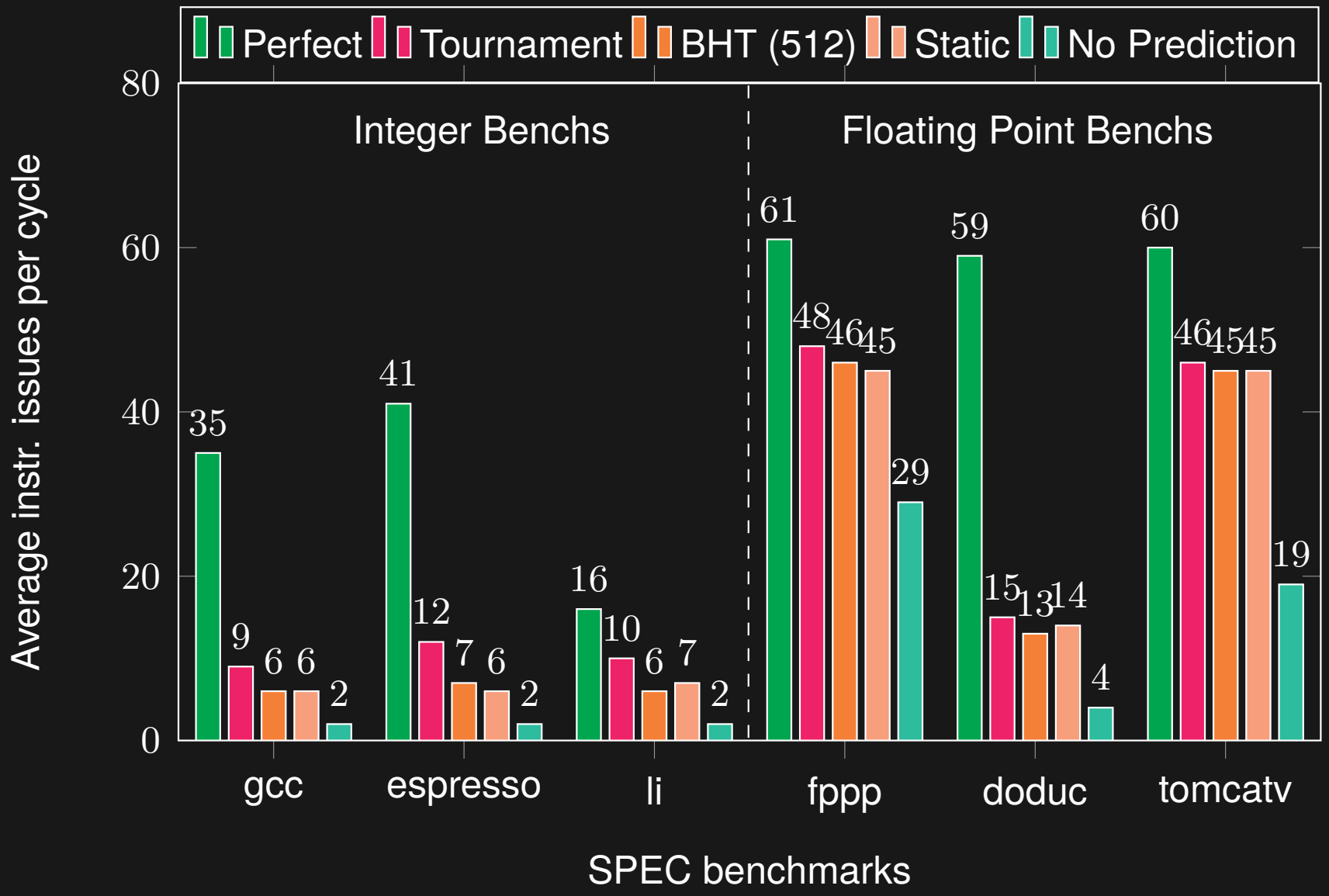


Realistic Branch Predictors

- Realistic CPUs do not have perfect branch predictors.
- What is the max ILP if we use Tournament, 2-bit saturate counters and no predictions?

