Parallel Computer Architecture
Spring 2019

Thread Level Parallelism Simultaneous Multithreading (SMT)

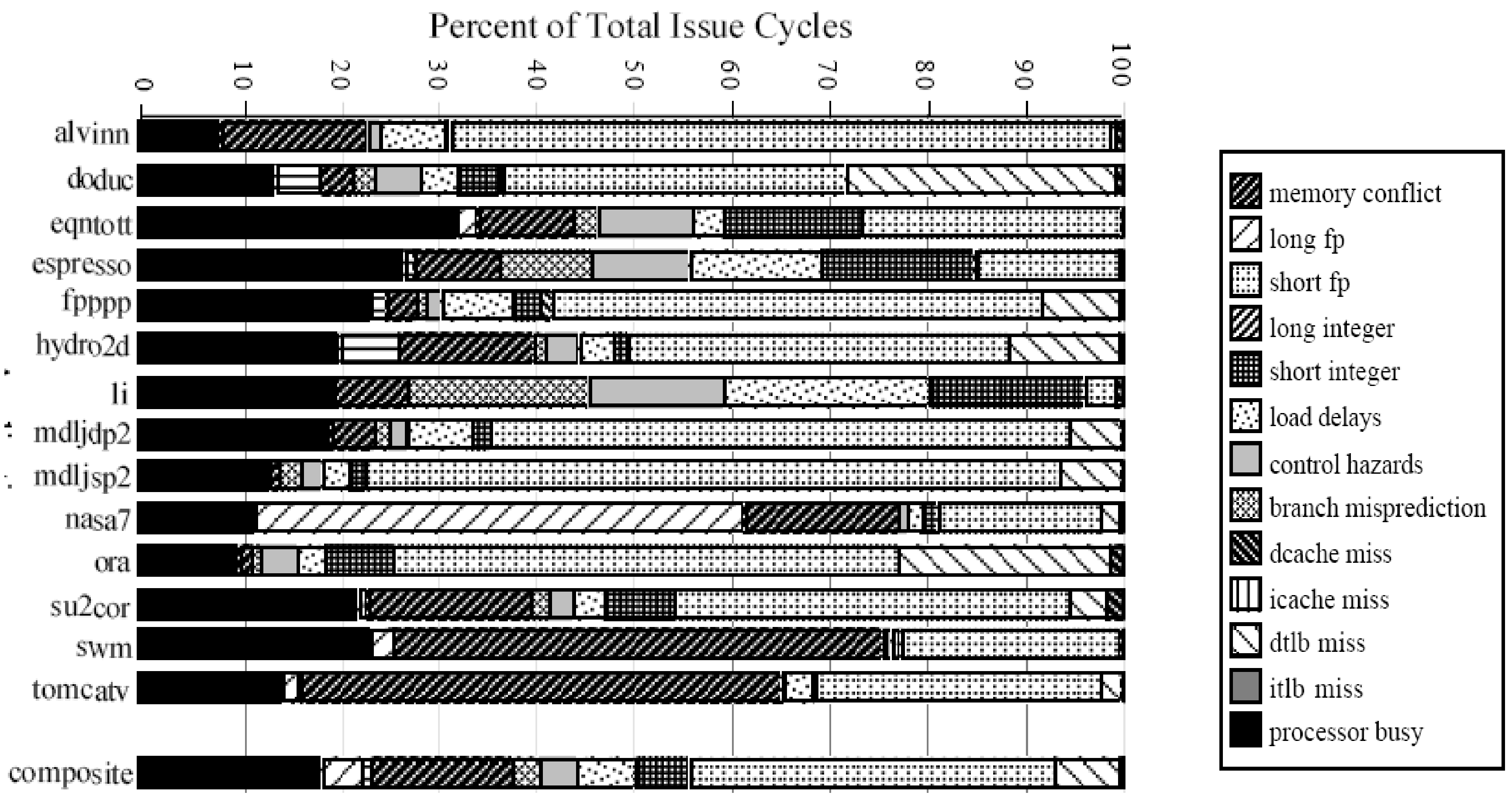
Nikos Bellas

Electrical and Computer Engineering Department
University of Thessaly

ILP is not enough

- ILP is not enough in a high performance CPU
  - ILP is local. Due to the limited instruction window size, only instructions from a small region are examined for parallelism
  - Cache misses (to L2, L3 and mainly to main memory), imperfect branch predictions and large functional latencies (e.g. DIV.D) have significant latency
  - In a lot of programs, ILP is inherently small (e.g. SPECint benchmarks)

