CS152 Discussion Section

Midterm 2 Review

Week of April 1
Spring 2024

Midterm 2 Logistics

- When: next Thursday (April 11th) from 11-12:30pm (during class)
- Where: website will be updated with location

- Single-sheet letter-sized handwritten/typed cheatsheet allowed

Midterm 2 Topics

1. VIPT Caches
2. Out-of-Order Execution
    a. Superscalar, Branch Prediction
3. VLIW
4. Multithreading
5. Vector Architecture and GPUs

Cutoff: Anything covered in lecture up to (and including) April 2nd.

Review Exams:

- 2011 Quiz 3 and 4
- 2018 Midterm 2: Question 2

Superscalar OoO Processors and **Branch Prediction**

OoO Processor Checklist

- Register Renaming
  - Tomasulo vs data-in-ROB vs unified physical register file
- What does the ROB contain?
- How to maintain precise exceptions?
  - Out-of-order completion, in-order commit
- How to recover from mispredictions or exceptions?
  - Which structures need to be rolled back?
- Speculative memory operations:
  - Load/store queue, speculative store buffer, address speculation

Branch Prediction Checklist

- Temporal correlation (local history), spatial correlation (global history)
- Branch history table
  - 2-bit saturating counters
  - How is the BHT indexed?
- Two-level predictors
- Branch target buffer
  - What does the BTB contain?
- When are the BTB and BHT accessed in a pipeline?

OoO Pipeline (2011 Quiz 3)

Conceptual OoO pipeline stages: **Fetch**, **Decode**, **Register Rename**, **Dispatch**, **Issue**, **Execution**, **Writeback/Completion**, and **Commit**

- Which stage(s) allocate entries in the ROB?
- At which stage(s) is an entry in the ROB deallocated?
- At which stage(s) is an instruction allocated an entry in the issue window? (For a split issue window/ROB design)
- At which stage(s) is an instruction entry in the issue window deallocated?

OoO Pipeline (2011 Quiz 3)

Conceptual OoO pipeline stages: **Fetch**, **Decode**, **Register Rename**, **Dispatch**, **Issue**, **Execution**, **Writeback/Completion**, and **Commit**

- Which stage(s) allocate entries in the ROB?
  Dispatch

  Data-in-ROB: Rename/Dispatch

- At which stage(s) is an entry in the ROB deallocated?

  Commit

- At which stage(s) is an instruction allocated an entry in the issue window? (For a split issue window/ROB design)
  Dispatch
- At which stage(s) is an instruction entry in the issue window deallocated?

  Issue, but nuance due to misspeculation possibility

OoO Pipeline (2011 Quiz 3)

Conceptual OoO pipeline stages: **Fetch**, **Decode**, **Register Rename**, **Dispatch**, **Issue**, **Execution**, **Writeback/Completion**, and **Commit**

- At which stage(s) is a store entry allocated in the load/store queue?
- At which stage(s) is a load entry allocated in the load/store queue?
- At which stage(s) is a store performed (i.e., sent to the memory)?
- At which stage(s) is a load performed (i.e., when is data returned that can be used by other dependent instructions)?

OoO Pipeline (2011 Quiz 3)

Conceptual OoO pipeline stages: **Fetch**, **Decode**, **Register Rename**, **Dispatch**, **Issue**, **Execution**, **Writeback/Completion**, and **Commit**

- At which stage(s) is a store entry allocated in the load/store queue?
  Dispatch
- At which stage(s) is a load entry allocated in the load/store queue?
  Dispatch
- At which stage(s) is a store performed (i.e., sent to the memory)?
  Commit
- At which stage(s) is a load performed (i.e., when is data returned that can be used by other dependent instructions)?

  Execute

VLIW (Very Long Instruction Word)

VLIW Checklist

- Compare with OoO dynamic scheduling
  - Both improve ILP but in different ways
  - Qualitatively describe the similarities and differences
- Compilation techniques
  - Loop unrolling
  - Software pipelining
  - Trace scheduling
- Predication

Software Pipelining (2018 Midterm 2)

```
for (i = 0 ; i < N ; i++) { // imax
      if (max < l[i]) {
            idx = i;
            max = l[i];
      }
}
```

```
loop:
      fld f1, 0(a1)
      flt.d t1, f0, f1
      fmax.d f0, f0, f1
      beqz t1, skip
      addi s0, t0, 0
skip:
      addi a1, a1, 8
      addi t0, t0, 1
      bltu t0, a0, loop
```

