CS152 Discussion Section

Multithreading

Mar 18-22
Spring 2024

Administrivia

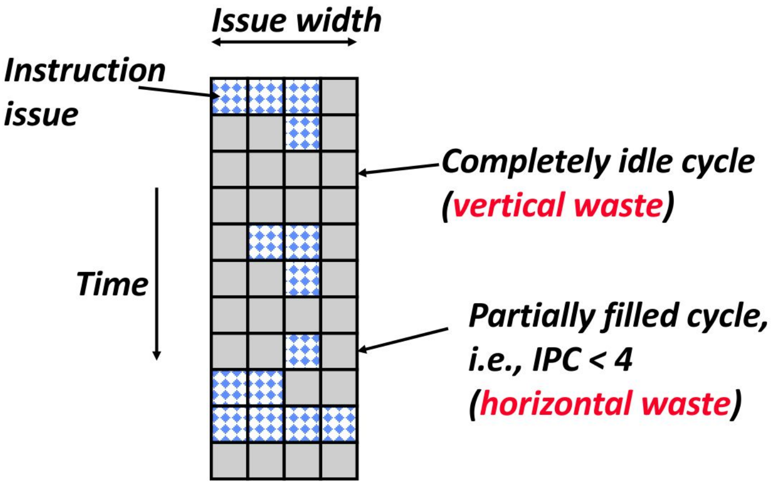
- Lab 3 due March 24
- HW4 coming out soon

Multithreading

Superscalar Limitations

Superscalar execution is limited by instruction dependencies (available parallelism) and long-latency operations in a single thread

- **Vertical waste**: No instructions issued in a cycle
- **Horizontal waste**: Not all issue slots filled in a cycle


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

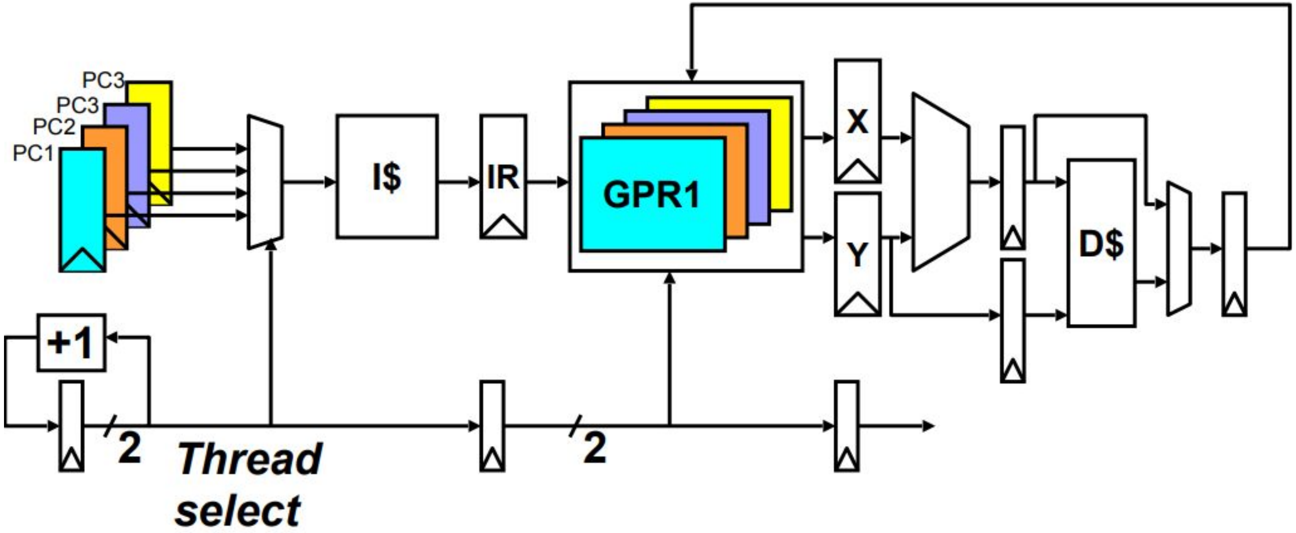
Q1. Comparing Architectures

	How is vertical waste reduced?	How is horizontal waste reduced?	Limitations / disadvantages compared to an in-order superscalar RISC machine?
Out-of-Order Superscalar Execution			
VLIW			
Vector			

