Speculative Call Stacks

Chris Pickett    Clark Verbrugge
School of Computer Science, McGill University
{cpicke,clump}@sable.mcgill.ca

Allan Kielstra
IBM Toronto Lab
kielstra@ca.ibm.com

January 25th, 2008

Sable Research Group
Method Level Speculation

Before:
- INVOKE
- RETURN

After:
- INVOKE
- RETURN
- fork
- join
- T2
This representation works fine for a simple invocation:

```java
... // non-speculative parent
a(); // non-speculative parent
... // speculative child
```
This representation works fine for a simple invocation:

But how do we handle more complex things?

```java
... // non-speculative parent
a() {
  ... // non-speculative parent
  b();
  ...
}
...
// speculative child
c();
...
```
This representation works fine for a simple invocation:

But how do we handle more complex things?

...    // non-speculative parent
a() { ...  // non-speculative parent
  b();    // can the parent create a new child here?
... }    // is this part executed by the second child?
...     // speculative child
c();
This representation works fine for a simple invocation:

But how do we handle more complex things?

```java
... // non-speculative parent
a() { ...
    // non-speculative parent
    b(); // can the parent create a new child here?
... } // is this part executed by the second child?
...
    // speculative child
    c(); // can the child create its own child here?
... // is this part ‘doubly-speculative’?
```
Remainder of this talk:

- Introduce notion of “speculative call stacks”
- Explore all possible stack operations
- Develop intuition with respect to speculative process tree behaviour
1. Method Level Speculation
2. Parent Thread Operations
3. Child Thread Operations
4. Out-of-Order Nesting
5. In-Order Nesting
6. Combined In-Order and Out-of-Order Nesting
7. Conclusions and Future Work
1. push (P)
1. push (P)
Parent Thread Operations

1. a
2. pop (P)
Parent Thread Operations

2. pop (P)
3. fork (P)
3. fork (P)
Parent Thread Operations

4. fork (P), push (P)
4. fork (P), push (P)
Parent Thread Operations

5. abort (P)
5. abort (P)
Parent Thread Operations

6. pop (P), commit (P)
Parent Thread Operations

6. pop (P), commit (P)
Parent Thread Operations

1. push (P)

2. pop (P)

3. fork (P)

4. fork (P), push (P)

5. abort (P)

6. pop (P), commit (P)
1 Method Level Speculation
2 Parent Thread Operations
3 Child Thread Operations
4 Out-of-Order Nesting
5 In-Order Nesting
6 Combined In-Order and Out-of-Order Nesting
7 Conclusions and Future Work
1. push (C1)
Child Thread Operations

1. push (C1)
Child Thread Operations

2. pop (C1)
Child Thread Operations

2. pop (C1)
Child Thread Operations

3. pop (C1)
Child Thread Operations

3. pop (C1)
4. pop (P), commit (P)
Child Thread Operations

4. pop (P), commit (P)
Child Thread Operations

1. push (C1)

2. pop (C1)

3. pop (C1)

4. pop (P), commit (P)
Out-of-Order Nesting

1. fork (P), push (P)
Out-of-Order Nesting

1. fork (P), push (P)
Out-of-Order Nesting

2. push (C1)
Out-of-Order Nesting

2. push (C1)
Out-of-Order Nesting

3. pop (C2)
Out-of-Order Nesting
Out-of-Order Nesting

4. pop (P), commit (P)
Out-of-Order Nesting

4. pop (P), commit (P)
Out-of-Order Nesting

1. fork (P), push (P)

2. push (C1)

3. pop (C2)

4. pop (P), commit (P)
Outline

1. Method Level Speculation
2. Parent Thread Operations
3. Child Thread Operations
4. Out-of-Order Nesting
5. In-Order Nesting
6. Combined In-Order and Out-of-Order Nesting
7. Conclusions and Future Work
1. fork (C1), push (C1)
1. fork (C1), push (C1)
In-Order Nesting

2. pop (P), abort (P)
In-Order Nesting

2. pop (P), abort (P)
In-Order Nesting

3. pop (C2)
In-Order Nesting

4. pop (C2)
In-Order Nesting

1. Push (P)
2. C1
3. C2
4. pop (C2)
In-Order Nesting

5. pop (P), commit (P)
In-Order Nesting

5. pop (P), commit (P)
In-Order Nesting

6. pop (P), commit (P)
In-Order Nesting

6. pop (P), commit (P)
In-Order Nesting

7. pop (C2)
In-Order Nesting

7. pop (C2)
In-Order Nesting

1. fork (C1), push (C1)
2. pop (P), abort (P)
3. pop (C2)
4. pop (C2)
In-Order Nesting

5. pop (P), commit (P)

6. pop (P), commit (P)

7. pop (C2)
Combined In-Order and Out-of-Order Nesting

a
P

fork (P)
push (P)
Combined In-Order and Out-of-Order Nesting

fork (P)  push (P)  fork (C1)  push (C1)
Combined In-Order and Out-of-Order Nesting
Combined In-Order and Out-of-Order Nesting

fork (P) push (P) fork (C1) push (C1) fork (C1) push (C1) push (C3)
Combined In-Order and Out-of-Order Nesting

fork (P) push (P) fork (C1) push (C1) fork (C1) push (C1) push (C3) pop (P) commit (P) pop (P) commit (P)
Combined In-Order and Out-of-Order Nesting
WHO EVEN CARES???
People with multiprocessors care:
- lots of cores, lots of single-threaded applications, lots of idle time
People with multiprocessors care:
- lots of cores, lots of single-threaded applications, lots of idle time

Implementors care:
- “Any technique that implements a form of TLS will need to abide by similar rules outlined by the authors. Then, why is this abstraction useful?” – non-testimonial from reviewer
People with multiprocessors care:

- lots of cores, lots of single-threaded applications, lots of idle time

Implementors care:

- “Any technique that implements a form of TLS will need to abide by similar rules outlined by the authors. Then, why is this abstraction useful?” – non-testimonial from reviewer

Academics care:

- We can use this to understand method speculation proposals
Future Work

- Measure extent of dynamic in-order and out-of-order nesting
- Formalize abstraction as small-step operational semantics
- Use formal abstraction to prove:
  - Behaviour is equivalent to sequential execution
  - Correctness of thread fork/join algorithm