`t0`	`i`
`s0`	`idx`
`a0`	`N`
`a1`	pointer to `l[i]`

- Two ALU units, 1-cycle latency
  - Also used for branch operations
- One memory unit, 2-cycle latency
  - Fully pipelined
- Two floating-point units, 3-cycle latency
  - Fully pipelined
  - Both can perform flt.d and fmax.d

Software Pipelining (2018 Midterm 2)

```
loop:
      fld f1, 0(a1)
      flt.d t1, f0, f1
      fmax.d f0, f0, f1
      beqz t1, skip
      addi s0, t0, 0
skip:
      addi a1, a1, 8
      addi t0, t0, 1
      bltu t0, a0, loop
```

`t0`	`i`
`s0`	`idx`
`a0`	`N`
`a1`	pointer to `l[i]`

Schedule VLIW instructions w/o pipelining

1. Identify data and control dependencies
    - fld -> flt.d (RAW)
    - beqz -> flt.d
    - addi t0 -> addi s0, t0
    - bltu -> addi t0

Naive Implementation

Label	ALU1	ALU2	MEM	FPU1	FPU2
loop:			fld f1, 0(a1)		
				fmax.d f0, f0, f1	flt.d t1, f0, f1
		beqz t1, skip			
		addi s0, t0, 0			
skip:	addi a1, a1, 8	addi t0, t0, 1			
		bltu t0, a0, loop			

```
loop:
      fld f1, 0(a1)
      flt.d t1, f0, f1
      fmax.d f0, f0, f1
      beqz t1, skip
      addi s0, t0, 0
skip:
      addi a1, a1, 8
      addi t0, t0, 1
      bltu t0, a0, loop
```

Loop inversion

Label	ALU1	ALU2	MEM	FPU1	FPU2
loop:	addi a1, a1, 8	addi t0, t0, 1	fld f1, 0(a1)		
				fmax.d f0, f0, f1	flt.d t1, f0, f1
		beqz t1, skip			
		addi s0, t0, -1			
skip:		bltu t0, a0, loop			

```
loop:
      fld f1, 0(a1)
      flt.d t1, f0, f1
      fmax.d f0, f0, f1
      beqz t1, skip
      addi s0, t0, 0
skip:
      addi a1, a1, 8
      addi t0, t0, 1
      bltu t0, a0, loop
```

Label	ALU1	ALU2	MEM	FPU1	FPU2
	addi a1, a1, 8	addi t0, t0, 1	fld f1, 0(a1)		
				fmax.d f0, f0, f1	flt.d t1, f0, f1
	addi a1, a1, 8	addi t0, t0, 1	fld f1, 0(a1)		
		beqz t1, skip		fmax.d f0, f0, f1	flt.d t1, f0, f1
		addi s0, t0, -1			
skip:					
		beqz t1, skip1			
		addi s0, t0, -1			
skip1:					

```
loop:
      fld f1, 0(a1)
      flt.d t1, f0, f1
      fmax.d f0, f0, f1
      beqz t1, skip
      addi s0, t0, 0
skip:
      addi a1, a1, 8
      addi t0, t0, 1
      bltu t0, a0, loop
```

Label	ALU1	ALU2	MEM	FPU1	FPU2
	addi a1, a1, 8	addi t0, t0, 1	fld f1, 0(a1)		
				fmax.d f0, f0, f1	flt.d t1, f0, f1
	addi a1, a1, 8	addi t0, t0, 1	fld f1, 0(a1)		
loop:		beqz t1, skip		fmax.d f0, f0, f1	flt.d t1, f0, f1
		addi s0, t0, -1			
skip:	addi a1, a1, 8	addi t0, t0, 1	fld f1, 0(a1)		
	bltu t0, a0, loop				
		beqz t1, skip1		fmax.d f0, f0, f1	flt.d t1, f0, f1
		addi s0, t0, -1			
skip1:					
		beqz t1, skip2			
		addi s0, t0, -1			
skip2:					

```
loop:
        fld f1, 0(a1)
        flt.d t1, f0, f1
        fmax.d f0, f0, f1
        beqz t1, skip
        addi s0, t0, 0
skip:
        addi a1, a1, 8
        addi t0, t0, 1
        bltu t0, a0, loop
```

Multithreading

Multithreading Checklist

- Different types of multithreading
  - Fine-grained, coarse-grained, simultaneous multithreading (SMT)
  - How many threads are required for different thread scheduling policies to hide significant latencies?

- Which microarchitectural components could be **shared** and which should be **dedicated per thread**?

