Accelerating Multi-core Processor Design Space Evaluation Using Automatic Multi-threaded Workload Synthesis

Clay Hughes & Tao Li
Department of Electrical and Computer Engineering
University of Florida

(IISWC 2008)

Presentation By
Clay Hughes

IDEAL Research
(Intelligent Design of Efficient Architectures Laboratory)
Electrical and Computer Engineering
University of Florida
Motivation

• Uni-core systems are becoming increasingly rare

• Multi-threaded programming is being widely adopted

• Threaded code can take full advantage of multiple processing elements

• Multiple threads running on multiple processing elements is the prevalent execution paradigm for the next generation of computer systems
CMP Design Space Exploration

- Relies on simulation based performance models

- Inherently more expensive than uni-core processor
  - Shared/private caches, coherency protocols, interconnections, heterogeneity of cores
  - Extra simulator slow down

- Optimizes performance of multi-core oriented applications
Existing Methods

• Machine Learning and Sampling
  – SimPoint [Sherwood et al. 2002]
  – SMARTS [Wunderlich et al. 2003]

• FPGA Acceleration
  – RAMP [Wawrzynek et al. 2007]

• Statistical Simulation
  – HLS [Oskin et al. 2000]
  – [Nussbaum & Smith 2002]
  – [Eeckhout et al. 2003]
Existing Methods - 2

- Synthetic Benchmarks [Bell et al. 2005]
- Capture workload characteristics at low-level
- Build representative instruction trace
- Memory and branch models
- Compilable (portability!)
Contributions

- Synchronized Statistical Flow Graph (SSFG)
- Thread-aware memory model
- Wavelet-based branch model
• SFG
  – Nodes represent basic blocks
  – Edges represent transition probabilities

• SSFG
  – Separate SFG for each thread
  – Nodes contain thread-management information
  – Nodes contain synchronization information
Thread-aware Memory Model

```c
size_t myUnsigned = 7;
void * myFunction (void *ptr)
{
    unsigned int i = 5;
    unsigned int u_int_1 = 600;
    unsigned int array_1[10] = {0};
    ...
    myUnsigned = myUnsigned + 1;
    myUnsigned = myUnsigned * 3;
    myUnsigned = myUnsigned + 10;
    u_int_1 = u_int_1 - 1;
    array_1[i] = u_int_1;
    ...
}
```

```c
Generate Memory Reference From Distribution

(size_t myUnsigned = 7;
void * myFunction (void *ptr)
{
    unsigned int i = 5;
    unsigned int u_int_1 = 600;
    unsigned int array_1[10] = {0};
    ...
    myUnsigned = myUnsigned + 1;
    myUnsigned = myUnsigned * 3;
    myUnsigned = my Unsigned + 10;
    u_int_1 = u_int_1 - 1;
    array_1[i] = u_int_1;
    ...
})
```

```c
("movl %1, %eax" :="a" (r_outa) :="m" (shared_memInt[12]));
```

```c
("movl %eax, %0" :="m" (memInt[53]) :="a" (r_outa));
```

```c
1)
```
Wavelet-based Branch Model

- Branch transitions stored as bit vectors
- Each bit vector as a time series (e.g., 1 stands for taken and 0 represent not-taken)
- Wavelet analysis is used to extract key patterns
  - 16 wavelet coefficients
  - \( k \)-mean algorithm classifies branching patterns into clusters based on their similarity
- Using a representative pattern for all branches within the same cluster reduces the overhead of storing each block’s branch pattern
Framework (High-Level)

Instrumentation

- Application
- Disassembler
- Routine Profiler
- Instruction Profiler

Analysis

- Pin
- CFG
- Calculate Edge Weights
- Reduced SFG
- Identify Children
- SFG
- Trace Generator
- Synthetic Source Code
- Synthetic Executable
Code Generation

1. Identify Cluster ID
2. Append Branch Calculation
   - {:subw %0, %1, %2/3
   - jnz L2_46
   - :r(temp), :m(choose)
   - :m(branch[3])

Find Opcode
subl

Determine Operand A
$0x1

Determine Operand B
%0

 Populate Variables
"=m(memInt[50])"

Emit New Instruction

Thread *R-SFG
Generate Header
Enter *R-SFG
Check Thread Control
Check Thread Synchronization
Code Generator
Decrement Node
Choose Next Node

Branch?

Mem Access?

