Abstract Analysis of Method-Level Speculation

Clark Verbrugge
McGill University

Allan Kielstra
IBM Toronto Lab

Christopher J.F. Pickett
McGill University
Contents

- Essential background
- Modeling MLS
  - In-order, out-of-order, nested
  - Signaling
- Abstraction
- Experiments
- Conclusion and future work
• Method-Level Speculation
• Method-Level Speculation
Background

• Issues
  - Safety: validate speculative thread
  - Overhead
    • Forking
    • Joining & validating
    • Speculative isolation
  - Parallel work
    • Length of method, continuation
    • Misspeculation
    • Fork points
Background

- Existing systems
  - Focus on data dependencies
  - Careful heuristics
  - Context-specific
  - Varying performance...

- Why?
  - Feedback; resource-limited.
  - Speculative “style” vs code
Modeling MLS

- MLS Constraint Graph

\[
A() \{
  \text{work1}
B()
  \text{work2}
}\]
\[
B() \{
  \text{work3}
C()
  \text{work4}
}\]
\[
C() \{
  \text{work5}
}\]
Modeling MLS

- MLS Constraint Graph

\[
\begin{align*}
A() \{ & \text{work1} \\
\& B() \{ & \text{work2} \\
\} & \text{work3} \\
\& C() \{ & \text{work4} \\
\} & \text{work5} \\
\} & \text{work5} \\
\} & \text{work5} \\
\}
\end{align*}
\]

Execution: 
A → w1 → B → w3 → C → w5 → w4 → w2 → 0
Modeling MLS

• MLS Constraint Graph

\[
A() \{
\quad \text{work1}
B()
\quad \text{work2}
\}
\]

\[
B() \{
\quad \text{work3}
C()
\quad \text{work4}
\}
\]

\[
C() \{
\quad \text{work5}
\}
\]

Execution:  \[A \rightarrow w1 \rightarrow B \rightarrow w3 \rightarrow C \rightarrow w5 \rightarrow w4 \rightarrow w2 \rightarrow 0\]

Compilation edges
Modeling MLS

- All possible MLS executions

A → w1 → B → w3 → C → w5 → w4 → w2

Compiler-Driven Performance, November 10, 2011
Modeling MLS

- All possible MLS executions

A → w1 → B → w3 → C → w5 → w4 → w2 → 0

A → w1 → B → w3 → C → w5 → w4 → w2 → 0
Modeling MLS

- All possible MLS executions

A → w1 → B → w3 → C → w5 → w4 → w2
A → w1 → B → w3 → C → w5 → w4
A → w1 → B → w3 → C → w5

Compiler-Driven Performance, November 10, 2011
• All possible MLS executions

A → w1 → B → w3 → C → w5 → w4 → w2

A → w1 → B → w3 → C → w5 → w4 → w2

A → w1 → B → w3 → C → w5 → w4 → w2 → 0

A → w1 → B → w3 → C → w5 → w4 → w2 → 0
Modeling MLS

- Speculation Styles
  - Usually more than 1 speculative thread
- Out-of-order
  - Create multiple spec children from a thread
- In-order
  - Spec children can create spec children
- Nested
  - Both
Modeling MLS

- **Signaling Disciplines**
  - Support thread reuse

- **Forward-signaling**
  - Parent signals child to stop
  - Improves parallelism, mostly for out-of-order

- **Backward-signaling**
  - Child signals parent
  - Improves parallelism, mostly for in-order
    - But must retain child states
Modeling MLS

- Assume $T = SABC$
  - $S$ is the sequential preamble
  - $A$ method body
  - $B$ continuation (pre-join)
  - $C$ continuation (post-join)

- Full formula:
  \[
  MLS(T=SABC) = S \ ; \ MLS(A) | MLS(B) + MLS(C)
  \]
Abstraction

\[ T = t_1, t_2, \ldots, t_n \]

\[ MLS(T, d, time) = \]
\[ \text{for all } S = \text{preamble}(T, d) \text{ s.t. } time(S) < time \]
\[ \text{let } (t_{|S|+1}, t_b) \text{ be a continuation edge} \]
\[ T_A = t_{|S|+1}, \ldots, t_{b-1} \]
\[ \text{for all } d_1, d_2 = d-1, 0 \text{ // out-of-order} \]
\[ 0, d-1 \text{ // in-order} \]
\[ \text{split}(d-1) \text{ // nested} \]
\[ \text{for all } A = MLS(T_A, d_1, time(time(S)-F)) \]
\[ T_B = t_b, \ldots, t_n \]
\[ \text{for all } B = MLS(T_B, d_2, time(A)) \]
\[ T_C = t_{|S|+|A|+|B|+1}, \ldots, t_n \]
\[ \text{time}(S;A|B) = \text{time}(S) + F + \max(\text{time}(A), \text{time}(B)) + J \]
\[ \text{for all } C = MLS(T_C, d, time(time(S;A|B))) \]
\[ \text{time}(T) = \text{time}(S;A|B) + \text{time}(C) \]
\[ \text{return } S ; A | B + C \]
Abstraction

