ΕΠΛ605:Προχωρημένη Αρχιτεκτονική Υπολογιστών

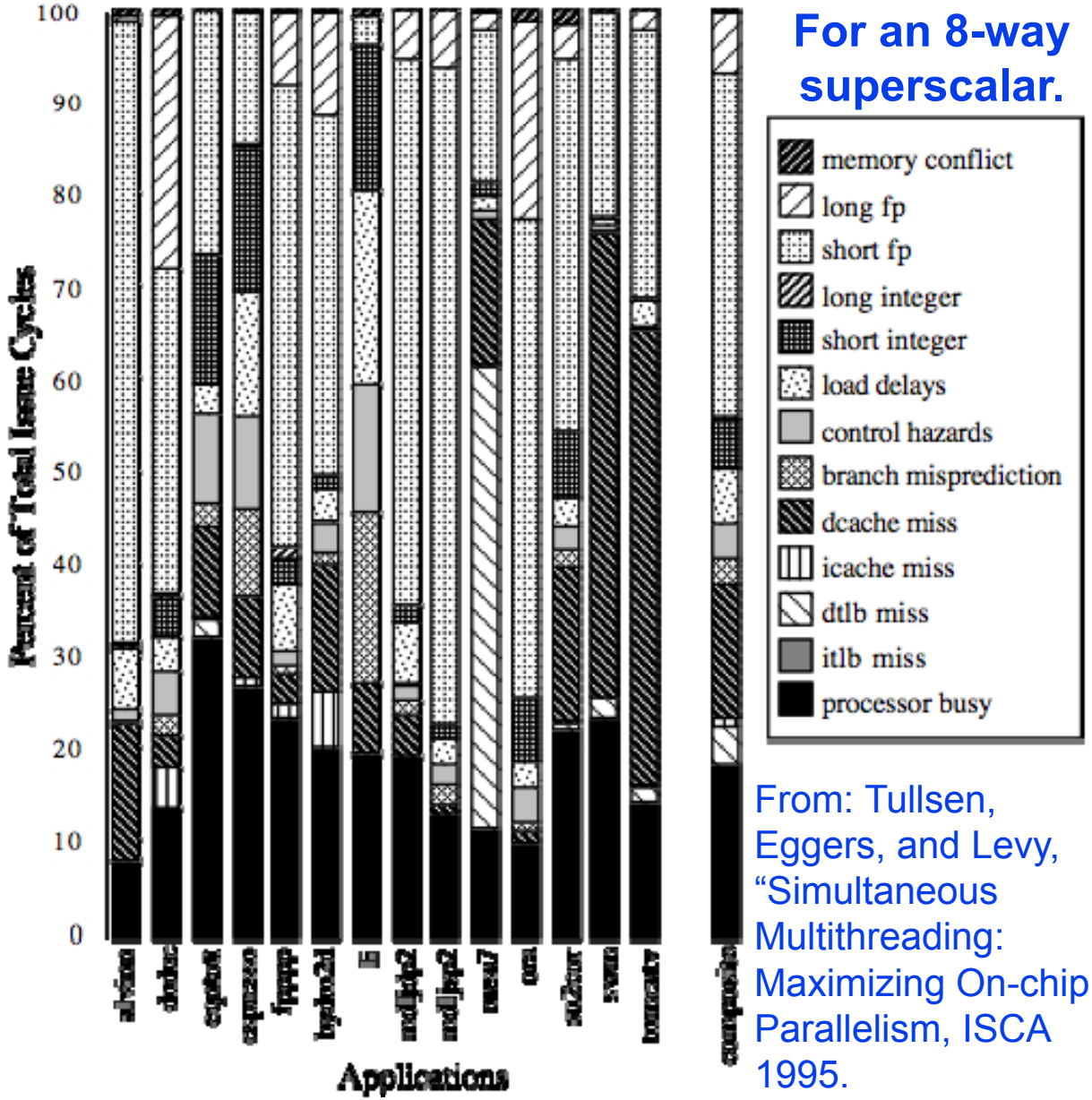
Γιάννος Σαζεϊδης

Κεφ. 3: MT and SMT
how to execute multiple threads

Διαβάστε κεφ. 3

Εαρινό Εξάμηνο 2017

OBSERVATION: for most apps, most execution units lie idle



For an 8-way superscalar.