	Perfect Model	Current Model
Max Issues per Cycle	Infinite	64
Instruction Window Size	Infinite	2048
Renaming Regisers	Infinite	Infinite
Branch Prediction	Perfect	8K Tournament, 512 2-bit, Compiler Static, none
Memory Alias	Perfect	Perfect
Memory/ Cache	Perfect	Perfect

Realistic Branch Predictors cont'd

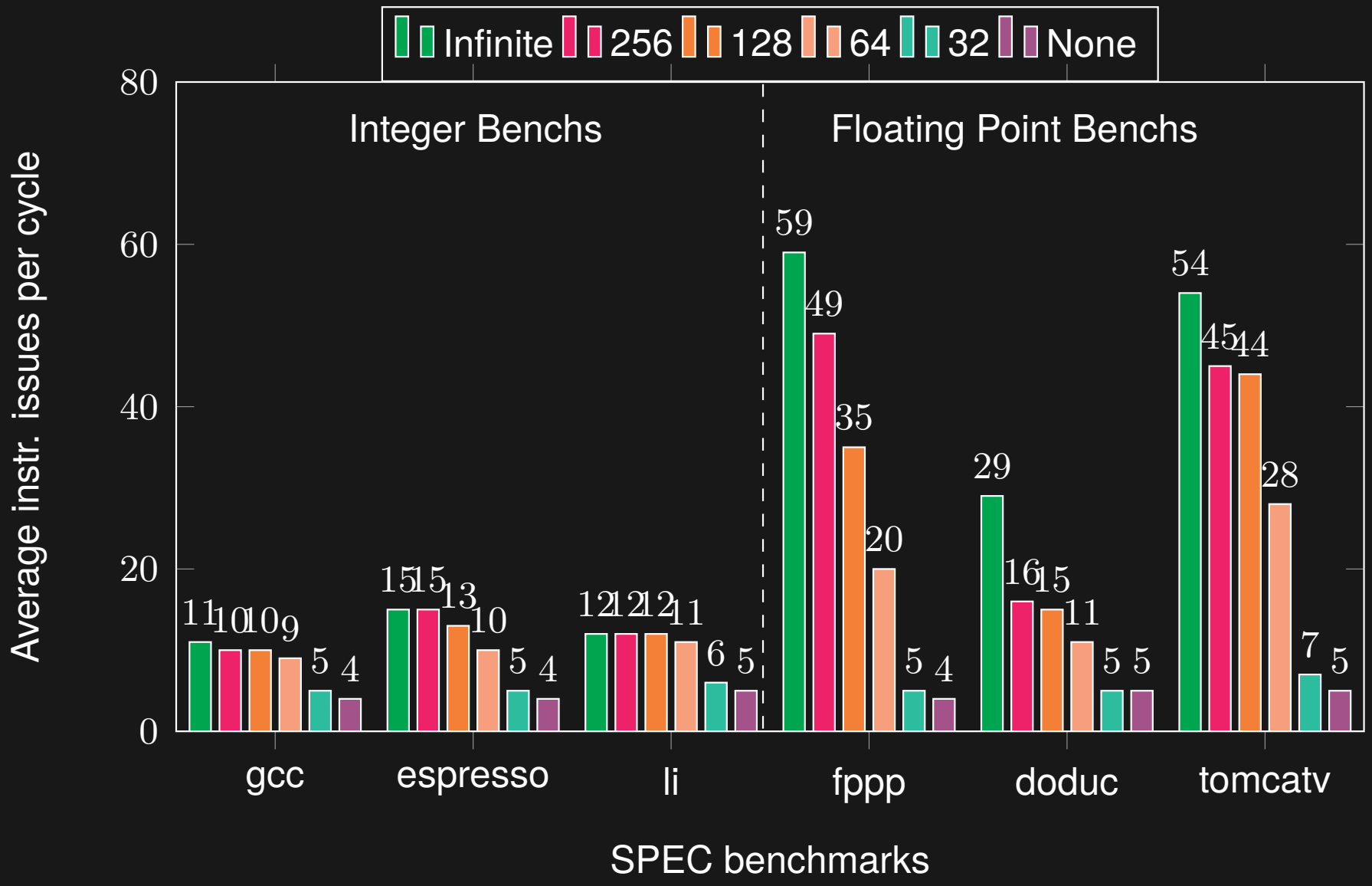


Realistic Renaming Registers

- Realistic CPUs do not have infinite registers (i.e, no infinite reservation stations).
- What is the max ILP if we have fewer registers?