Q1. Comparing Architectures

	How is vertical waste reduced?	How is horizontal waste reduced?	Limitations / disadvantages compared to an in-order superscalar RISC machine?
Out-of-Order Superscalar Execution	Out-of-order completion and speculative execution	Issue multiple non-consecutive instructions simultaneously	More complex/costly structures to manage control, more sensitive to misprediction
VLIW	Software pipelining, loop unrolling	Packing multiple operations into single instruction (Very Long Word)	Compiler is more complex and dependent on microarchitecture, static scheduling cannot easily tolerate variable latencies
Vector	Deep vector pipeline, chaining, multi-banked memory	Leveraging multiple vector lanes with longer vector lengths	Not all code is vectorizable, such as complex control flow

Multithreading

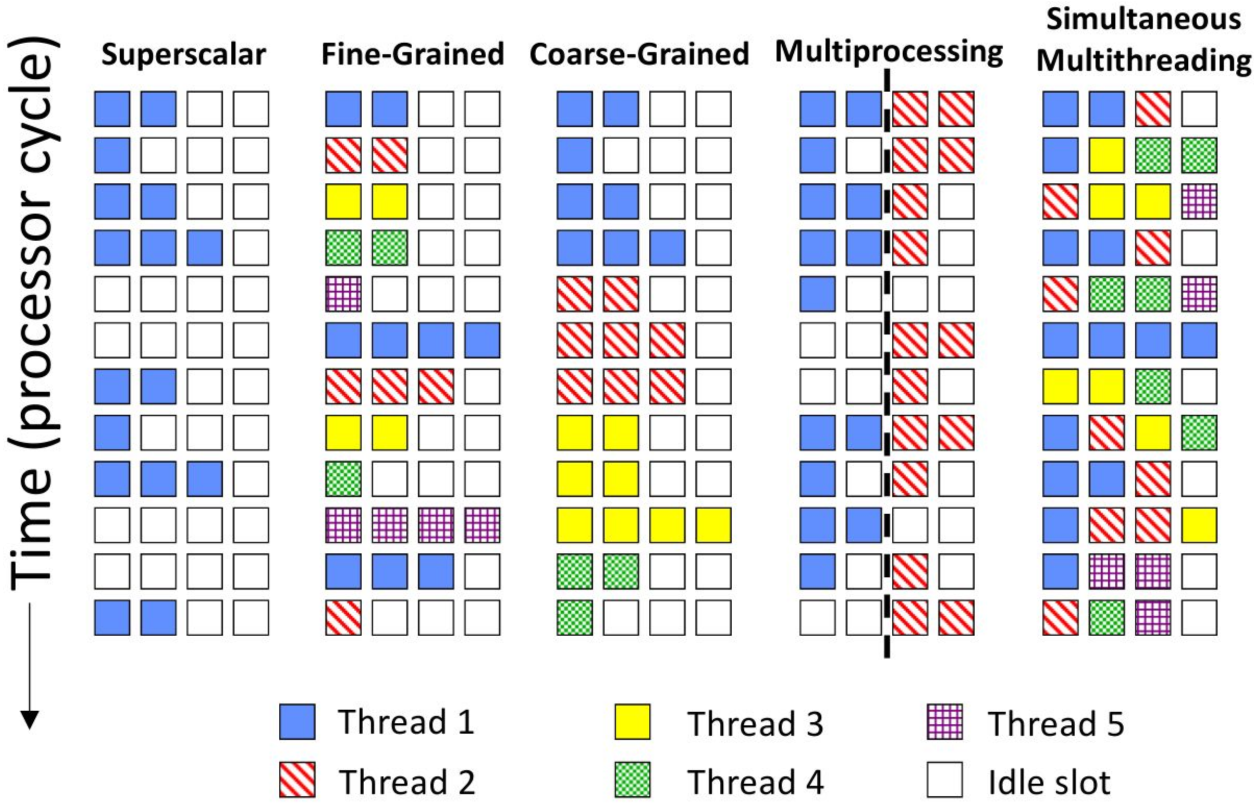


- Interleave execution of multiple threads (improve utilization of core; TLP)
- What is required in hardware to support multithreading?
    - Extra state (PC, GPR, PT base register, Exception-handling)

