Lecture 23
Thread-Level Speculation

I. Overview & System Requirements
II. Compiler Support for Scalar Forwarding
III. Scheduling Instructions
   • Stack Analysis
   • Earliest Analysis
IV. Speculative Scheduling

Automatic Parallelization

Proving independence of threads is hard:
- complex control flow
- complex data structures
- pointers, pointers, pointers
- run-time inputs

How can we make the compiler’s job feasible?

**Thread-Level Speculation (TLS)**
while (...){
    x = hash[index1];
    ...
    hash[index2] = y;
    ...
}
Example of Thread-Level Speculation

Processor

Epoch 1
= hash[3]
...
hash[10] =
...

Processor

Epoch 2
= hash[19]
...
hash[21] =
...

Processor

Epoch 3
= hash[33]
...
hash[30] =
...

Processor

Epoch 4
= hash[10]
...
hash[25] =
...

Time
Example of Thread-Level Speculation

Epoch 1
= hash[3]
...

Epoch 2
= hash[19]
Violation!

Epoch 3
= hash[33]
...

Epoch 4
= hash[10]
...

Processor

Processor

Processor

Processor

Time
Example of Thread-Level Speculation

**Epoch 1**
- \( \text{hash}[3] \)
- ...\( \text{hash}[10] \) = ...
- commit? ✓

**Epoch 2**
- \( \text{hash}[19] \)
- ...\( \text{hash}[21] \) = ...
- commit? ✓

**Epoch 3**
- \( \text{hash}[33] \)
- ...\( \text{hash}[30] \) = ...
- commit? ✓

**Epoch 4**
- \( \text{hash}[10] \)
- ...\( \text{hash}[25] \) = ...
- commit? ×

Time ↓
Example of Thread-Level Speculation

Processor
Epoch 1
= hash[3]
... 
hash[10] = ...
commit?

Epoch 2
= hash[19]
... 
hash[21] = ...
commit?

Epoch 3
= hash[33]
... 
hash[30] = ...
commit?

Epoch 4
= hash[10]
... 
hash[25] = ...
commit?

Time

Processor

Violation!

Retry

✓

✓

✓

✓
System Requirements for Thread-Level Speculation

System requirements:

1) Detect data dependence violations
   • extend invalidation-based cache coherence

2) Buffer speculative modifications
   • use the caches as speculative buffers
Life Cycle of an Epoch

- **Spawned**
- **Becomes Speculative**
- **Commit?**
- **Complete, Pass Homefree token**

**Speed-up depends on commit times**

- **Slow Commit:**
- **Fast Commit:**
Another Limitation on Speed-up: Possible Dependences

- Good when $p \neq q$
- Bad when $p = q$, will cause violation
Solution: Add Synchronization and Value Forwarding

\[ T_i \quad \text{store} \quad *p \quad \text{load} \quad *q \]

(Speculation)

\[ T_i \quad \text{store} \quad *p \quad \text{signal} \]

\[ T_{i+1} \quad \text{wait} \quad (\text{stall}) \quad \text{load} \quad *q \]

Memory

\[ \text{good when } p == q \]
II. Compiler Support for Thread-Level Speculation

Identifying Forwarding Scalar

... = a

a = ...;

Inserting Wait/Signal

Scheduling Instructions

Scheduling Instructions Speculatively
Compiler’s Tasks

- Identifying Forwarding Scalar
- Inserting Wait/Signal
- Scheduling Instructions Speculatively
Synchronizing Scalars

\[ T_i \]

\[ \ldots = a \]

\[ a = \ldots \]

\[ \text{wait}(a) \]

\[ \text{signal}(a) \]

\[ T_{i+1} \]

\[ \ldots = a \]

\[ a = \ldots \]

\[ a > 0 \]

\[ a = a \times 7 \]

\[ \text{wait}(a); \]

\[ a > 0 \]

\[ a = a \times 7 \]

\[ \text{signal}(a) \]
The Critical Forwarding Path

Shorter critical forwarding path → less execution time
Reducing the Critical Forwarding Path