	Perfect Model	Current Model
Max Issues per Cycle	Infinite	64
Instruction Window Size	Infinite	2048
Renaming Regisers	Infinite	256, 128, 64, 32, None
Branch Prediction	Perfect	8K 2-Bit counter
Memory Alias	Perfect	Perfect
Memory/ Cache	Perfect	Perfect

Realistic Renaming Registers cont'd

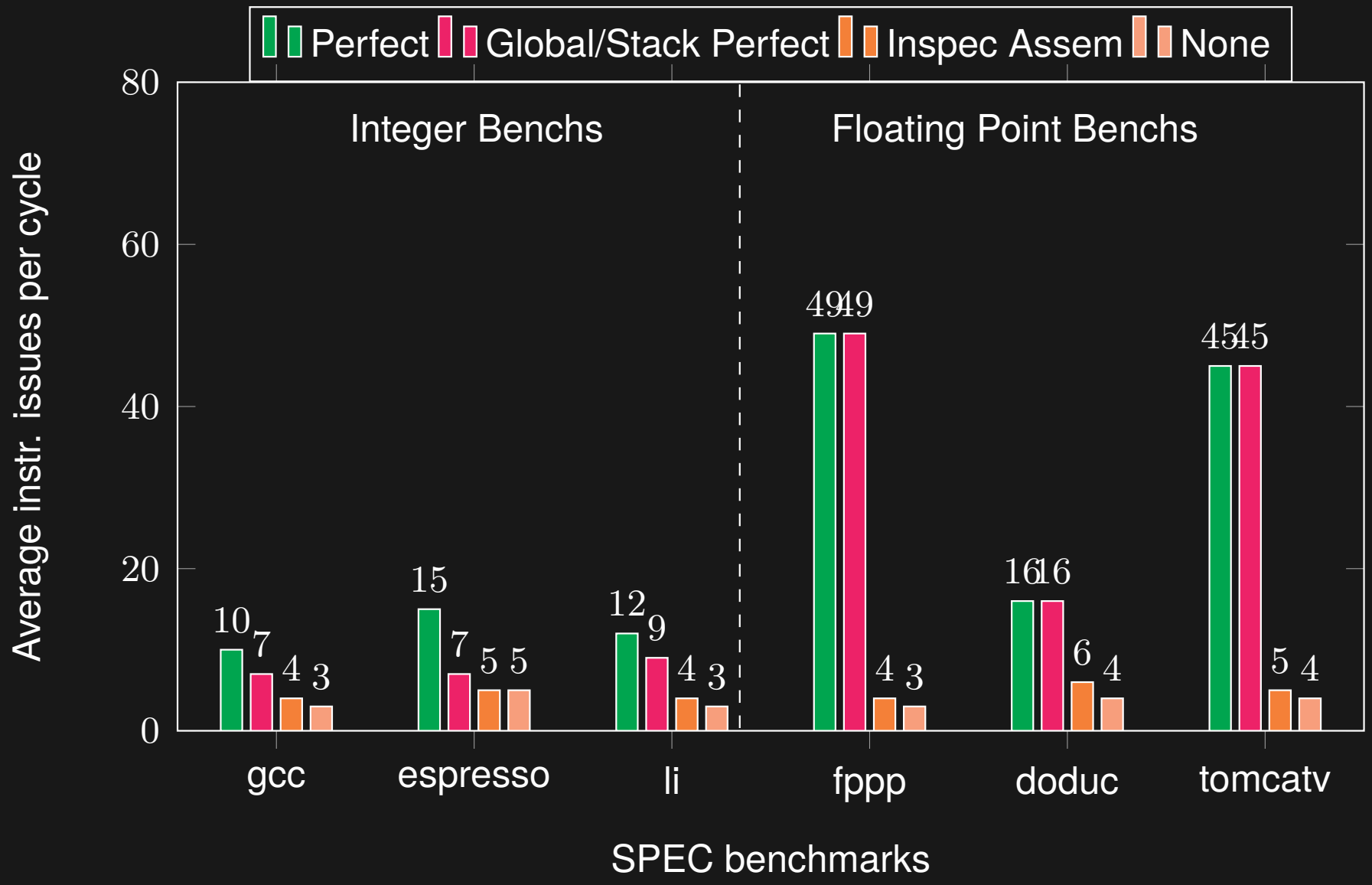


Realistic Memory Alias Analysis

- Realistic CPUs do not have perfect memory aliasing.
- What is the max ILP if we have fewer registers?

	Perfect Model	Current Model
Max Issues per Cycle	Infinite	64
Instruction Window Size	Infinite	2048
Renaming Regisers	Infinite	256 Int + 256 FP
Branch Prediction	Perfect	8K 2-Bit Counter
Memory Alias	Perfect	Perfect only for Global/Stack, Inspect, None
Memory/ Cache	Perfect	Perfect

Realistic Memory Alias Analysis cont'd



CPU Parameters for Current Intel CPUs

	Nehalem	SandyBridge	Haswell	Skylake
x86 Decoders	4 insn	4 insn	4 insn	5 insn
Max Insn/Cycle	4 ops	6 ops	8 ops	8 ops
Reorder Buffer	128 ops	168 ops	192 ops	224 ops
Load Buffer	48 loads	64 loads	72 loads	72 loads
Store Buffer	32 stores	36 stores	42 stores	56 stores
Scheduler	36 entries	54 entries	60 entries	97 entries
Integer Rename Regs	In ROB	160 regs	168 regs	180 regs
FP Rename Regs	In ROB	144 regs	168 regs	168 regs
Allocation Queue	28/thread	28/thread	56 total	64/thread

- "Store Buffer" is the number of entries in the finished store buffer (FSB)