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

Same hardware do both ILP and TLP? (TLP: thread level parallelism)

- TLP and ILP exploit two different kinds of parallel structure in a program
  - TLP: Increase throughput of computers by running independent programs OR parallel programs
- Could a processor oriented at ILP to 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 ILP exists?

Alternative to ILP: Multithreaded Execution

- Multithreading: multiple threads to share the functional units of 1 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/larger TLB
  - memory shared through the virtual memory mechanisms, which already support multiple processes
  - HW for fast thread switch; much faster than full process switch ≈ 1 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)

Fine-Grained Multithreading (FGMT)

- Switches between threads on each cycle, 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 it can hide both short and long stalls, since instructions from other threads executed 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: trade-off throughput for latency
- Conceptually at any given time
  - one thread owns pipe stage
  - But different threads can own different stages
  - threads can be switched very quickly
- Used in Niagara (some variation in GPUs)
  - No or simple forwarding – simpler hardware

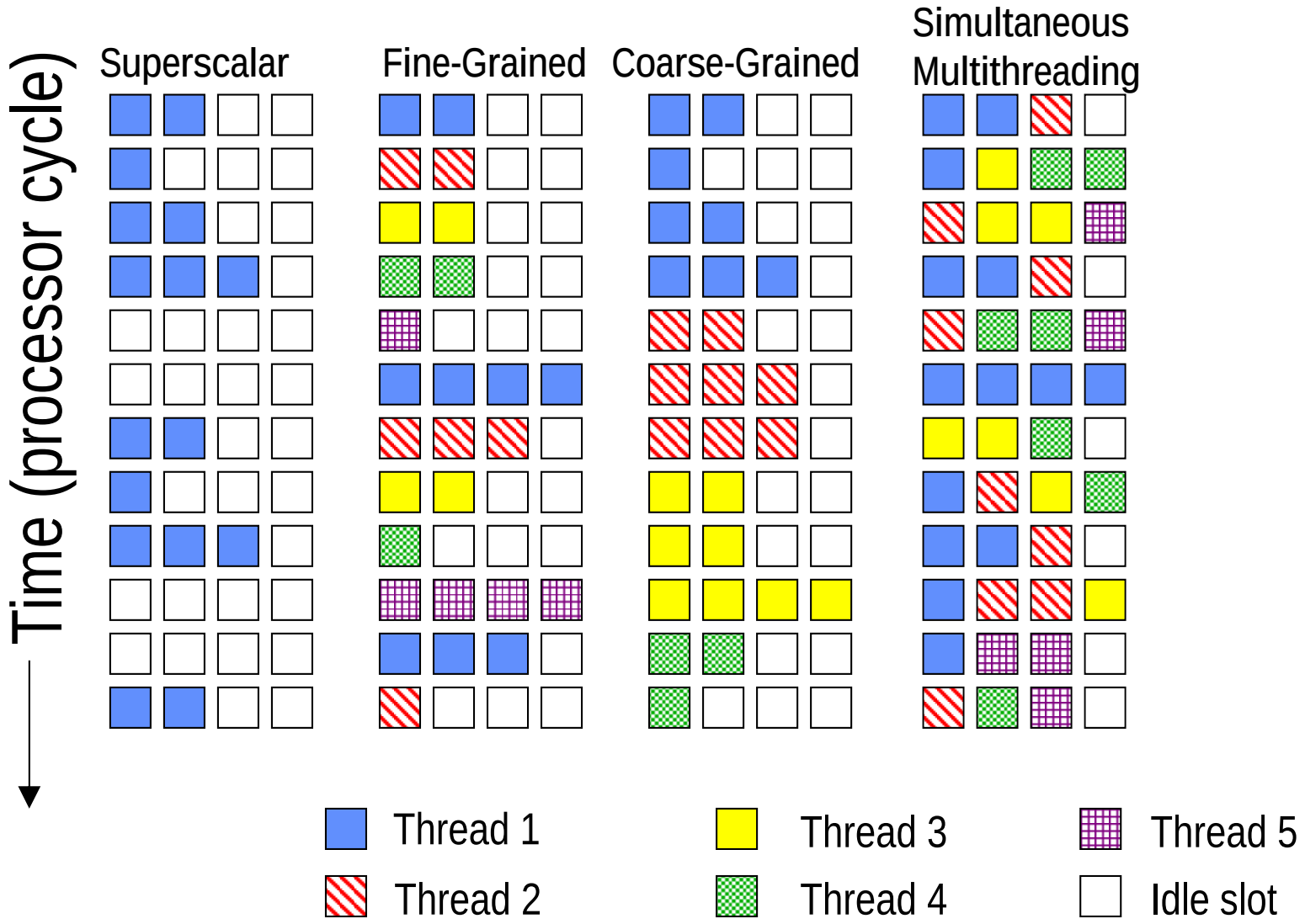
Coarse-Grained Multithreading (CGMT)