Determine Memory Type
INT LONG LONG FLOAT DOUBLE

Determine Next Offset
Increment Base

("sub $0x1, %0":="m(memInt[50])");
Evaluation: Methods

- **Workloads**
  - Splash-2

- **Tools**
  - Pin
  - VTune

- **Platforms**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Platform A</th>
<th>Platform B</th>
<th>Platform C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor Type</td>
<td>Pentium 4 - HT</td>
<td>Dual Core Pentium D</td>
<td>Core 2 Quad</td>
</tr>
<tr>
<td>Memory</td>
<td>1024MB DDR400</td>
<td>4096MB DDR2-533MHz</td>
<td>4096MB DDR2-533MHz</td>
</tr>
<tr>
<td>Storage</td>
<td>80GB SATA HDD</td>
<td>160GB SATA HDD</td>
<td>180GB SATA HDD</td>
</tr>
<tr>
<td>Operating System</td>
<td>SuSE 10.01</td>
<td>SuSE 10.01</td>
<td>SuSE 10.2</td>
</tr>
</tbody>
</table>
Evaluation: Accuracy

\[
\text{Speedup} = \frac{\text{Execution Time (Dual)}}{\text{Execution Time (Quad)}}
\]

<table>
<thead>
<tr>
<th></th>
<th>Barnes</th>
<th>Cholesky</th>
<th>FFT</th>
<th>LU</th>
<th>Ocean-C</th>
<th>Ocean-NC</th>
<th>Water-SP</th>
<th>Radix</th>
<th>Volrend</th>
<th>(Cross Benchmark Error)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Quad /Dual</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Original</td>
<td>2.26</td>
<td>1.75</td>
<td>1.26</td>
<td>1.67</td>
<td>1.23</td>
<td>1.63</td>
<td>1.73</td>
<td>1.84</td>
<td>2.73</td>
<td></td>
</tr>
<tr>
<td>Synthetic (Error)</td>
<td>2.04</td>
<td>1.92</td>
<td>1.30</td>
<td>1.53</td>
<td>1.1</td>
<td>1.53</td>
<td>1.63</td>
<td>1.74</td>
<td>3.05</td>
<td>(-9.8%) (9.7%) (3.3%) (-8.6%) (-10.3%) (-6.1%) (-5.6%) (-5.6%) (11.7%) (7.9%)</td>
</tr>
<tr>
<td><strong>Quad /HT</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Original</td>
<td>2.87</td>
<td>1.8</td>
<td>1.96</td>
<td>3.03</td>
<td>2.8</td>
<td>3.45</td>
<td>2.93</td>
<td>2.28</td>
<td>3.92</td>
<td></td>
</tr>
<tr>
<td>Synthetic (Error)</td>
<td>2.87</td>
<td>1.98</td>
<td>2.12</td>
<td>2.64</td>
<td>2.84</td>
<td>2.95</td>
<td>2.93</td>
<td>2.41</td>
<td>4.14</td>
<td>(0%) (10%) (8.5%) (-12.9%) (13%) (-14.4%) (0%) (5.5%) (5.6%) (6.5%)</td>
</tr>
<tr>
<td><strong>Dual /HT</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Original</td>
<td>1.27</td>
<td>1.02</td>
<td>1.55</td>
<td>1.82</td>
<td>2.28</td>
<td>2.12</td>
<td>1.7</td>
<td>1.24</td>
<td>1.44</td>
<td></td>
</tr>
<tr>
<td>Synthetic (Error)</td>
<td>1.41</td>
<td>1.03</td>
<td>1.63</td>
<td>1.73</td>
<td>2.57</td>
<td>1.93</td>
<td>1.8</td>
<td>1.38</td>
<td>1.36</td>
<td>(11%) (0.3%) (5%) (-4.7%) (12.9%) (-8.8%) (5.7%) (11.8%) (-5.6%) (7.3%)</td>
</tr>
<tr>
<td>(Cross Platform Error)</td>
<td>(6.9%)</td>
<td>(6.7%)</td>
<td>(5.6%)</td>
<td>(8.7%)</td>
<td>(8.2%)</td>
<td>(9.8%)</td>
<td>(3.8%)</td>
<td>(7.6%)</td>
<td>(7.6%)</td>
<td></td>
</tr>
</tbody>
</table>
**Evaluation: Efficiency**

\[ \frac{\text{Runtime}_{\text{Original}}(s)}{\text{Runtime}_{\text{Synthetic}}(s)} \]