Instruction scheduling can reduce critical forwarding path
Compiler’s Tasks

- Identifying Forwarding Scalar
- Inserting Wait/Signal
- Scheduling Instructions Speculatively

\[ ... = a \]
\[ a = ...; \]
\[ wait(a); \]
\[ ... = a \]
\[ a = ...; \]
\[ signal(a); \]
\[ wait(a); \]
\[ ... = a \]
\[ a = ...; \]
\[ signal(a); \]
III. Scheduling Instructions

**Dataflow analysis**

Handles complex control flow

**Define two dataflow analyses**

- **Stack**
  
  Find the instructions to compute the forwarded value?

- **Earliest**
  
  Find the earliest node to compute the forwarded value?
**Computation Stack**

Stores the instructions to compute a forwarded value

Associating a stack with every node for every forwarded scalar

---

| a = a * 11 | signal a |

We know how to compute the forwarded value

|              |

We don’t know how to compute the forwarded value

|              |

We have not evaluated this node
A Simplified Example from GCC

do {
    ...
} while (p);

start

wait (p)
counter = 0
p -> jmp?
p = p -> jmp

q = p
p -> real?
q = q -> next;
counter++; q?
p = p -> next

end

20
Stack Analysis

start

wait(p)
counter=0
p->jmp?

p=p->jmp

p->real?

q=p

p=p->next

counter++;
q=q->next;
q?

p=p->next

signal p

end

wait(p)

p=p->next

counter=0
Stack Analysis

start

wait(p)
counter=0
p->jmp?

p=p->jmp

q = p
p->real?

p->next
signal p

p=p->next

p=p->next

q=q->next;
q?

counter++;"counter++";

end

p = p->next

signal p

15-745: Thread Level Speculation
Stack Analysis

start

wait(p)
counter = 0
p -> jmp?

p = p -> jmp

q = p
p -> real?

q = p
p -> real?

counter++;
q = q -> next;
q?

p = p -> next

signal p

signal p

end

p = p -> next

p = p -> next

signal p
Stack Analysis

```
p = p - > next
signal p

p = p - > next
signal p

counter++; q = q - > next; q?

p = p - > next

end

signal p

p = p - > next
signal p
```
Stack Analysis

start

wait(p)

counter=0

p->jmp?

p=p->jmp

q = p

p->real?

p=p->next

signal p

end

counter++;

q=q->next;

q?

p = p->next

signal p

p=p->next

signal p

p=p->next

signal p

p=p->next

signal p
Stack Analysis

start

\[ p = p - > \text{jmp} \]

\[ q = p \]

\[ p = p - > \text{real?} \]

\[ p = p - > \text{next} \]

\[ \text{wait}(p) \]

\[ \text{counter} = 0 \]

\[ p = p - > \text{next} \]

\[ \text{signal} \ p \]

\[ \text{counter}++ \]

\[ q = q - > \text{next} \]

\[ q \]

end

\[ p = p - > \text{jmp} \]

\[ p = p - > \text{next} \]

\[ \text{wait}(p) \]

\[ p = p - > \text{next} \]

\[ \text{signal} \ p \]

\[ p = p - > \text{next} \]

\[ \text{signal} \ p \]

\[ p = p - > \text{next} \]

\[ \text{signal} \ p \]

\[ p = p - > \text{next} \]

\[ \text{signal} \ p \]

\[ p = p - > \text{next} \]

\[ \text{signal} \ p \]

\[ p = p - > \text{next} \]

\[ \text{signal} \ p \]

15-745: Thread Level Speculation
Stack Analysis

start

wait(p)
counter=0
p->jmp?

p=p->jmp

p=p->next
signal p

q = p
p->real?

end

p=p->next
signal p

signal p

counter++;
q=q->next;
q?