- Switches threads only on costly stalls, such as L2 cache misses
- Advantages
  - Relieves need to have very fast thread-switching
  - Doesn't slow down thread, since instructions from other threads issued only when the thread encounters a costly stall
- Disadvantage
  - 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
- Conceptually at any given time
  - one thread owns entire pipeline
- Used in IBM AS/400

Simultaneous Multithreading (SMT)

- Simultaneous multithreading (SMT): dynamically scheduled processor already has many HW mechanisms to support multithreading
  - Large set of physical 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 adding a per thread renaming table and keeping separate PCs, prediction history, TIDs in caches
  - Independent commit can be supported by logically keeping a separate reorder buffer for each thread
- Conceptually at any given time
  - many thread can share pipe stages
  - Ease of implementation: some stages owned by one thread at a time (e.g. fetch, commit)
- Used in Intel, IBM and AMD processors

Multithreaded Categories



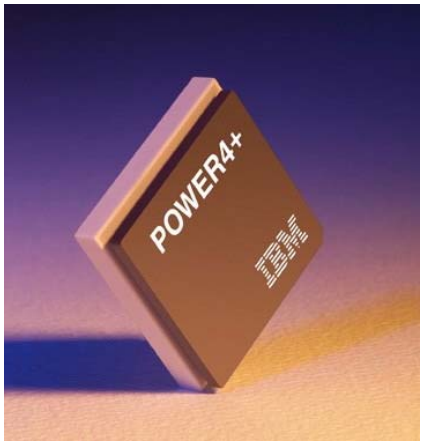
Design Challenges in SMT

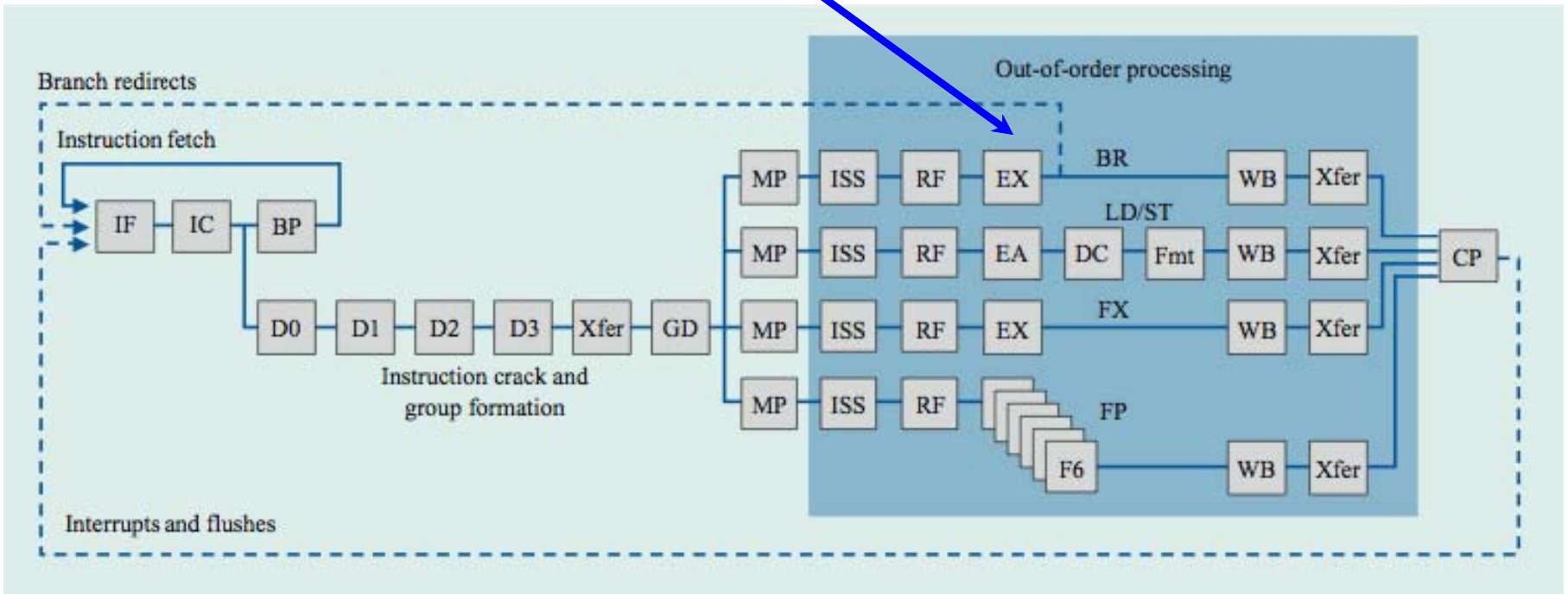
- Larger register file needed to hold register state for multiple threads
- More pressure on shared resources, such as register, caches, predictors, fu, buses
- Not affecting clock cycle time, especially in

- Selection tricky:
  - If can fetch, rename and complete from one thread at a time, how to choose next thread at each stage?

