Software Speculative Multithreading for Java

Christopher J. F. Pickett
cpicke@sable.mcgill.ca

Sable Research Group
School of Computer Science
McGill University

Friday the 13th
November, 2009

Ph.D. Proposal
Motivation

Do this for Java, in software, automatically, at the VM level

Why is this a good problem?

- multicore everywhere
- lots of deployed sequential code out there
- manual parallelization is error-prone
- novel hardware is expensive
- Java programs tend to be irregular
Motivation

Do this for Java, in software, automatically, at the VM level

Why is this a good problem?
- multicore everywhere
- lots of deployed sequential code out there
- manual parallelization is error-prone
- novel hardware is expensive
- Java programs tend to be irregular

Why is this a hard problem?
- Java programs tend to be irregular
- Java is a rich and complex language
- no source code annotations
- software overheads are high
- nobody has built a working system yet
Related Work

Hardware findings:
- Method speculation subsumes loop speculation (Chen, PACT’98)
- Method speculation is viable for Java in hardware (Chen, ISCA’03)
- Return value prediction helps Java method speculation (Hu, JILP’03)
- Fork heuristics help Java method speculation (Whaley, ICPP’05)

Software findings:
- Loop speculation is viable in software (Cintra, TPDS’05)
- Speculative (“safe”) futures work for Java in software (Welc, OOPSLA’05)
- Coarse-grained speculation is viable in software (Ding, PLDI’07)
- Method speculation works for Haskell in software (Harris, ICFP’07)

Our work is differentiated by being:
- software based
- virtual machine based
- able to handle complex benchmarks with side effects
Initial System and Profiling

initial system
Initial System and Profiling

- initial system
- dependence buffering
Initial System and Profiling

- Initial system
  - dependence buffering
  - return value prediction
Initial System and Profiling

- Initial system
- Dependence buffering
- Return value prediction
- Method level speculation
Initial System and Profiling

- VM design
- Dependence buffering
- Return value prediction
- Method level speculation
Initial System and Profiling

VM design

Java language considerations

dependence buffering

initial system

method level speculation

return value prediction
Initial System and Profiling

- VM design
- fork heuristics
- Java language considerations
- dependence buffering
- method level speculation
- return value prediction
Initial System and Profiling

- VM design
- fork heuristics
- Java language considerations
- method level speculation
- dependence buffering
- return value prediction

profiling
Initial System and Profiling

VM design
fork heuristics
Java language considerations
method level speculation
dependence buffering
return value prediction

profiling

high RVP overhead
Initial System and Profiling

- VM design
- fork heuristics
- Java language considerations
- method level speculation
- dependence buffering
- return value prediction

profiling

- high RVP overhead
- idle processors
Initial System and Profiling

VM design
fork heuristics

Java language considerations
method level speculation

dependence buffering
return value prediction

initial system

profiling

high RVP overhead
idle processors
short threads
| high RVP overhead | idle processors | short threads |
Profiling and Optimization

- high RVP overhead
- idle processors
- short threads

adaptive RVP
Profiling and Optimization

- high RVP overhead
  - adaptive RVP
- idle processors
  - nested speculation
- short threads
Profiling and Optimization

- high RVP overhead
  - adaptive RVP
- idle processors
  - nested speculation
- short threads
  - improved heuristics
Profiling and Optimization

- High RVP overhead
- Idle processors
- Short threads

- Adaptive RVP
- Nested speculation
- Improved heuristics

???

PROFIT!
SableVM (McGill)

research
Java bytecode
interpreter
// execute foo() non-speculatively
r = foo(a, b, c);
Example

// execute foo() non-speculatively
r = foo(a, b, c);

// execute foo()’s continuation speculatively?
Fork Heuristics

Capture information at probes:
- parent and child thread lengths
- speculation success rates
- various support component statistics

Process information, compute score, create child if profitable
// execute foo() non-speculatively
r = foo (a, b, c);

// execute foo()’s continuation speculatively
if (r > 10) // predict return value
{
    ...
}

Return Value Prediction Framework

Last Value Predictor

example: 1, 2, 3 . . . 3

\[ v_n = v_{n-1} \]

fields: last
predict():
return last
update(return_value):
last = return_value

Stride Predictor

example: 1, 2, 3 . . . 4

\[ v_n = v_{n-1} + (v_{n-1} - v_{n-2}) \]

fields: last, stride
predict():
return last + stride
update(return_value):
stride = return_value - last
last = return_value
Table-based Predictors