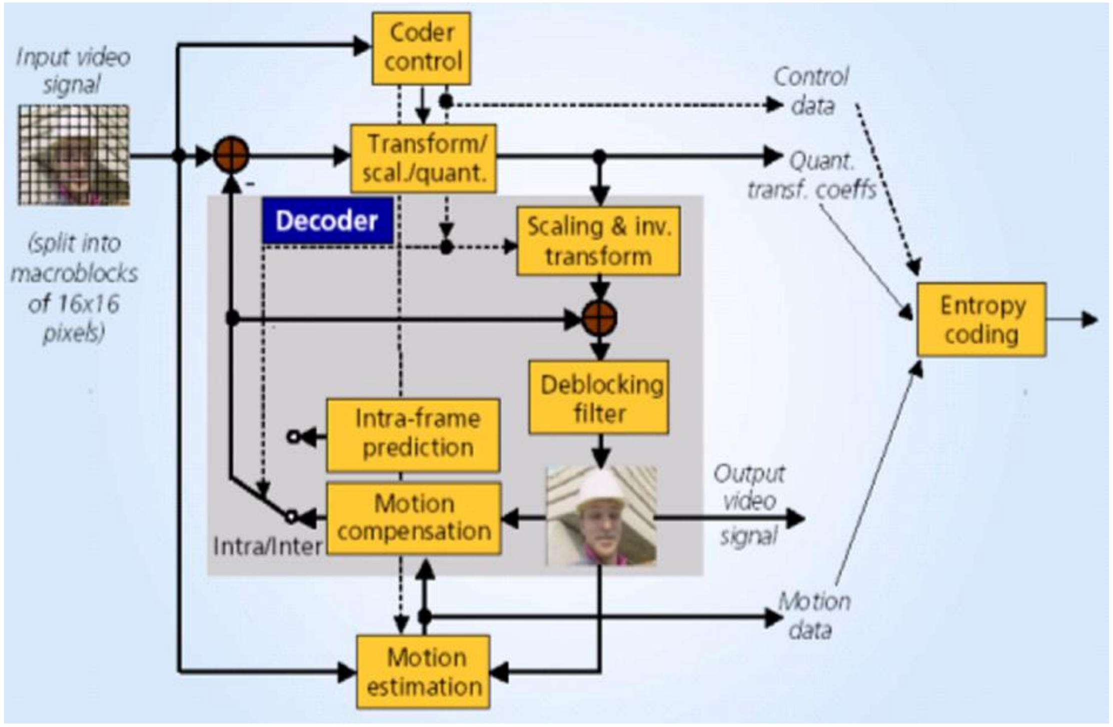
Where do cycles go?



Based on Alpha 21164 (8 issue peak rate)
SPEC92 benchmarks
Only 19% useful cycles

Coarse-grain level parallelism

- **Frequently, abundant parallelism at higher level of the application**
- **Mainly due to the nature of the algorithm**
  - Transaction processing systems (web-servers, DB servers, storage servers)
  - Video compression



Coarse-grain level parallelism

- Different levels of coarse-grain parallelism
- Processes are the “heaviest” unit of granularity
    - Processes own resources allocated by the O/S
    - Resources include memory, file handles, sockets, device handles, and windows
    - Normally, no sharing of resources among processes
    - Processes are typically pre-emptively multitasked by the O/S
    - E.g. Writing a word document, listening to an MP3, web browsing at the same time
    - Only one process runs at any time in a single-threaded core

Coarse-grain level parallelism

- Threads are in the same process and share the address space
- ILP is implicit, the programmer knows nothing about it
- Thread Level Parallelism (TLP) is explicitly represented by the programmer
    - POSIX thread programming :