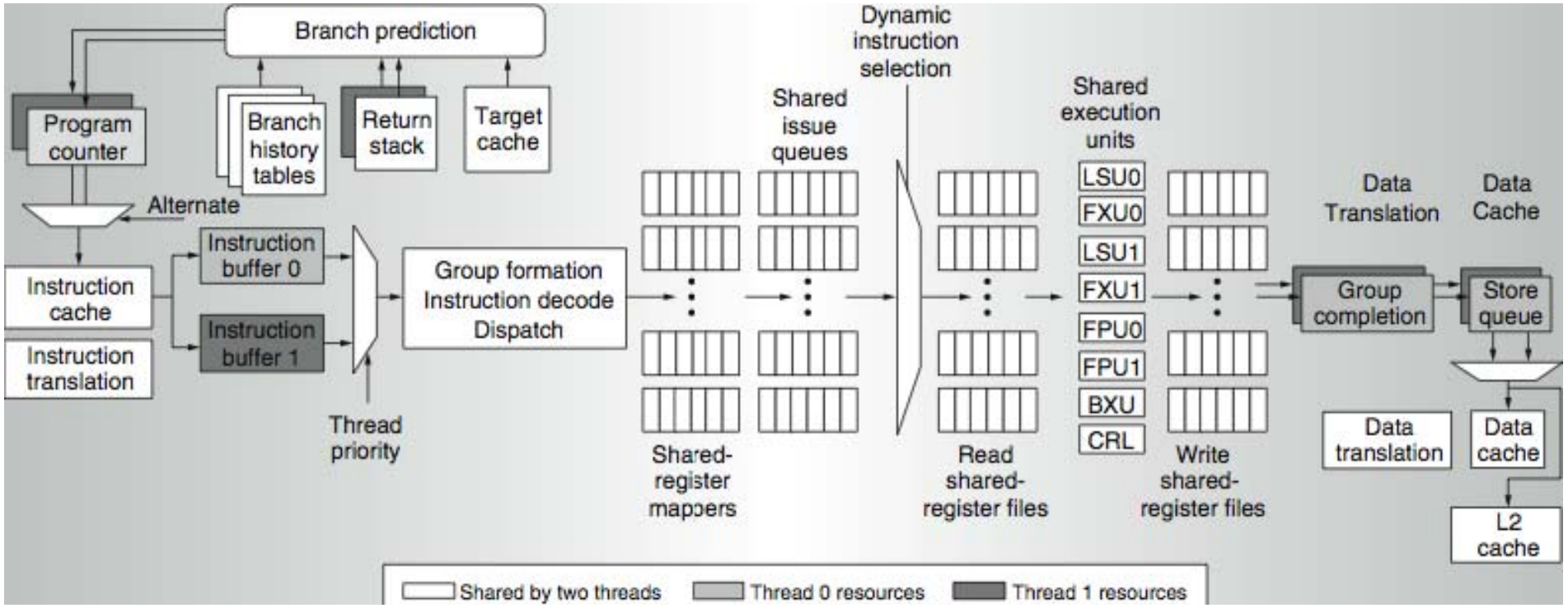
Power 4

Single-threaded predecessor to Power 5. 8 execution units in out-of-order engine, each may issue an instruction each cycle.