<table>
<thead>
<tr>
<th></th>
<th>Barnes</th>
<th>Cholesky</th>
<th>FFT</th>
<th>LU</th>
<th>Ocean-C</th>
<th>Ocean-NC</th>
<th>Water-SP</th>
<th>Radix</th>
<th>Volrend</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HT</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-Thread</td>
<td>30</td>
<td>24</td>
<td>7</td>
<td>10</td>
<td>34</td>
<td>25</td>
<td>413</td>
<td>4</td>
<td>252</td>
</tr>
<tr>
<td>4-Thread</td>
<td>290</td>
<td>145</td>
<td>15</td>
<td>9</td>
<td>21</td>
<td>15</td>
<td>335</td>
<td>12</td>
<td>357</td>
</tr>
<tr>
<td><strong>Dual</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-Thread</td>
<td>30</td>
<td>21</td>
<td>8</td>
<td>9</td>
<td>32</td>
<td>26</td>
<td>356</td>
<td>4</td>
<td>286</td>
</tr>
<tr>
<td>4-Thread</td>
<td>261</td>
<td>144</td>
<td>14</td>
<td>9</td>
<td>19</td>
<td>17</td>
<td>316</td>
<td>11</td>
<td>378</td>
</tr>
<tr>
<td><strong>Quad</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-Thread</td>
<td>33</td>
<td>19</td>
<td>8</td>
<td>9</td>
<td>31</td>
<td>23</td>
<td>340</td>
<td>4</td>
<td>317</td>
</tr>
<tr>
<td>4-Thread</td>
<td>236</td>
<td>158</td>
<td>14</td>
<td>8</td>
<td>17</td>
<td>16</td>
<td>298</td>
<td>10</td>
<td>422</td>
</tr>
</tbody>
</table>
- Maximum CPI error is 12% for 4-thread
- Maximum branch prediction error is 4%
Evaluation: Cache Performance

- Maximum cache hit error is 7%
Evaluation: Multi-Core Events

- **Locked Operations Impact**
  - Measures the penalty caused by locked operations

- **Modified Data Sharing Ratio**
  - Measures the frequency of modified data sharing due to two threads using and modifying data in one cache line

- **Data Snoop Ratio**
  - Measures how often a cache is snooped by an adjacent processing element or an external one
## Evaluation: Multi-Core Events (2)

<table>
<thead>
<tr>
<th></th>
<th>Barnes</th>
<th>Cholesky</th>
<th>FFT</th>
<th>LU</th>
<th>Ocean-C</th>
<th>Ocean-NC</th>
<th>Water-SP</th>
<th>Radix</th>
<th>Volrend</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Locked Operations Impact</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Original</td>
<td>0.17%</td>
<td>1.27%</td>
<td>0.82%</td>
<td>0.25%</td>
<td>2.22%</td>
<td>2.62%</td>
<td>0.08%</td>
<td>0.64%</td>
<td>2.3%</td>
</tr>
<tr>
<td>Synthetic Error</td>
<td>3.5%</td>
<td>17.6%</td>
<td>-3.2%</td>
<td>6.6%</td>
<td>-3.2%</td>
<td>9.2%</td>
<td>-11.4%</td>
<td>-2.7%</td>
<td>11.7%</td>
</tr>
<tr>
<td><strong>Modified Data Sharing Ratio per 1k Instructions</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Original</td>
<td>0.24</td>
<td>0.27</td>
<td>0.17</td>
<td>0.1</td>
<td>0.02</td>
<td>3.1</td>
<td>0.18</td>
<td>0.23</td>
<td>0.23</td>
</tr>
<tr>
<td>Synthetic Error</td>
<td>-3.5%</td>
<td>11.6%</td>
<td>7.7%</td>
<td>-10%</td>
<td>1%</td>
<td>-9.2%</td>
<td>4.4%</td>
<td>2.6%</td>
<td>5.6%</td>
</tr>
<tr>
<td><strong>Data Snoop Ratio per 1k Instructions</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Original</td>
<td>21</td>
<td>14</td>
<td>46</td>
<td>9</td>
<td>55</td>
<td>75</td>
<td>3</td>
<td>23</td>
<td>3</td>
</tr>
<tr>
<td>Synthetic Error</td>
<td>-7%</td>
<td>-4.8%</td>
<td>-7.7%</td>
<td>13.2%</td>
<td>3.5%</td>
<td>6.8%</td>
<td>-3.4%</td>
<td>-1.6%</td>
<td>-5.6%</td>
</tr>
</tbody>
</table>
Summary

- Multi-core design space is time consuming
  - Joint-optimal designs

- Previous methods suited mostly for single-threaded applications and single PEs

- Synchronized statistical flow graphs allow for multi-threaded synthetic code generation

- Thread-aware memory model

- Wavelet-based branch model
Conclusions

- Preserve high-level program characteristics
- Preserve micro-architecture events
- Preserve complex memory operations
- Reduce runtime by one to two orders
On-going and Future Work

• Support for additional threading/sharing protocols
  – OpenMP
  – HPC
    • UPC
    • MPI

• Support for additional ISAs
  – PowerPC
  – MIPS

• Extend to simulation environment
  – SESC
  – PTLSim

• Commercial and server workload evaluation
SSFG Example
Evaluation: Instruction Mix

• Some instructions merged (calls → branches)