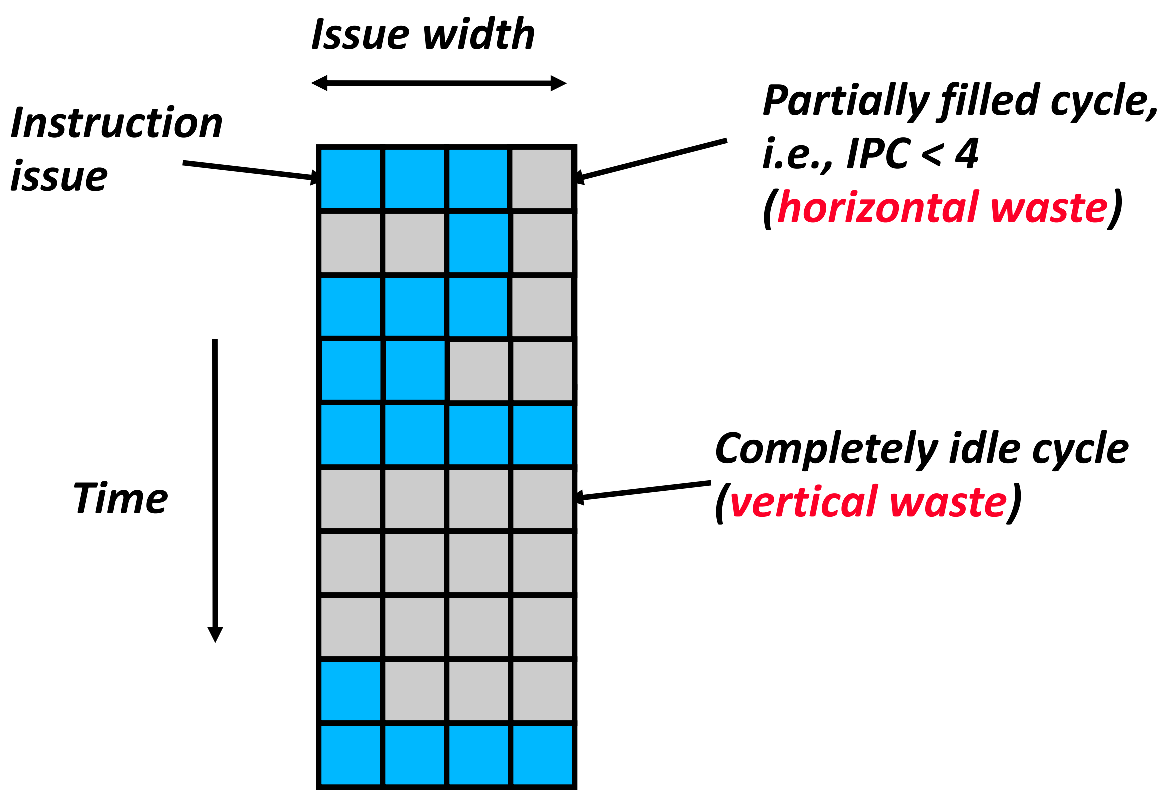
```
int iret1 = pthread_create( &thread1, NULL, &function_name, (void*) message1);
```

    - Previous example with video compression
    - Automatic TLP extraction by the compiler, is a hot research topic.
    - With very limited results….
- ILP and TLP are complementary
- Exploit different sources of parallelism in the program

Coarse-grain level parallelism

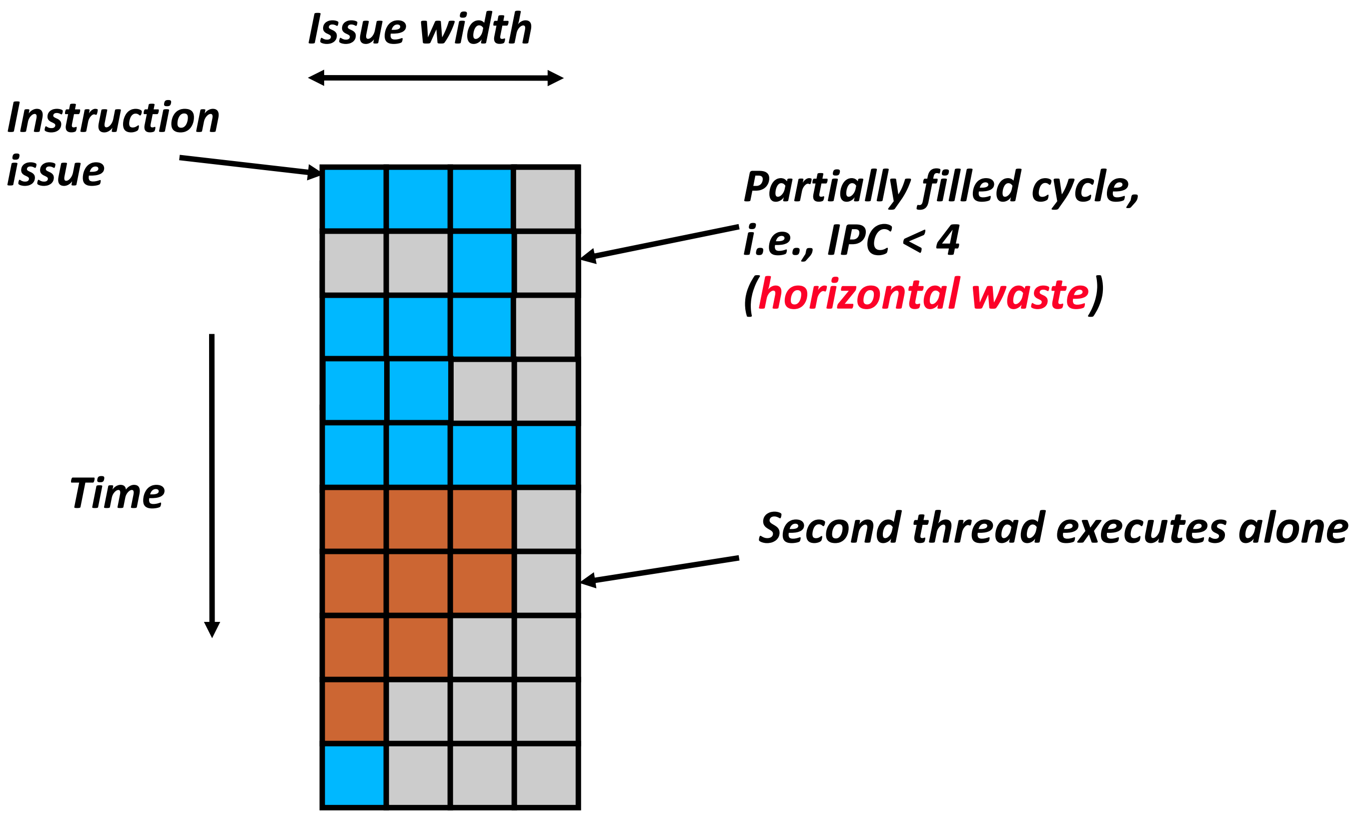
- **Questions**
  - Can a superscalar single-core processor exploit both ILP and TLP?
  - Can the TLP be used to employ idle resources when insufficient ILP exists?
  - If yes, what changes are needed in the architecture?
  - What are the performance gains versus the extra complexity and power dissipation?