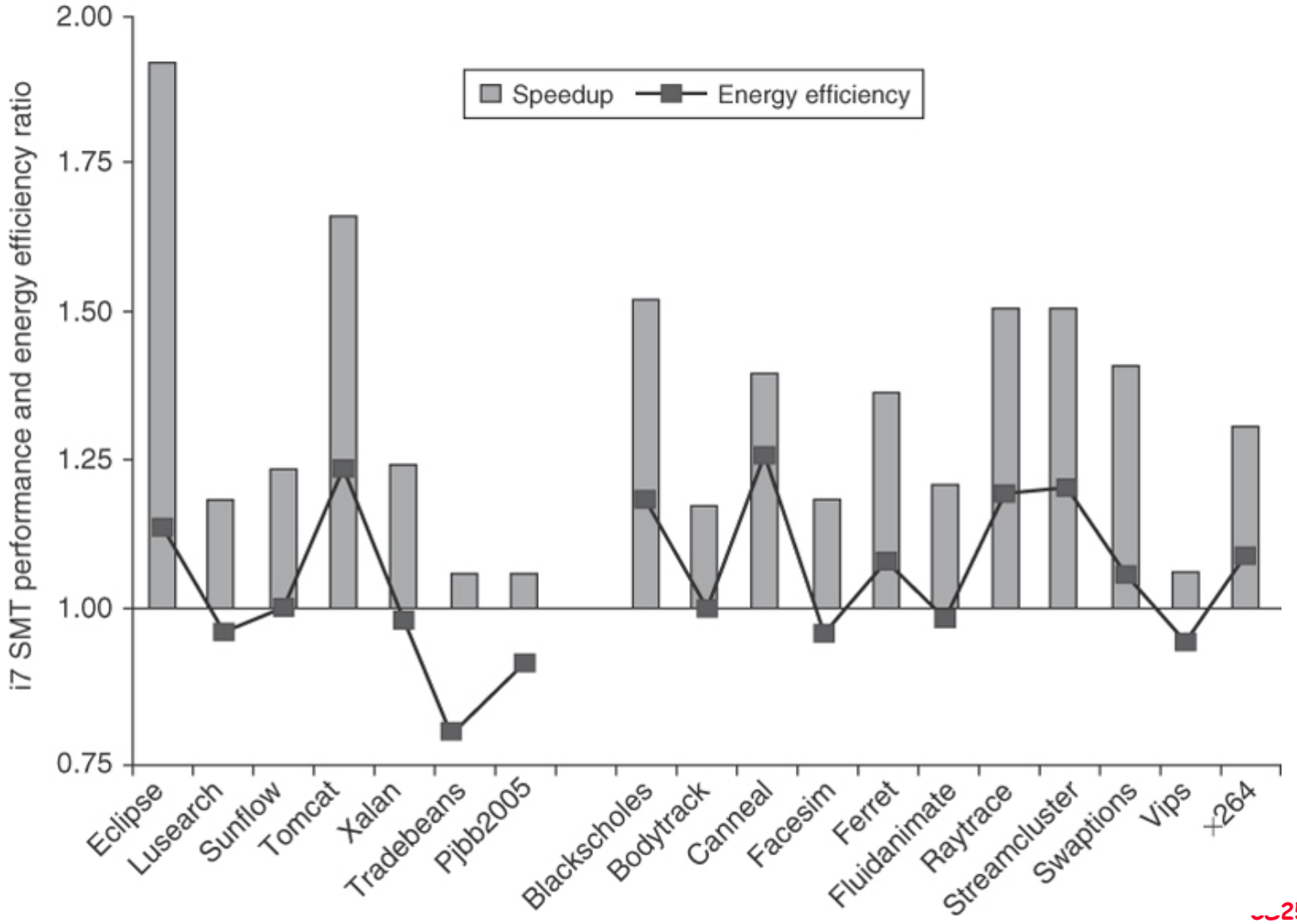
Power 5 SMT


Branch prediction
Program counter
Branch history tables
Return stack
Target cache
Alternate
Instruction cache
Instruction translation
Instruction buffer 0
Instruction buffer 1
Thread priority
Group formation
Instruction decode
Dispatch
Shared issue queues
Dynamic instruction selection
Shared execution units
LSU0
FXU0
LSU1
FXU1
FPU0
FPU1
BXU
CRL
Shared-register mappers
Read shared-register files
Write shared-register files
Data Translation
Data Cache
Group completion
Store queue
Data translation
Data cache
L2 cache
Shared by two threads
Thread 0 resources
Thread 1 resources

Changes in Power 5 to support SMT

- Two PCs
- 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 physical (virtual) registers from 152 to 240
- Increased the size of several issue queues
- The Power5 core is about 24% larger than the Power4 core because of the addition of SMT support

Performance of SMT (intel i7) running two threaded parallel program



And in conclusion …

- Limits to ILP (power efficiency, dependencies, stalls …) seem to limit to 4 to 8 issue for practical options
  - Still critical to improve
  - Amdahl's Law
- Improve utilization with Multithreading
- Coarse grain vs. Fine grained multihreading
  - Only on big stall vs. every clock cycle
  - Can help simplify hardware
