Software Behavior Oriented Parallelization

Xipeng Shen
The College of William and Mary

Joint work with Chen Ding, Kirk Kelsey, Chris Tice, Ruke Huang, and Chengliang Zhang
at University of Rochester and Microsoft
High-level Parallelism

- Parallel computing is 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 mem. management,
- third-party libraries

**Uncertain parallelism**

Example*:

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

* Simplified Parser in SPEC2k by Sleator & Temperly
Behavior Oriented Parallelization (BOP)

• Goal
  • To improve executions with coarse-grained uncertain parallelism, while guaranteeing correctness and basic efficiency.

• 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
Possibly Parallel Regions (PPRs)

while (1) {
    get_work( );
    ...
    BeginPPR(1);
    step1();
    step2();
    EndPPR(1);
    ...
}

... BeginPPR(1);
    work(x);
    EndPPR(1);
 ...
BeginPPR(2);
    work(y);
    EndPPR(2);
 ...

- Region-based
- Likely parallelism
- Allows unpredictable entries or exits

At BeginPPR, a speculation process is created, which jumps to EndPPR and starts executing there.

Just hints, no harm to correctness
BOP Execution Model

A
Prog.
with
3
PPR
instances
BOP Execution Model

• Process-level parallelization
  • Strong isolation of processes

• Overhead is addressed by
  • forming a speculation pipeline with overhead overlapped
  • keeping most overhead off the critical path
  • using understudy to guarantee basic efficiency
    • A sequential-parallel race
BOP Correctness Protection

- Strategies
  - Classify data into different categories based on size, and patterns of accesses & values
  - Protect data in different granularity & strategies
  - Employ virtual memory paging support
- Correctness proved as an extension to data dependence theorems [similar proof as Allen & Kennedy, 2001]
### BOP Correctness Protection

<table>
<thead>
<tr>
<th>Properties to check</th>
<th>Shared</th>
<th>Checked</th>
<th>(Likely) private</th>
</tr>
</thead>
<tbody>
<tr>
<td>! ( lead-write &amp; spec-read )</td>
<td>lead: same value at BeginPPR &amp; EndPPR</td>
<td>spec: write before read</td>
<td></td>
</tr>
<tr>
<td>Protection granularity</td>
<td>page/element</td>
<td>element</td>
<td>element</td>
</tr>
</tbody>
</table>

**Overhead proportional to data size rather than accesses**
# Threads vs. Processes

- **Threads**: more efficient, fine-grained parallelism
- **Processes**: stronger isolation, coarse-grained parallelism

<table>
<thead>
<tr>
<th>Feature</th>
<th>Strong isolation</th>
<th>Weak isolation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy rollback</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Synchronization-free</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Relaxed dependence</td>
<td>yes (removes anti- &amp; output-, allows value-based check)</td>
<td>no</td>
</tr>
<tr>
<td>Independent of hardware memory consistency</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Opportunistic parallelism</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>
Evaluation

• Machine
  • Dell PowerEdge 6850 with 4 dual-core Intel 3.4GHz Xeon 7140M processors

• Compiler
  • GCC 4.0.1 with “-O3”

• Instrumentor
  • Modified GCC 4.0.1.
Gzip compressing an 84MB file

<table>
<thead>
<tr>
<th>version</th>
<th>sequential</th>
<th>speculation depth</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>times (sec)</td>
<td>8.46, 8.56,</td>
<td>7.29, 7.71</td>
</tr>
<tr>
<td></td>
<td>8.50, 8.51</td>
<td>7.32, 7.47</td>
</tr>
<tr>
<td></td>
<td>8.53, 8.48</td>
<td>5.70, 7.02</td>
</tr>
<tr>
<td>avg time</td>
<td>8.51</td>
<td>7.09</td>
</tr>
<tr>
<td>avg speedup</td>
<td>1.00</td>
<td>1.20</td>
</tr>
</tbody>
</table>
Speedup

- seq.
- spec-1
- spec-3
- spec-7

<table>
<thead>
<tr>
<th></th>
<th>seq.</th>
<th>spec-1</th>
<th>spec-3</th>
<th>spec-7</th>
</tr>
</thead>
<tbody>
<tr>
<td>gzip</td>
<td>1.0</td>
<td>1.5</td>
<td>2.0</td>
<td>3.5</td>
</tr>
<tr>
<td>parser</td>
<td>1.0</td>
<td>1.5</td>
<td>2.0</td>
<td>3.0</td>
</tr>
<tr>
<td>xlisp</td>
<td>1.0</td>
<td>1.5</td>
<td>2.0</td>
<td>3.0</td>
</tr>
</tbody>
</table>
Intel MKL (Solving 8 Linear Systems)

Instructions per sec.

Equations per system

- seq.
- 4 threads
- bop-3
- 8 threads
- bop-7
Conclusions

• BOP exploits coarse-grained (input-dependent) parallelism
  • **Strong isolation** on process level with cost amortized by
    • granularity, pipelining & efficient checking
  • **Basic efficiency** (even in the worst case)
    • Overhead proportional to data size NOT to accesses
    • Understudy & minimal critical path
• **Programmability**
  • Incremental parallelization based on partial information
  • No parallel programming or debugging
  • Only hints needed, no harm to correctness
  • PPRs allow error handling, GC, hidden dependences, and
    abnormal exits or entries
Thanks!

More details in “Software Behavior Oriented Parallelization”, PLDI’2007