- Exhaustive analysis
  - Model in-order, out-of-order, nested
- Show maximum parallel potential
  - Interaction of spec design and code
  - Assume no misspeculation
    - Adds overhead, reduces available threads
Experiments

- Basic coding idioms
  - Iteration
    - for(...) { work(); } (10 iters)
  - Head-recursion
    - head() { head(); work(); } (10 levels)
  - Tail-recursion
    - tail() { work(); tail(); } (10 levels)
  - Tree-add: double head-recursion
    - ta() { ta(); ta(); work; } (3 levels)
Experiments

• Abstract time units
  - Method-call: 5 units
  - Fork: 5 units
  - Join: 20 units
  - Work: 1000 units

• Maximal parallelism; no misspeculation
Experiments

- Measurements
  - Speedup
    - In-order, out-of-order, nested (forward-signaling)
    - Max, average, “greedy” fork heuristic
  - Weight sensitivity
    - Scale fork/join overhead 0...10000 units
    - (not shown)
  - Code structure
    - Simple code changes
Experiments: Speedup

Speedup of Iter

- Best In-order
- In-order Greedy
- Avg In-order
- Best Out-of-order
- Out-of-order Greedy
- Avg Out-of-order
- Best Mixed
- Best Mixed Greedy
- Avg Mixed

Speedup Ratio

Number of Speculative Threads

1 2 3 4 5 6 7 8 9

Compiler-Driven Performance, November 10, 2011
Experiments: Speedup

Speedup of head

Speedup Ratio

Number of Speculative Threads

Best In-order
In-order Greedy
Avg In-order
Best Out-of-order
Out-of-order Greedy
Avg Out-of-order
Best Mixed
Best Mixed Greedy
Avg Mixed
Experiments: Speedup

![Speedup Graph]

- Best In-order
- In-order Greedy
- Avg In-order
- Best Out-of-order
- Out-of-order Greedy
- Avg Out-of-order
- Best Mixed
- Best Mixed Greedy
- Avg Mixed

Number of Speculative Threads

Speedup Ratio

Compiler-Driven Performance, November 10, 2011
Experiments: Structure

- Greedy forking
- Prefix:
  - `prefix() { work; }; benchmark();`
- Wrap:
  - `wrap { benchmark(); work; }`
- Suffix:
  - `benchmark(); suffix() { work; }`
Experiments: Structure

Greedy behaviour of iter compared with prefix, wrap, and suffix versions

Speedup Ratio

Number of Speculative Threads

Compiler-Driven Performance, November 10, 2011
Greedy behaviour of head compared with prefix, wrap, and suffix versions

Number of Speculative Threads

Speedup Ratio

- In-order
- In-order Prefix
- In-order Wrap
- In-order Suffix
- Out-of-order
- Out-of-order Prefix
- Out-of-order Wrap
- Out-of-order Suffix
- Mixed
- Mixed Prefix
- Mixed Wrap
- Mixed Suffix
Conclusions

- Improve understanding of TLS
  - Interaction of speculation-style and code
  - Feedback properties

- Abstraction
  - Exhaustive analysis
  - Greedy behaviour

- Step to further abstraction

Compiler-Driven Performance, November 10, 2011
Future Work

- Examine other factors
  - Misspeculation due to data-dependencies
  - Non-spec instructions
  - Backward-signaling; mixed signaling
  - Different fork heuristics
- Real program workloads
- Basis for new fork heuristics
Questions?
Experiments: Structure

Greedy behaviour of tail compared with prefix, wrap, and suffix versions

Speedup Ratio

Number of Speculative Threads

Compiler-Driven Performance, November 10, 2011
Experiments: Structure

Greedy behaviour of treedd compared with prefix, wrap, and suffix versions

Speedup Ratio vs Number of Speculative Threads