- Simultaneous Multithreading: fine grained multithreading based on superscalar microarchitecture

Conclusion/Trends

- ILP Trends:
  - Still interesting but power efficient
  - Complex hardware is power hungry and hot
  - Use more efficiently power
  - Power/Temp considerations motivate shift to multi-cores
- Compiler based ILP
  - Low cost
  - Complimentary to dynamic ILP
  - VLIW do all in sw with specially designed ISA and uarch
- SMT-cores to exploit idle resources
  - Threaded program
  - Independent programs
- Is there enough thread level parallelism
  - Depends on application
  - Beyond SMT: SIMD, DLP, multi-core, many-core, accelerators

Fine grain multithreading

- 6 Threads on a 5 stage pipeline
- No hazards
  - Without MT: CPI 1
  - With MT 6 threads CPI = 1 cycle

	F	D	E	M	WB
1	T1				
2	T2	T1			
3	T3	T2	T1		
4	T4	T3	T2	T1	
5	T5	T4	T3	T2	T1
6	T6	T5	T4	T3	T2
7	T1	T6	T5	T4	T3
8	T2	T1	T6	T5	T4
9	T3	T2	T1	T6	T5
10	T4	T3	T2	T1	T6
11	T5	T4	T3	T2	T3

Fine grain multithreading

- Say 10% of LDs miss, 40% loads, 10 cycle penalty
  - Assume no miss for more than a thread at the same time
  - Without MT: 1.4
  - With MT 6 Threads = 1.04 cycles

	F	D	E	M	WB
1	T1				
2	T2	T1			
3	T3	T2	T1		
4	T4	T3	T2	T1	
5	T5	T4	T3	----	T1
6	T6	T5	T4	T3	----
7	T1	T6	T5	T4	T3
8	T3	T1	T6	T5	T4
9	T4	T3	T1	T6	T5
10	T5	T4	T3	T1	T6
11	T6	T5	T4	T3	T1
12	T1	T6	T5	T4	T3
13	T2	T1	T6	T5	T4
14	T3	T2	T1	T6	T5
15	T4	T3	T2	T1	T6