Superscalar – single thread


Issue width
Instruction issue
Partially filled cycle, i.e., IPC < 4 (horizontal waste)
Completely idle cycle (vertical waste)
Time

Constrained by lack of ILP

Coarse-grain multithreading

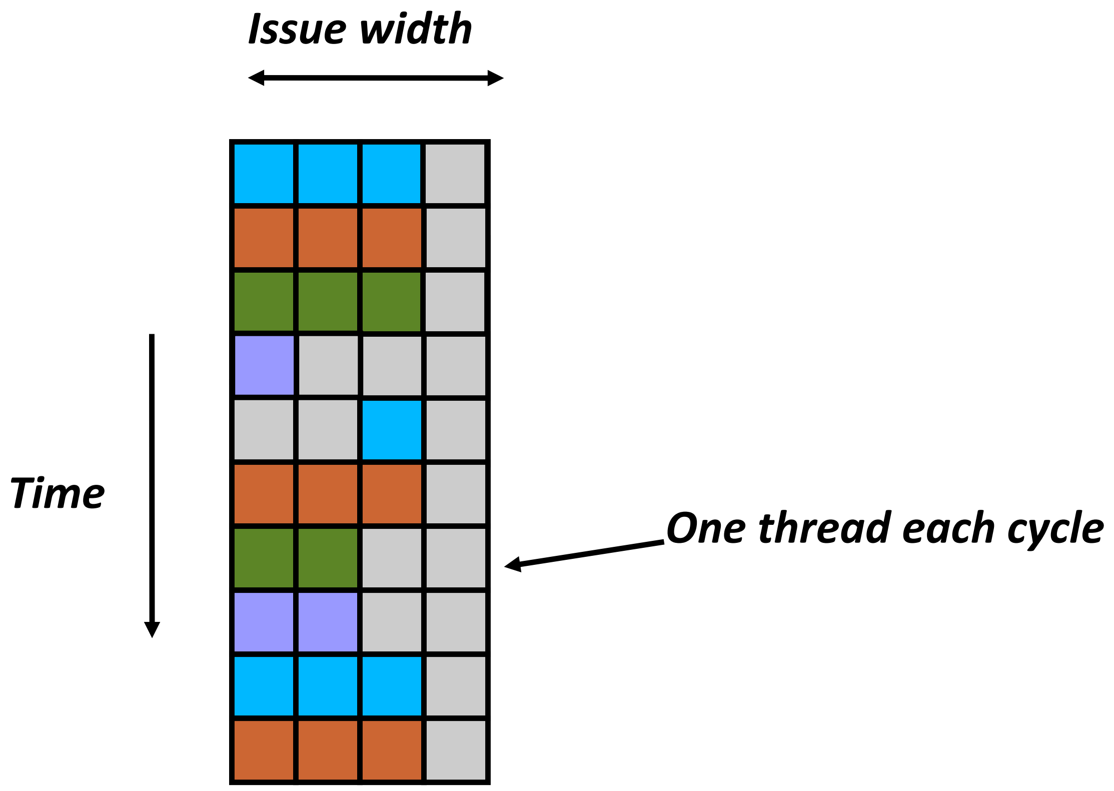

Issue width
Instruction issue
Time
Partially filled cycle, i.e., IPC < 4 (horizontal waste)
Second thread executes alone

Threads run for more than a cycle. Only one thread every cycle.