- "Scheduler" is the number of entires in the centralized issue queue (IQ). RS sends insns to IQ, which sends insns to FUs.
- "Integer/FP Rename" is the number of physical integer and floating point registers
- "Allocation Queue" is a decoupling queue between front-end and back-end
- Nehalem to Sandy Bridge transitioned from value- to pointer-based register renaming

Improving ILP in Realistic CPU

- There is a huge gap between the ILPs of the perfect CPU and realistic CPUs.
- Theoretically, realistic CPU's ILP can be improved with better compiler and hardware designs.
  - For example,
    - Execute both speculated paths.
    - Value predictions to overcome data dependencies.
    - VLIW compilers.
  - There has been significant research effort on improving ILPs, some were not successful, but most of them require complex changes to the CPU.
  - "Designer discovered that trying to extract more ILP was simply too inefficient" – H&P

Road Map

The Limitations of ILP

Thread-level Parallelism

Acknowledgment

Performance Beyond Simple 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 Level Parallelism (TLP): Execute the instructions from multiple threads at the same time.
  - Threads may be from one process, or they may be from independent processes.
  - Each thread has all the state (instructions, data, PC, register state, and so on) necessary to allow it to execute
- Data Level Parallelism (DLP): Perform identical operations on data, and lots of data
  - i.e., SIMD.