p=p->next
signal p

p=p->next
signal p

signal p

15-745: Thread Level Speculation
Stack Analysis

start

wait(p)

p=p->jmp

p=p->next

signal p

p=p->jmp

q=p

p->real?

p=p->next

signal p

p=p->next

signal p

q=q->next;

q?

counter++;

p=p->next

signal p

p=p->next

signal p

p=p->next

signal p

end

signal p
Stack Analysis

```
start

wait(p)
counter=0
p->jmp?
p=p->next
q=p

p=p->jmp

signal p

p=p->next
q=p

p=p->next
signal p

p=p->next
signal p

p=p->next

signal p

counter++
q=q->next;

signal p

p=p->next

signal p

p=p->next

wait(p)
counter=0
p->jmp?
p=p->next

signal p

p=p->next

signal p

p=p->next

signal p

p=p->next

signal p

p=p->next

signal p

signal p

p=p->next

signal p

signal p

end

15-745: Thread Level Speculation
```
Stack Analysis

start

wait(p)
\text{counter} = 0
\text{p -> jmp?}

p = p -> jmp

\text{q = p}
\text{p -> real?}

p = p -> next
\text{signal p}

p = p -> next
\text{signal p}

p = p -> next
\text{signal p}

\text{counter++;
q = q -> next;
q?}

\text{p = p -> next}

\text{signal p}

end

\text{p = p -> next}
\text{signal p}

\text{p = p -> next}
\text{signal p}

\text{p = p -> next}
\text{signal p}
Stack Analysis

start

\[ p = p \rightarrow \text{jmp} \]
\[ p = p \rightarrow \text{next} \]
\[ \text{signal p} \]

\[ \text{wait}(p) \]
\[ \text{counter} = 0 \]
\[ p \rightarrow \text{jmp} ? \]

\[ q = p \]
\[ p \rightarrow \text{real} ? \]

\[ p = p \rightarrow \text{jmp} \]

\[ p = p \rightarrow \text{next} \]
\[ \text{signal p} \]

\[ q = q \rightarrow \text{next} \]
\[ q ? \]

\[ p = p \rightarrow \text{next} \]
\[ \text{counter}++ \]
\[ \text{signal p} \]

\[ p = p \rightarrow \text{next} \]
\[ \text{signal p} \]

\[ p = p \rightarrow \text{next} \]
\[ \text{signal p} \]

\[ \text{signal p} \]

end

\[ p = p \rightarrow \text{next} \]
\[ \text{signal p} \]
The Solution is Consistent

```
p = p->next
signal p
```

```
p = p->next
signal p
```

```
p = p->next
signal p
```

```
p = p->next
signal p
```

```
counter++;
q = q->next;
q?
```

```
p = p->next
```

```
p = p->next
```

```
wait(p)
```

```
counter=0
p->jmp
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```