Removes vertical waste, but leaves some horizontal waste

Thread switch only in case of costly stalls (memory stalls) and have to pay the cost of refilling the pipeline

Fine-grain multithreading

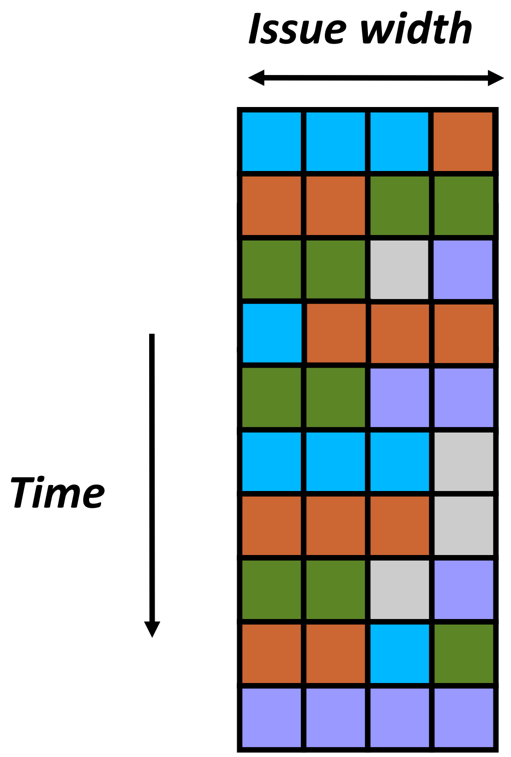


Threads run for a single cycle. Only one thread every cycle.

Removes vertical waste, but leaves some horizontal waste

- GPUs
- Sun's Niagara processor

Simultaneous multithreading (SMT)



- Interleave multiple threads to multiple issue slots with no restrictions

SMT issues

- Higher throughput than single thread superscalar
- BUT, higher latency per thread
- Tradeoffs between throughput and latency are possible
- “Preferred” threads are given higher priority
    - Instruction prefetch, resources are given mainly to the preferred thread
    - If it stalls, other threads are considered

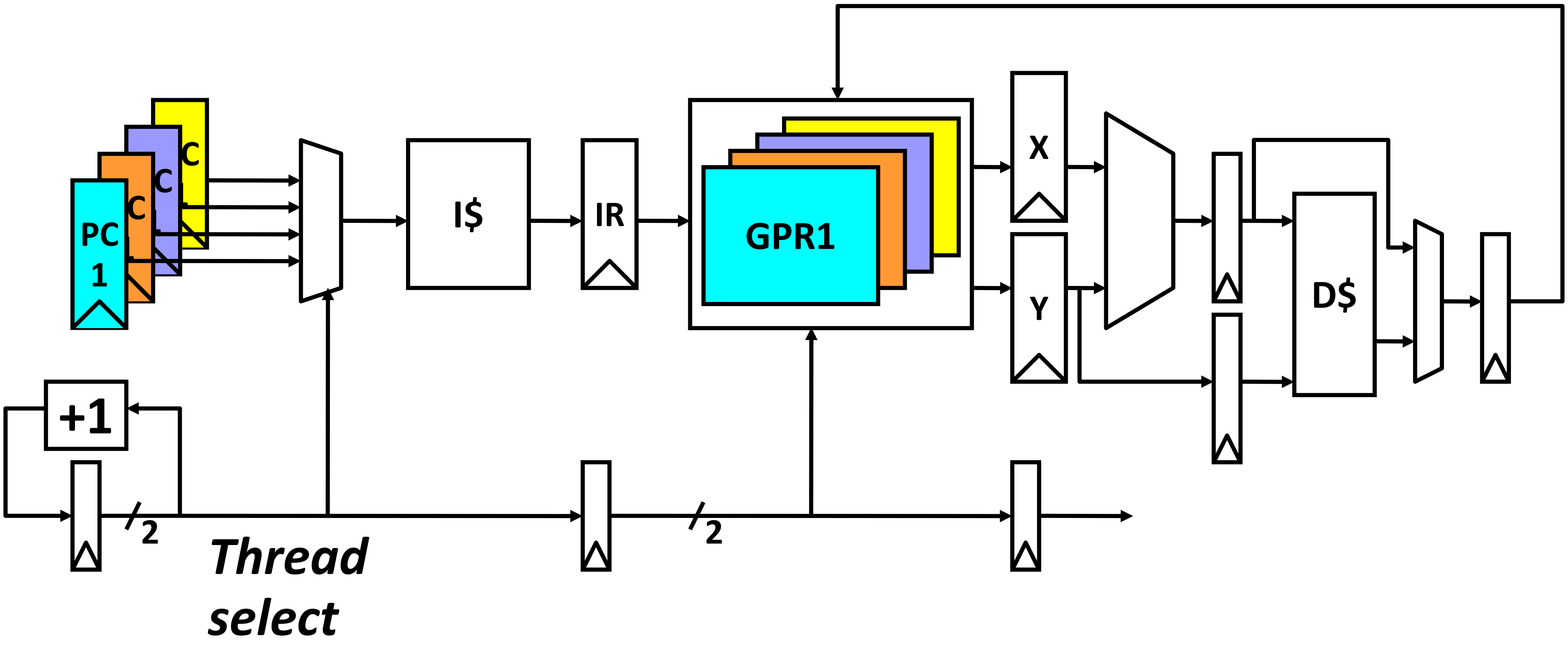
Out of Order SMT

- Add multiple contexts and fetch engines and allow instructions fetched from different threads to issue simultaneously
- Utilize wide out-of-order superscalar processor issue queue to find instructions to issue from multiple threads
- OoO instruction window already has most of the circuitry required to schedule from multiple threads
- Any single thread can utilize whole machine
- In all SMTs, instruction issue from a single thread in each cycle
- However, execution can combine instructions from multiple thread in each cycle

