Adaptive Optimization using Hardware Performance Monitors

Master Thesis by Mathias Payer

Supervising Professor: Thomas Gross
Supervising Assistant: Florian Schneider
1. Summary

- **Tasks:**
  - Interface for HPM
  - Optimization of MM using HPM information

- **Challenges:**
  - Fast sampling & processing
  - Precise sampling
  - Runtime benefit

- **Method:**
  - Used kernel perfmon2 kernel patch
  - User-space library libpebsi
  - Collector-Thread in Jikes
  - Changes in Memory-Management
1. Summary / Introduction

2. Preliminaries:
   - Profiling vs. HW Sampling
   - Pentium 4 Sampling (PEBS)
   - Perfmon2 & Jikes RVM

3. Extensions
   - libpebsi
   - Jikes RVM & Collector

4. Evaluation & Benchmarks

5. Application for HW profiling (Hot/Cold GC)
2. Profiling vs. Sampling

- Modern compilers/VMs (may) use two types of information:
  - **Profiling:**
    - Monitor & trace runtime events
    - Platform independent (written in Java)
    - Data is used by AOS (OptCompiler)
  - **HW-Sampling:**
    - Uses low-level hardware information
    - Direct HW feedback
    - Can be used for (new) optimizations
    - Relatively new field
2. Pentium 4 Profiling (PEBS)

- Pentium 4 offers many (new) Hardware Performance Monitors
- Supports *Precise* Event Based Sampling (PEBS)
- HW takes & saves sample in memory, int generated on overflow
- Programmable over special register, runs in global context
- Many events can be sampled:
  - Cache misses (L1 & L2), DTLB misses, memory accesses, arithmetic instructions, ...
2. Perfmon2

- Fast, precise sampling is needed for effective optimizations.
- Many different kernel extensions exist, most are obsolete no longer maintained or outdated.
- Perfmon2 is a low level kernel interface and a high level user library.
- Supports virtualization, access restrictions, PEBS & randomization.
• The Jikes Research Virtual Machine is a complete OO Java VM.
• Used for implementations of new ideas, GCs and optimizations.
• The Adaptive Optimization System uses profiling to decide which methods need recompilation at a higher opt. level.
• HPMs are not yet used for additional information.
• A Pebsi thread runs inside Jikes to collect and process samples.
3. libpebsi

- libpebsi directly accesses the PMU (read/write to PMC & PMD)
- Offers a simple interface to PEBS (event, interval, buffer)
- Bindings for C, C++ and JNI available
- Written as redistributable library, independent from Jikes
  - Language independent
3. PEBS Control-Flow

1. Jikes loads & inits libpebsi
2. libpebsi inits perfmon2
3. perfmon2 inits buffer & hw
3. PEBS Data-Flow

1. The CPU copies autonomously

1. Jikes polls libpebsi which polls perfmon2
2. Samples are copied from kernel space into libpebsi
3. libpebsi copies the samples into Jikes
3. Jikes Collector Thread

- Polls libpebsi for new samples
  - EFLAGS, EIP, EAX, EBX, ECX, EDX, ESI, EDI, EBP & ESP
- Maps the IP to the corresponding compiled method
- Analyzes the bytecode instruction & gathers information
  - Saves additional statistics if selected
  - Analyzes field references
  - Analyzes method references
4. Benchmarks

• Typical benchmarks with high memory and gc activity are used:
  • spec MTRT    Concurrent raytracer with two threads
  • spec JACK     Java parser generator & lexical analyzer
  • DaCapo ANT   Parser and lexer for grammar files
  • DaCapo FOP   XML to PDF transformation using XSL-FO
  • DC HSQLDB    JDBC in memory DB with transactions
  • DC JYTHON    Python interpreter in Java
  • DaCapo PS    PostScript interpreter
  • pseudo JBB   spec pseudo JBB transactional DB
4. Overhead Benchmarks

- Overhead (l2 cache miss) & # processed Samples per Second

<table>
<thead>
<tr>
<th>Interval</th>
<th>pseudo spec JBB</th>
<th>Benchmark</th>
<th>Overhead</th>
<th>Samples / Sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>5000</td>
<td>2.85%</td>
<td>spec MTRT</td>
<td>0.55%</td>
<td>103.06</td>
</tr>
<tr>
<td>10000</td>
<td>1.59%</td>
<td>spec JACK</td>
<td>0.39%</td>
<td>82.05</td>
</tr>
<tr>
<td>15000</td>
<td>1.40%</td>
<td>DaCapo ANTLR</td>
<td>-0.22%</td>
<td>132.96</td>
</tr>
<tr>
<td>25000</td>
<td>0.54%</td>
<td>DaCapo FOP</td>
<td>2.24%</td>
<td>305.19</td>
</tr>
<tr>
<td>50000</td>
<td>0.73%</td>
<td>DC HSQLDB</td>
<td>6.89%</td>
<td>139.31</td>
</tr>
<tr>
<td>100000</td>
<td>0.62%</td>
<td>DC JYTHON</td>
<td>0.87%</td>
<td>83.21</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DaCapo PS</td>
<td>-0.05%</td>
<td>67.01</td>
</tr>
</tbody>
</table>

- In detail for pseudo spec JBB and all other benchmarks at interval 10000
4. Benchmark status

- Bytecode distribution of second level cache misses:
• The HW information is used in an extended garbage collector
• Objects are handled differently if they are frequently used
  • A special memory space is reserved only for hot objects
• Hotness depends on variable threshold
  • Adjusted during runtime
• Analyzes field references and reorders hot fields
Standard generational garbage collector:

Hot Cold garbage collector with a hot copy space:
5. Hot Scanning Algorithm

```java
for (int i = 0; i < NrReferences(type); i++)
{
    Address slot = type.getSlot(object, type.pebsiFieldOrder[i]);
    trace.traceObjectLocation(slot);
}
```

class Foo {
    Bar a,b;
    X d;
    X e;
    X f;
    Bar c;
}

<table>
<thead>
<tr>
<th>pebsiFieldOrder:</th>
<th>e</th>
<th>d</th>
<th>f</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heat:</td>
<td>150</td>
<td>60</td>
<td>60</td>
<td>20</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>
5. HC GC Benchmarks

### Runtime [s] | Total # Samples
--- | ---
spec MTRT | 15.9 | 900.5
spec JACK | 21.52 | 627.25
DaCapo FOP | 38.54 | 11941.75
DaCapo ANTLR | 12.24 | 2675.25
DaCapo HSQLDB | 79.66 | 2846.25
DaCapo JYTHON | 49.31 | 2087.75
DaCapo PS | 27.04 | 2520.5
specJBB | 200.67 | 63295
Conclusions

• Extendable interface for low overhead sampling
  • Useful for offline performance analysis
  • Suited for direct adaptive optimizations (avg. overhead: ~1%)
• Many events and rich statistic available inside Jikes
• Easy portable to other VM/HW-Interface