Multithreading (2011 Quiz 4)

```
for (i = 0; i < N; i++) { // N = 1024
     S[i] = A[i] * B[i] + Y[i];
}
```

```
     addi $n, $0, 1024
     addi $i, $0, 0
loop:ld   $a, A($i)
     ld   $b, B($i)
     fmul $t, $a, $b
     ld   $y, Y($i)
     fadd $s, $t, $y
     sd   $s, S($i)
     addi $i, $i, 8
     addi $n, $n, -1
     bnez $n, loop
```

- In-order issue
- There is no cache; each memory operation directly accesses main memory and takes 50 cycles
- The load/store unit is fully pipelined
- After the processor issues a memory operation, it can continue executing instructions until it reaches an instruction that is dependent on an outstanding memory operation
- The `fmul` and `fadd` instructions both have a latency of 5 cycles

Multithreading (2011 Quiz 4)

Suppose threads are switched every cycle in a **fixed round-robin schedule**.

What is the minimum number of threads in steady state needed to fully utilize the processor?

You may reschedule the code as necessary to minimize the number of threads required.

```
     addi $n, $0, 1024
     addi $i, $0, 0
loop:ld   $a, A($i)
     ld   $b, B($i)
     fmul $t, $a, $b
     ld   $y, Y($i)
     fadd $s, $t, $y
     sd   $s, S($i)
     addi $i, $i, 8
     addi $n, $n, -1
     bnez $n, loop
```

Multithreading (2011 Quiz 4)

Suppose threads are switched every cycle in a **fixed round-robin schedule**.

What is the minimum number of threads in steady state needed to fully utilize the processor?

You may reschedule the code as necessary to minimize the number of threads required.

(5N + 1) - (N + 1) >= 50

N = ceil(50/4) = 13

```
      addi $n, $0, 1024
      addi $i, $0, 0
loop: ld   $a, A($i)     # cycle 1
      ld   $b, B($i)     #   N + 1
      ld   $y, Y($i)     #  2N + 1
      addi $i, $i, 8     #  3N + 1
      addi $n, $n, -1    #  4N + 1
      fmul $t, $a, $b    #  5N + 1
      fadd $s, $t, $y    #  6N + 1
      sd   $s, S-8($i)   #  7N + 1
      bnez $n, loop      #  8N + 1
```

Multithreading (2011 Quiz 4)

Suppose threads are switched whenever there is a **data-dependent stall**.

What is the minimum number of threads in steady state needed to fully utilize the processor?

You may reschedule the code as necessary to minimize the number of threads required.

```
loop:ld   $a, A($i)
     ld   $b, B($i)
     ld   $y, Y($i)
     addi $i, $i, 8
     addi $n, $n, -1
     …
     fmul $t, $a, $b
     …
     fadd $s, $t, $y
     …
     sd   $s, S-8($i)
     bnez $n, loop
```

Multithreading (2011 Quiz 4)

Suppose threads are switched whenever there is a **data-dependent stall**.

What is the minimum number of threads in steady state needed to fully utilize the processor?

You may reschedule the code as necessary to minimize the number of threads required.

Each thread does 7 instructions in steady-state

Longest stall is 46 cycles

ceil(46/7) + 1= 8 threads total

```
loop:ld   $a, A($i)
     ld   $b, B($i)
     ld   $y, Y($i)
     addi $i, $i, 8
     addi $n, $n, -1
     ...
     fmul $t, $a, $b
     ...
     fadd $s, $t, $y
     ...
     sd   $s, S-8($i)
     bnez $n, loop
```

Vectors and GPUs

Vector and GPUs Checklist

- Vectorization techniques
  - Stripmining, vector masks, vector reductions, vector strided loads/stores, vector scatter/gather
  - Loop-carried dependencies
- Vector performance characteristics
  - Lanes, chaining, dead time
- Compare and contrast with packed SIMD and GPUs

Vectorization (2011 Quiz 4)

Addi x0

Addi … x0

Which loops can be safely vectorized?

1.
```
for (i = 0; i < N; i++)
        A[i] = A[i] + B[i];
```

2.
```
for(i = 0; i < N; i++)
        A[i] = A[i+1] + B[i];
```

3.
```
for(i = 0; i < N; i++)
        A[i] = A[i-1] + B[i];
```

Vectorization (2011 Quiz 4)

Addi x0

Addi … x0

Which loops can be safely vectorized?

1.
```
for (i = 0; i < N; i++)
        A[i] = A[i] + B[i];
```
Yes

2.
```
for(i = 0; i < N; i++)
        A[i] = A[i+1] + B[i];
```
Yes

3.
```
for(i = 0; i < N; i++)
        A[i] = A[i-1] + B[i];
```
No