SMT Costs

- Each thread requires its own user state
  - PC
  - Fetch Units
  - Control Registers
- Also, needs its own system state
  - virtual memory page table base register
  - exception handling registers
- Design challenges
  - Larger register file
  - Larger fetch units
  - Clock cycle may be affected

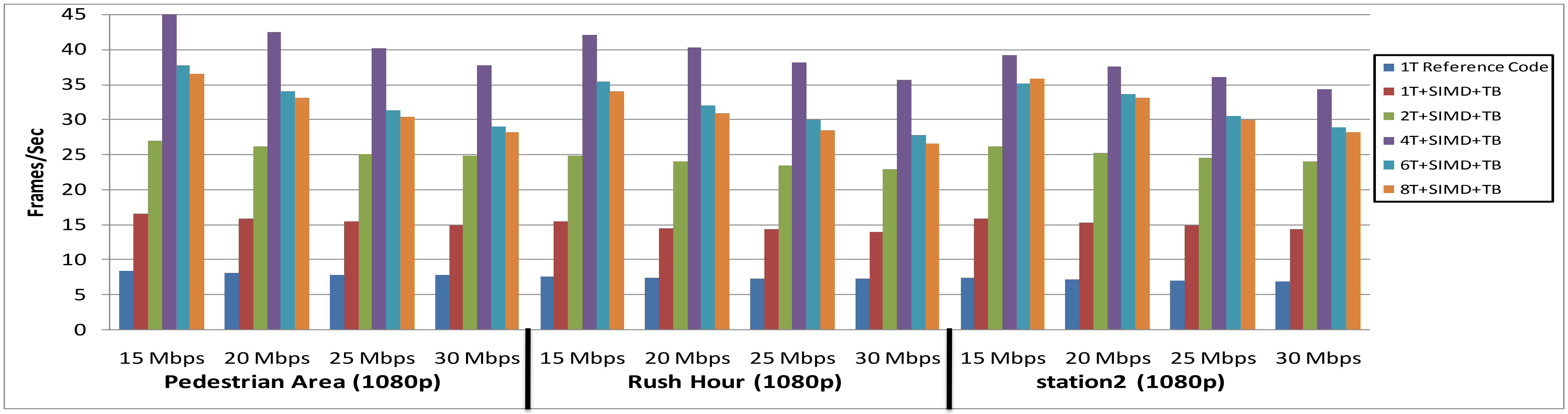
Simple SMT pipeline



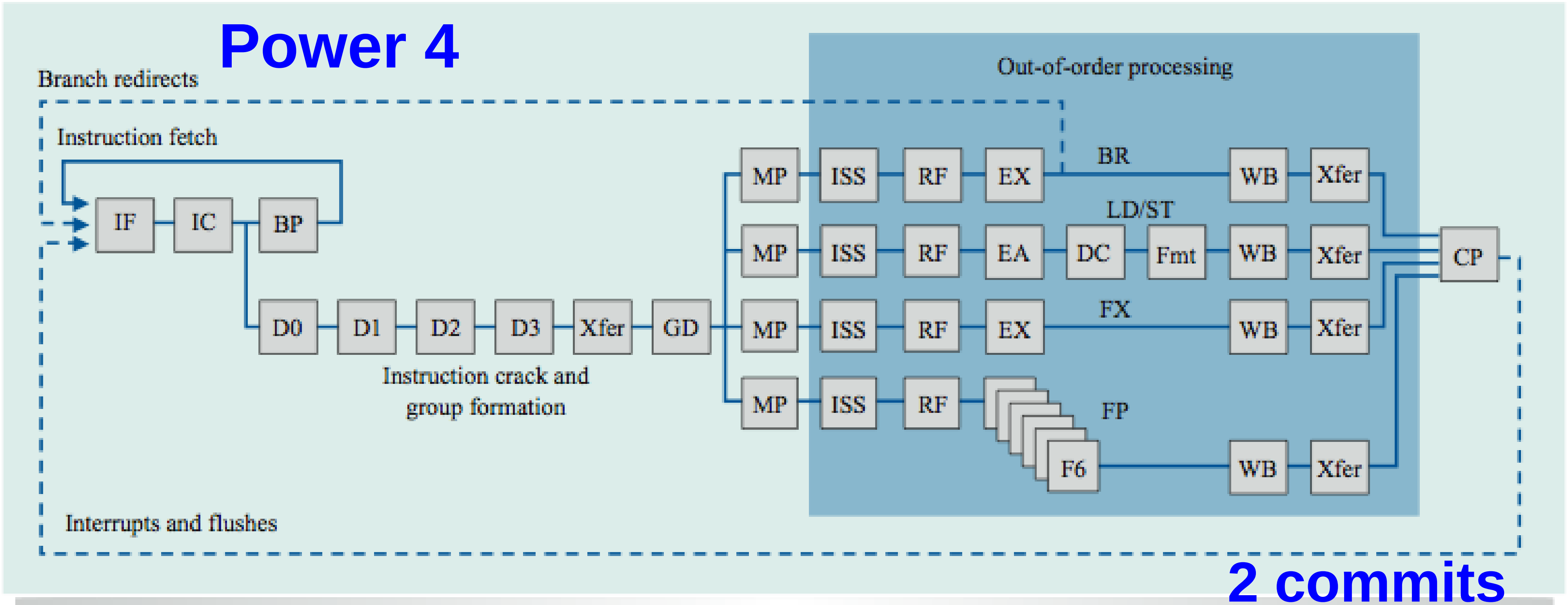
- Have to carry thread select down pipeline to ensure correct state bits read/written at each pipe stage
- Appears to software (including OS) as multiple, albeit slower, CPUs