```
signal p
```
**Scheduling Instructions**

**Dataflow analysis**

Handles complex control flow

**Define two dataflow analyses**

**Stack**

Find the instructions to compute the forwarded value?

**Earliest**

Find the earliest node to compute the forwarded value?
The Earliest Analysis

Start

wait(p) counter=0
p->jmp?

p=p->jmp

q=p
p->real?

p=p->jmp

q=q
p=p->next

counter++;
q=q->next;
q?

p=p->next

p=p->next

p=p->next

end
The Earliest Analysis

start

wait(p)
counter=0
p->jmp?
p

q = p
p->real?
p

counter++;
q=q->next;
q?
p = p->next

end
The Earliest Analysis

- `p = p -> jmp` and `signal p`
- `p = p -> next` and `signal p`
- `counter++` and `q = q -> next` and `q?`
- `p = p -> next` and `signal p`
- `end`

Earliest
Not earliest
Not evaluated
The Earliest Analysis

start

\text{wait}(p) \quad \text{counter}=0
\quad p=\text{next} > \text{real}?

q = p
\quad p=\text{next} > \text{real}?

p=\text{next} > \text{real}?
\quad \text{counter}++;
\quad q=q->\text{next};
\quad q?

\quad p=\text{next} > \text{real}?

\quad p=\text{next} > \text{real}?

\quad p=\text{next} > \text{real}?

end
The Earliest Analysis

start

wait(p) counter=0
p->jmp?

q = p
p->real?

q
p->jmp

end

p=p->jmp

p=p->next

signal p

p=p->next

signal p

counter++;
qu=q->next;
q?

p = p->next

p=p->next

signal p

p=p->next

signal p

p=p->next

signal p

signal p
start

wait(p) counter=0
p->jmp?

p2=p->jmp
p1=p->next
Signal(p1)

q = p
p->real?

p=p->next

p1=p->next
Signal(p1)

counter++; q=q->next;
q?

p = p->next

end
Experimental Framework

Benchmarks
  – from SPECint95 and SPECint2000, -O3 optimization

Underlying architecture
  – 4-processor, single-chip multiprocessor
  – speculation supported by coherence

Simulator
  – superscalar, similar to MIPS R10K
  – models all bandwidth and contention

 détail simulation!
Benefit of Instruction Scheduling

*U* = TLS but no Instruction Scheduling
*A* = TLS with Instruction Scheduling

(Normalized to sequential execution time)

Improves performance by 18% over no instruction scheduling
IV. Compiler’s Tasks: Speculative Scheduling

Identifying Forwarding Scalar

Inserting Wait/Signal

Scheduling Instructions

Scheduling Instructions Speculatively

\[ ... = a \]
\[ a = ...; \]
\[ a = ...; \]
\[ \text{signal}(a); \]
\[ \text{wait}(a); \]
\[ \text{signal}(a); \]
\[ a = ...; \]
\[ \text{wait}(a); \]
\[ a = ...; \]
\[ \text{signal}(a); \]
\[ a = \ast q; \]
Speculating Beyond a Control Dependence

Usual Path

\[ p = p \rightarrow \text{jmp} \]
\[ \text{signal} \ p \]
\[ p = p \rightarrow \text{jmp} \]
\[ p = p \rightarrow \text{next} \]
\[ \text{signal} \ p \]
\[ \text{signal} \ p \]
\[ \text{end} \]
Speculatively Scheduling Instructions Across Control Dependencies

No significant performance gain

A = Instruction Scheduling
C = Speculating Across Control Dependences

Carnegie Mellon
Speculating Beyond a Potential Data Dependence

```
q->next = NULL
p = p->next
signal p
p = p->next
signal p
end
```

```
p = load(addr1);
signal(p);
store(addr2);
```

Hardware Support Needed
Aggressively Scheduling Instructions Across Both Control and Data Dependences

A = Instruction Scheduling
C = Speculating Across Control Dependences
D = Speculating Across Control & Data Dependences

Improves performance significantly for some benchmarks
Summary: Compiler Support for Thread-Level Speculation

- Identifying Forwarding Scalar
- Inserting Wait/Signal
- Scheduling Instructions
- Scheduling Instructions Speculatively
Conclusions

Instruction scheduling for reducing synchronization

- Is effective in reducing critical forwarding path
  - Performance improved by 18%
- Is beneficial to handle complex control flow, such as inner loops
  - Improved GCC by 3%
- Gives additional benefit with speculative instruction scheduling
  - One biggest benefactor is GCC, performance improved by 18%
  - Some hardware support needed

Critical forwarding path can be addressed by the compiler
Today’s Class: Thread Level Speculation

I. Overview & System Requirements
II. Compiler Support for Scalar Forwarding
III. Scheduling Instructions
   • Stack Analysis
   • Earliest Analysis
IV. Speculative Scheduling

Coming Attractions

• No more class lectures
• Friday 4/14: Project Milestone Reports due at midnight
• Wednesday 4/19: In-class Exam
• Friday 5/5: Project Poster Session