Multithreading

- Fine-grained multithreading
  - Switch between threads on each clock cycle
- Coarse-grained multithreading
  - Switch threads only on costly stalls
- Simultaneous multithreading
  - Interleave multiple threads in multiple issue slots with no restrictions

Multithreading



Q1. Comparing Architectures

	How is vertical waste reduced?	How is horizontal waste reduced?	Limitations / disadvantages compared to an in-order superscalar RISC machine?
Finegrained/Vertical Multithreading			
Simultaneous Multithreading			

Q1. Comparing Architectures

	How is vertical waste reduced?	How is horizontal waste reduced?	Limitations / disadvantages compared to an in-order superscalar RISC machine?
Finegrained/Vertical Multithreading	Interleaving instructions from multiple threads	Not reduced compared to superscalar issue	Low utilization if there are insufficient threads, more architectural state, resource contention, potentially lower single-thread performance
Simultaneous Multithreading	Same as vertical multithreading	Fetch / issue from multiple threads in same cycle	Same limitations and disadvantages as OoO execution and vertical multithreading

Q2. Fine-grained Multithreading

In this problem, we would like to investigate the performance of the following C program on a multithreaded architecture. The arrays A, B, and C contain double-precision floating-point numbers.

```
for (int i = 0; i < M; i++) {
    C[i] = A[i] + B[i];
}
```

```
loop: fld  f1, 0(x1)
      fld  f2, 0(x2)
      fadd f3, f1, f2
      fsd  f3, 0(x3)
      addi x1, x1, 8
      addi x2, x2, 8
      addi x3, x3, 8
      addi x4, x4, -1
      bnez x4, loop
```

Q2. Fine-grained Multithreading

We rewrite the loop to split the work across N threads:

```
// TID is the thread ID (0 to N-1)
for (int i = TID; i < M; i += N) {
    C[i] = A[i] + B[i];
}
```

```
loop: fld  f1, 0(x1)
      fld  f2, 0(x2)
      fadd f3, f1, f2
      fsd  f3, 0(x3)
      addi x1, x1, 8N
      addi x2, x2, 8N
      addi x3, x3, 8N
      addi x4, x4, -1
      bnez x4, loop
```

Assume:

- Single-issue in-order processor
- 1-cycle integer operations, 3-cycle floating-point arithmetic operations, 2-cycle memory operations
- Fine-grained multithreading with fixed round-robin scheduling
- Perfect branch prediction.

Q2. Fine-grained Multithreading

1. How many threads need to fully utilize the pipeline?

2. Peak performance in FLOPs/cycle?

3. Can peak performance be reached with fewer threads by reordering instructions in the loop?

Q2. Fine-grained Multithreading

1. How many threads need to fully utilize the pipeline?

   The greatest number of cycles between instructions is 3, the time between the **fadd** instruction and the dependent **fsd** instruction. **Three** threads are therefore needed to achieve full utilization.

2. Peak performance in FLOPs/cycle?

   1/9 = 0.11 FLOPs/cycle

3. Can peak performance be reached with fewer threads by reordering instructions in the loop?

   Yes. We can reach peak performance with just a **single** thread by moving one addi instruction between the second fld instruction and the fadd instruction and two addi instructions between the fadd and the fsd instruction.

Q3. Simultaneous Multithreading (SMT)

Which resources must be duplicated to support simultaneous multithreading?

Program Counter	
Fetch Unit	
Rename Table	
Physical Register File	
Issue Window	
Functional Units	
Reorder Buffer	

Q3. Simultaneous Multithreading (SMT)

Which resources must be duplicated to support simultaneous multithreading?

Program Counter	duplicated
Fetch Unit	shared
Rename Table	duplicated
Physical Register File	shared
Issue Window	shared
Functional Units	shared
Reorder Buffer	shared

Q3. Simultaneous Multithreading (SMT)

Icount policy prioritizes fetching from the thread with the least in-flight instructions

Why does this improve throughput?

Q3. Simultaneous Multithreading (SMT)

Icount policy prioritizes fetching from the thread with the least in-flight instructions

Why does this improve throughput?

If a thread has many instructions in flight, it is likely that it is **blocked on one of the instructions**. Therefore, adding more instructions for that thread will not make progress because they may also be blocked. It is better to fetch instructions for the thread with the fewest instructions in flight, since those instructions will be less likely to block.