SMT performance

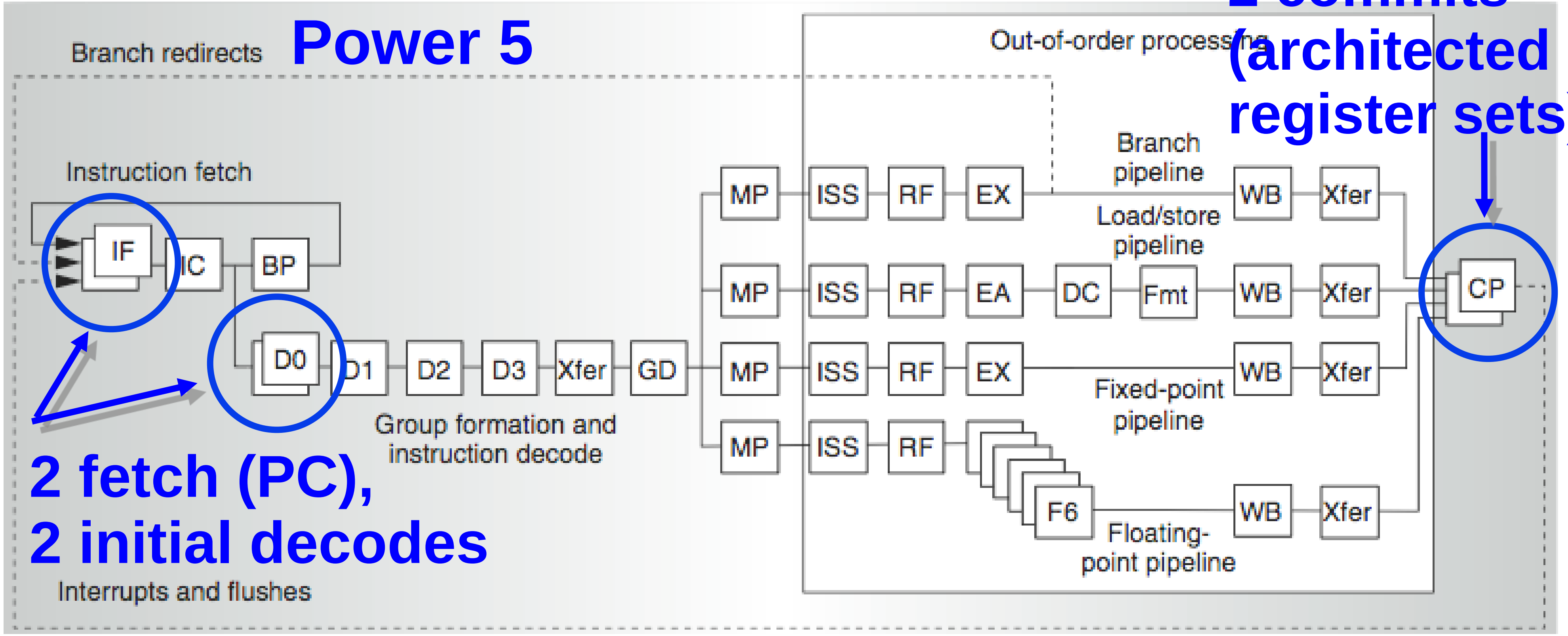
- SMT does not always offer performance improvements
  - AVS Video decoding example : performance (in fps) drops for #of threads > #cores
  - Core i7 with 4 cores and Hyperthreading enabled