INVOKEx

T1

return value history (context)
r1, r2, r3, r4, r5

hash

word-sized hash value

context table lookup

prediction

Java stack

objectref

p1, p2, pN

hash

word-sized hash value

memoization table lookup

prediction
Adaptive Hybrid Predictor

Object-oriented approach:
- one hybrid instance per fork point
- online specialization and despecialization

Optimize for:
- accuracy
- speed
- memory
Parameter dependence:

```c
foo (int a, int b, int c) {
    return a + c;
}
```

Return value use:

```c
r = foo (a, b, c);
if (r > 10) {
    ...
}
```
// execute foo() non-speculatively
r = foo(a, b, c);

// execute foo()’s continuation speculatively
if (r > 10)  // predict return value
{
    s = o1.f;  // buffer heap & static reads
    o2.f = r;  // buffer heap & static writes
}
Dependence Buffering

speculative Java stack

write

write buffer

read buffer

main memory

Java heap values and class statics

commit: if read buffer is valid
flush write buffer
// execute foo() non-speculatively
r = foo (a, b, c);

// execute foo()'s continuation speculatively
if (r > 10)  // predict return value
{
    s = o1.f;  // buffer heap & static reads
    o2.f = r;  // buffer heap & static writes
}

// invoke bar() speculatively
o3.bar();
Nested Method Level Speculation
Nested Method Level Speculation

call stack growth

b

a

P
Nested Method Level Speculation
Nested Method Level Speculation
Nested Method Level Speculation

Call stack growth

P

C1

C2
Nested Method Level Speculation

call stack growth
Nested Method Level Speculation
Nested Method Level Speculation

![Diagram of call stack growth with labels a, a', b, c, f, e, C1, C2, C3]
Nested Method Level Speculation
Nested Method Level Speculation
Nested Method Level Speculation
Nested Method Level Speculation
Nested Method Level Speculation

call stack growth

f
a''
P
Nested Method Level Speculation

call stack growth

\[ a'' \]

P
Child Thread Memory Management

1. If (empty) swap full for empty
2. If (full) swap full for empty
1. Add child
2. Remove child

- Child
- Thread block
- Runtime full blocks
- Runtime empty blocks
- Free child
- Alloc child
// execute foo() non-speculatively
r = foo (a, b, c);

// execute foo()'s continuation speculatively
if (r > 10) // predict return value
{
    s = o1.f; // buffer heap & static reads
    o2.f = r; // buffer heap & static writes
}

// invoke bar() speculatively
o3.bar();

// stop speculation
synchronized (o4) { ... }

Example
Stop speculation or abort threads when problems arise with:

- class loading
- object allocation
- garbage collection
- native methods
- exception handling
- synchronization
- memory model constraints
- bytecode verifiability

Otherwise stop when signalled or expected thread length is met.
General Evaluation Metrics

- **speedup**: sequential runtime / parallel runtime
- **overhead**: non-useful work / total work
- **thread lengths**: distances between fork points and stopping points
- **coverage**: speculative useful work / total useful work
- **relative speedup**: always fail runtime / sometimes succeed runtime
Specific Evaluation Metrics

- fork heuristics: thread lengths, success rates, score distributions
- return value prediction: accuracy, speed, memory consumption
- dependence buffering: overhead, mis-speculations, population
- nested speculation: nesting height, nesting depth, time-series
- memory management: overhead, comparisons with general mallocs
- language support: termination reasons, failure reasons
Milestone Schedule

Primary goals:
- 11-2009 (PLDI): return value prediction
- 10-2009 (ESOP): nested method level speculation
- 03-2010 (OOPSLA): JIT compiler speculation
- 03-2010 (PACT): fork heuristics
- 03-2010 (OOPSLA): whole system analysis
- 10-2010 (McGill): hard deadline for thesis submission

Secondary goals:
- 01-2010 (ISMM): data structure memory management
- 10-2010 (CC): static return value prediction analyses
- 11-2010 (PLDI): interpreter and JIT performance
- 02-2011 (VEE): library design
- anytime (TOPLAS): Java design