Thread Level Parallelism (TLP)

- ILP exploits implicit parallel operations within a loop or straight-line code segment (a thread)
- TLP explicitly represented by the use of multiple threads of execution that are inherently parallel
- Goal: Use multiple instruction streams to improve
  - Throughput of computers that run many programs
  - Execution time of multi-threaded programs
- TLP could be more cost-effective to exploit than ILP

Multi-Threaded Execution in One CPU

- Multithreading (MT): multiple threads to share the functional units of one processor via overlapping
  - Processor must duplicate resources to track the states of each thread, e.g., separate copies of register file, separate PCs, and for running independent programs, separate page tables
  - 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 cycles.
- When to switch threads?
  - Alternate instruction per thread (fine grain)
  - When a thread is stalled, perhaps for a cache miss, another thread can be executed (coarse grain)

Fine-Grained Multi-threading

- 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 is it can hide both short and long stalls, since instructions from other threads executed when one thread stalls
- Disadvantage is it 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.

Coarse-Grained Multi-threading

- Switches threads only on costly stalls, such as L2 cache misses
- Used in IBM AS/400
- Advantages
  - Relieves the need to have very fast thread-switching
  - Does not slow down one thread, since instructions from other threads issued only when the thread encounters 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

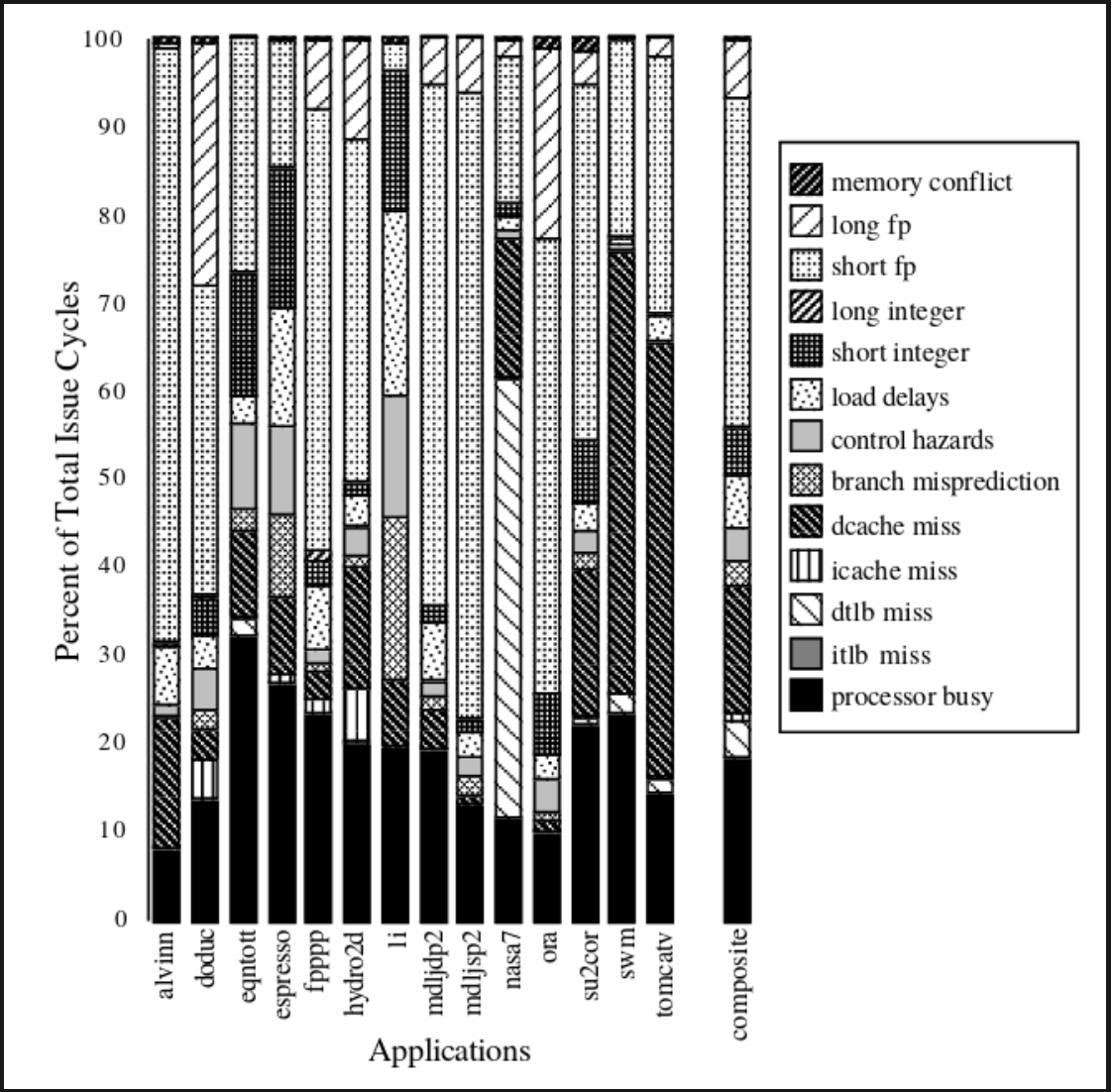
Integrate ILP and TLP

- TLP and ILP exploit two different kinds of parallel structure in a program.
- Could a processor oriented at ILP exploit TLP?
  - functional units are often idle in data path 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 ILPexists?

Simultaneous Multi-Threading

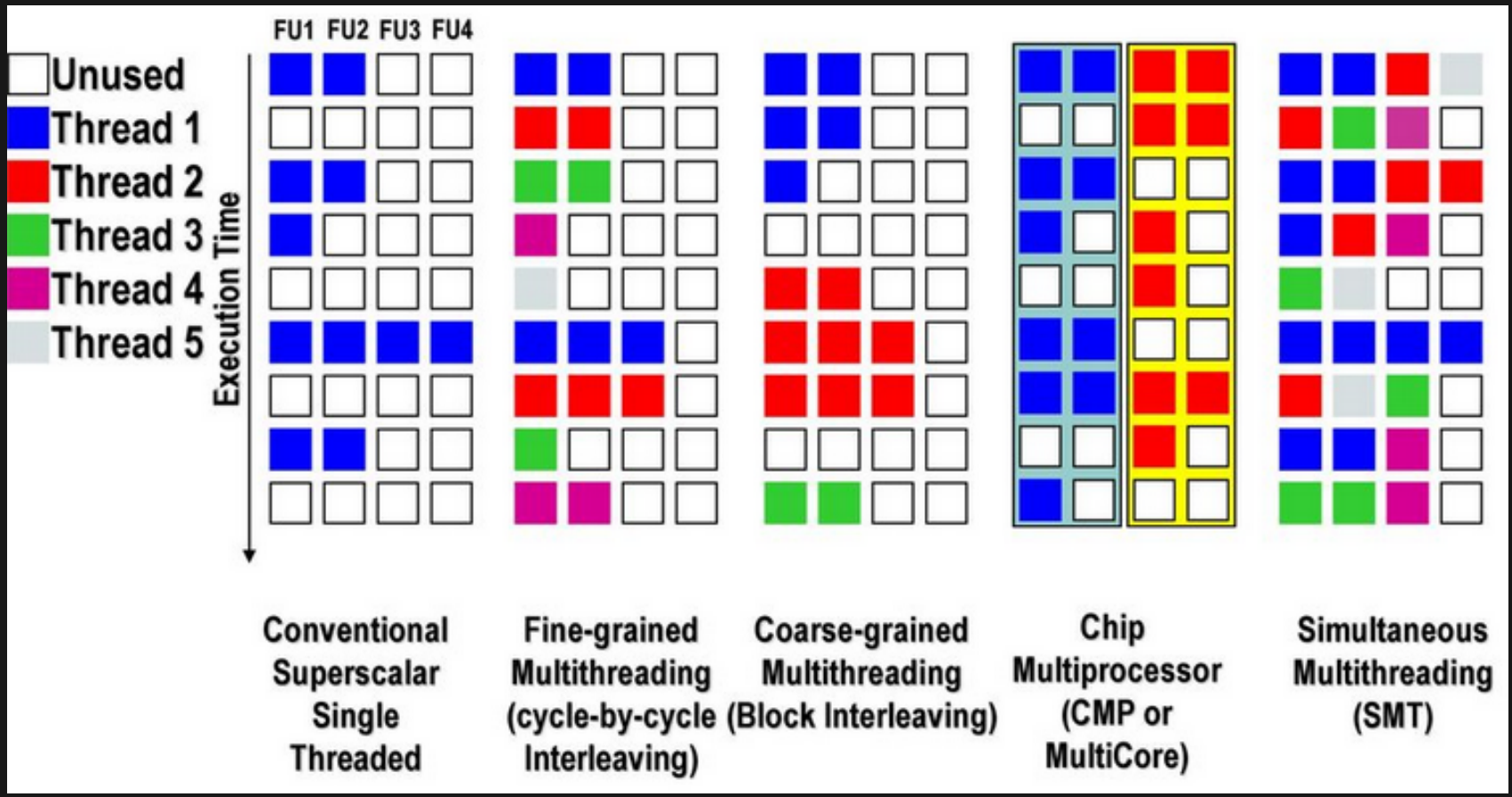
- Simultaneous Multi-Threading (SMT) is a variant of fine-grained MT.
- SMT is based on the observation that dynamic scheduling already has the HW mechanisms to support
  - Large set of ROB/RS registers that can be used to hold the register sets of independent threads
  - Register renaming provides unique 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 ofthe HW
- Just add a per thread renaming table and keep separate PCs
  - Independent commits can be supported by logically keeping a separate reorder buffer for each thread

