Speculation with Little Wasting

---Saving Cost in Software Speculation through Transparent Learning

Yunlian Jiang, Feng Mao, and Xipeng Shen

Computer Science Department
The College of William and Mary, Williamsburg, VA
High-Level Parallelism

- Parallel systems are becoming ubiquitous
- High-level parallelism exists in many programs
  - E.g. utilities, interpreters, scientific computations
- Difficult to parallelize

**Complexity in the code**
- Bit-level operations,
- Unrestricted pointers,
- Exception handling,
- Custom memory management,
- Third-party libraries

**Uncertain parallelism**

Example*:

```java
while ( s=nextSentence() )
{  parse(s);
     if ( isCommand(s) )
         updateParsingEnv(s);
}
```

* Simplified Parser in SPEC2k by Sleator & Temperly
Speculation-Based Parallelization

- Examples: TLS, BOP, ...

```
... work(x);
... work(y);
... work(z);
...
```

```
while (...) {
  ...
}
```

```
work(x)  work(y)  work(z)
```

```
iteration i  iteration i+1  iteration i+2
```
Penalty from Failures

- Waste of computing resources
  - Lower power efficiency
    - Portable devices
    - Green computing
  - Lower system throughput
    - Multi-programming environments
- Reduce running speed
  - Rollback overhead
  - Waiting time in critical path
Previous Solution

• Region selection
  • Strategy
    • Carefully selecting the regions to speculate through offline profiling
      • [Vijaykumar+:Micro’98, Du+:PLDI’04, Johnson+:PPoPP’07, etc.]

• Limitation
  • One decision for all inputs and all instances

Example:
```
while ( s=nextSentence() )
{  parse(s);
    if ( isCommand(s) )
        updateParsingEnv(s);
}
```
Goal of This Work

- Adaptive speculation
  - Runtime prediction of speculation profitability
  - Cross-input adaptation
  - Cross-instance adaptation
Outline

- **Scalable Loop-Level Adaptive Speculation**
- **Cross-Run Function-Level Adaptive Speculation**
- **Evaluation**
- **Conclusions**
Loop-Level Parallelism

- Properties
  - Typically many instances in one run
  - Statistical stability may exist across instances
- Strategy for adaptation
  - Cross-instance prediction
Consideration 1: Responsiveness & Robustness

- Responsive to changes across instances
- Robust to local temporary noises

last-value-based  single-value-based
Consideration II: Scalability

- Speculation depth
- The # of speculation processes
  ( [Jiang+:ICPP’08] allows level-1 adaptive speculation.)
Consideration III: The Unobserved

- Unobserved: the profitability of using a larger speculation depth.
Together

- Both responsiveness and robustness.
  - A tradeoff in the middle.
- Scalability.
  - Allowing adjustment of $d$ inside a range.
- Handling the unobserved.
  - Active exploration.

last-value-based                     single-value-based
High-Level View of Our Solution

initialize for deepest speculation

run to BeginPPR(i)

qualified for deeper speculation?  
   Y  
   increase spec depth
   
N  
   disqualified for this level speculation?  
      Y  
      decrease spec depth
      
N  
   execute with speculation

update speculation gain
Detailed Algorithm

- Responsive & robust
- Multilevel of speculation
- Active exploration without burning
Outline

• Scalable Loop-Level Adaptive Speculation
• Cross-Run Function-Level Adaptive Speculation
• Evaluation
• Conclusions
Function-Level Parallelism

- Properties
  - Typically few instances in one run
  - Cross-instance prediction is hard
- Strategy for adaptation
  - Cross-run prediction

```c
... work(x); ...
... work(y);
... work(z);
...
```
Cross-Run Learning

• Goal
  • Profitability = f (input)

run 1 → <I₁, P₁>
run 2 → <I₂, P₂>
... → <Iₙ, Pₙ>
learning → P = f (I)
Challenges

• Input complexity
• Learning algorithms and overhead control
• Prediction errors
Input Characterization

To extract important features from raw inputs

- Complex input syntax, semantics, attributes
  - e.g., a graph or a tree or a signal
- Interplay among input components
  - overshadow, equivalence, default values, etc.
Input Characterization

[CGO’09]

- eXtensible Input Characterization Language

The diagram illustrates the process of arbitary input being translated into a feature vector using the XICL language. The XICL spec is created by programmers.
Learning

- Classification Trees built transparently

X3 does not appear in the tree
Advantages of CT

• Automatic feature selection
• Handles both discrete and numeric features
• Efficient to build and use
• Good interpretability
Discriminative Prediction

• Predict only when confident
• Confidence of input-behavior models
  • Decayed average of history prediction accuracy
    \[ conf = (1 - \gamma) \times conf + \gamma \times acc \]
• Calculated periodically
  • when speculation is enabled regardless of prediction results
Evaluation

• Platform
  • Intel quadcore Xeon machine (3.4GHz) with Linux 2.6.23 installed
  • GCC 4.0.1 (“-O3”)

• Software speculation system
  • Behavior-Oriented-Parallelization system
Behavior Oriented Parallelization (BOP)

• A pure-software speculation system [Ding+:PLDI’07]
  • for coarse-grained uncertain parallelism

• Components
  • Selection of possibly parallel regions (PPRs)
    • By the programmer or a profiling tool
  • Data & code transformation
    • By a compiler
  • Runtime protection of correctness & efficiency
    • By a runtime system
BOP Execution Model

A Prog. with 3 PPR instances
Features of BOP

- Two reasons for failed speculations
  - Dependence violation
  - The parallel run is slower than the sequential counterpart

- Penalty for failed speculations
  - Waste of computing resource
    - major
  - Inferior running speed
    - minor due to understudy
Evaluation Metric

• Cost Efficiency Ratio = Speedup/ cost ratio
  • Speedup = $T_p/T_s$
  • Cost Ratio = sum(processors running times)/$T_s$
• The higher the better.
Synthetic Traces

• Include various profit patterns
• Enable flexible test & comprehensive coverage

<table>
<thead>
<tr>
<th>Trace</th>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trace 1</td>
<td>0 0 1 0 1 0 0 0 1 1 0 1 ...</td>
<td>5% 0: combinations of trace 1 and trace 2 with each phase as long as 10^2, 10^3, 10^4, 10^5.</td>
</tr>
<tr>
<td>trace 2</td>
<td>0 0 1 1 1 1 0 1 0 1 1 1 ...</td>
<td>5% 1: profitable</td>
</tr>
<tr>
<td>traces 3, 4, 5, 6</td>
<td>combinations of trace 1 and trace 2 with each phase as long as 10^2, 10^3, 10^4, 10^5.</td>
<td></td>
</tr>
</tbody>
</table>

\[
\gamma = 0.4, \ G \ TH = 0.25, \ \beta = 0.0037.
\]
Cost Eff. Ratio: ABOP/BOP

- max(d) = 1
- max(d) = 3
Cost Eff. Ratio & Speedup over BOP

cost eff. ratio

speedup
Cross-Run Results (SortRed)

```
sortedA = sort(A);
if (s)
    reduce(sortedA);
else
    reduce(A);
```
Conclusions

• Adaptive speculation may improve cost efficiency significantly
  • Cross-run learning for function-level parallelism
  • Cross-instance prediction for loop-level parallelism
Thanks!