Power 4



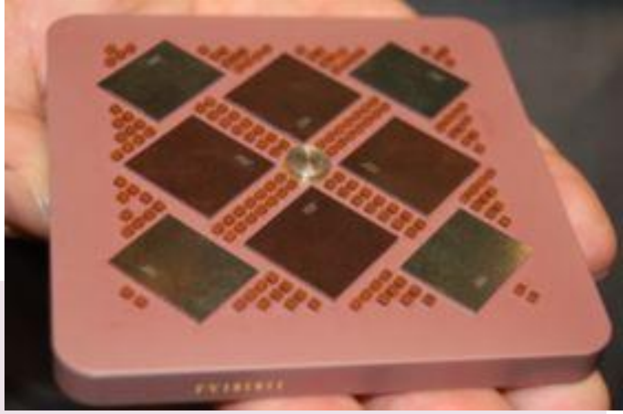
Power 5

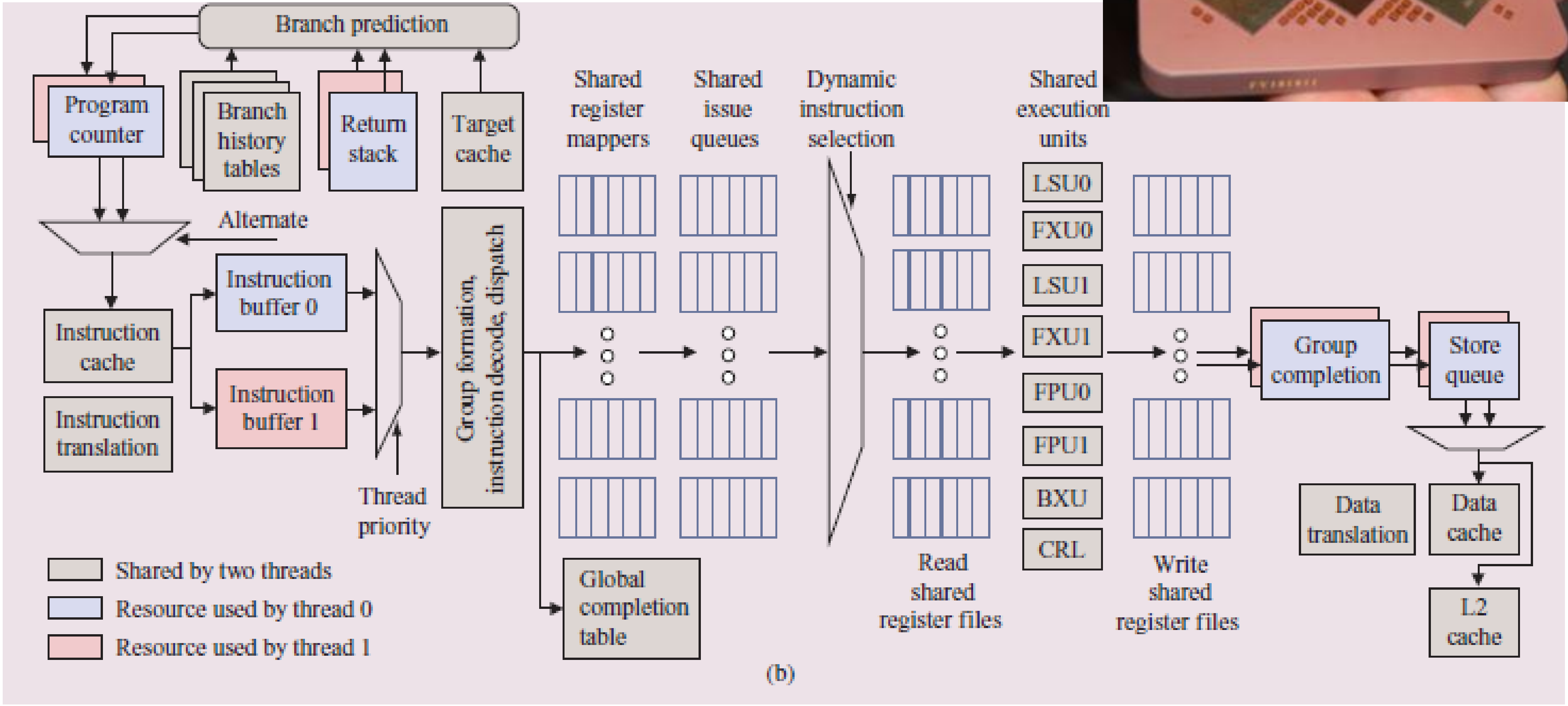


2 commits (architected register sets)

2 fetch (PC), 2 initial decodes

Power5 data flow ...





Why only 2 threads? With 4, one of the shared resources (physical registers, cache, memory bandwidth) would be prone to bottleneck

Changes in Power5 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 Power5 core is about 24% larger than the Power4 core because of the addition of SMT support