Quantitative Motivation for SMT



Most applications experience considerable stalls during execution. And for different applications, they stall at different functional units. Therefore, it is beneficial to interleaving the execution. Figure from Tullsen et al. 1995 ISCA.

Multi-threading Categories



In current SMT implementations, in one cycle, instructions are only issued from one thread. However, in the subsequent cycle, the instructions could be issued from any thread (instructions are issued once they are ready). Therefore, at any time, there could be instructions from different threads executing. Figure by Dr. Weidong Shi, UH.

Design Challenges in SMT

- Since SMT makes sense only with fine-grained implementation, impact of fine-grained scheduling on single thread performance?
  - Will a preferred thread approach sacrifice neither throughput nor single-thread performance?
  - Unfortunately, with a preferred thread, the processor is likely to sacrifice some throughput, when preferred thread stalls
  - More issues on single thread performance on next slide.
- Larger register file needed to hold multiple contexts

Design Challenges in SMT cont'd

- Should not affect clock cycle time, especially when
  - 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
  - Modern SMT implementation typically has performance penalty for each individual threads due to cache, TLB and FU contention.
    - i.e., a thread could be slower when running under SMT than no SMT.
  - Single-thread performance can be unstable on SMT due to the contention.
  - There has been quite some research on finding the optimal group of threads to shared one SMT CPU.

Examples of SMT and MT

- Sun's Niagara employs fine-grained MT.
- Examples of SMT include Intel's HyperThreading CPUs and IBM's Power CPUs.
- There is also a design that duplicates popular or frequently-contended FUs in SMT. This design is called conjoint-core. AMD's Bullzoer is such an architecture, with threads have their own integer FUs but share FPUs.

Real SMT Performance

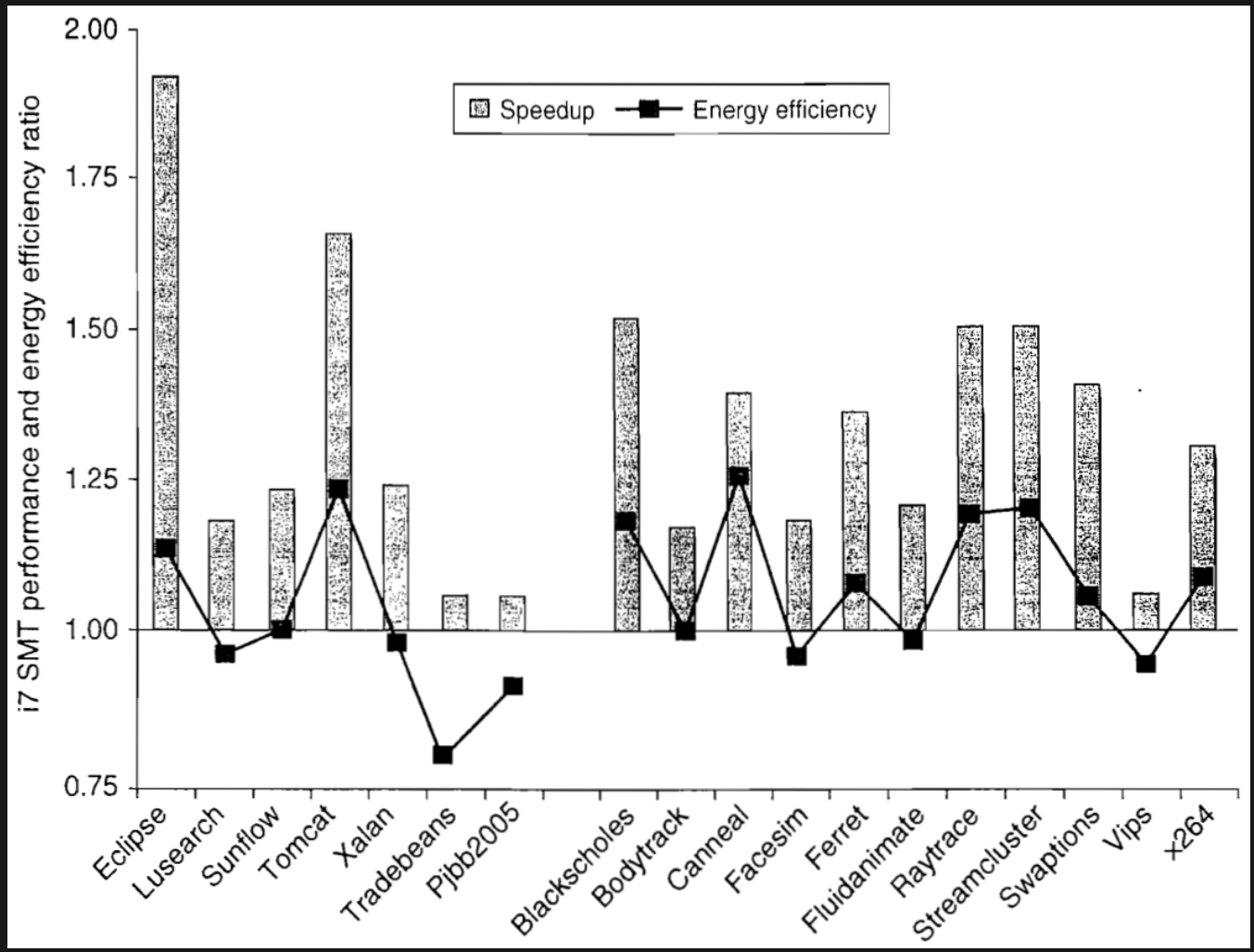


Figure 3.35 from H&P. Average SPEC Java programs speedup: 1.28, PARSEC speedup: 1.31. If the energy-efficiency is larger than 1, then the speedup is achieved with relatively less power consumption.

Summary on SMT

- The main performance benefit of SMT is not single thread performance, but overall throughput of multiple threads.
  - Modern SMT implementation typically has performance penalty for each individual threads due to cache and TLB contention.
- SMT has a major benefit in terms of energy-efficiency.
  - Stalled cycles are consuming as much power as a runing CPUs.
  - Therefore, reusing stalled cycles for other threads can improve energy-efficiency.

SMT and Security Concerns

- SMT can be a security vulnerability.
  - Simulatenously executing threads can inspect each other's execution status and cache status.
    - Allowing side-channel attacks
    - CVE-2005-0109, TLBleed (maybe...)
  - Disable SMT if concerned.

Road Map

The Limitations of ILP

Thread-level Parallelism

Acknowledgment

Acknowledgment

This lecture is partially based on the slides from Dr. Chau-Wen Tseng. Originally, the study on ILP limitation was done by David Wall in 1993 (“Limitations of Instruction-Level Parallelism”).