GENERIC OPTIMIZATION
DATA ANALYZER

P. Calafiura¹, S. Eranian², D. Levinthal², S. Kama³, R.A. Vitillo¹

CHEP 2012, New York, May 21-25

¹. Lawrence Berkeley National Laboratory
². Google
³. Southern Methodist University
WHAT IS GOODA?

- **Low overhead** open source Performance Monitoring Unit (PMU) event data analysis package
  - A CPU profiler
- Developed in collaboration between Google and LBNL
- Logically composed of four main components:
  - A kernel subsystem that provides an interface to the PMU
  - An event data collection tool
  - An analyzer creates call graphs, control flow graphs and spreadsheets for a variety of granularities (process, module, function, source etc.)
  - A web based GUI displays the data
### Generic Optimization Data Analyzer GUI

#### Cycles Samples

<table>
<thead>
<tr>
<th>process path</th>
<th>module path</th>
<th>unhalted_core_cycles</th>
<th>br_inst_ret...ear_return</th>
<th>function_sources</th>
<th>function_targets</th>
</tr>
</thead>
<tbody>
<tr>
<td>athena.py</td>
<td>libCaloEvent.so</td>
<td>196450 (100%)</td>
<td>467913</td>
<td>7483705</td>
<td>7483705</td>
</tr>
<tr>
<td></td>
<td>libCLHEP-Matrix-1.9.4.7.so</td>
<td>191816 (100%)</td>
<td>461322</td>
<td>7378950</td>
<td>7378950</td>
</tr>
<tr>
<td></td>
<td>lib2.12.so</td>
<td>14867 (100%)</td>
<td>71535</td>
<td>1382740</td>
<td>1132088</td>
</tr>
<tr>
<td></td>
<td>libCmalloc_minimal.so</td>
<td>4367 (100%)</td>
<td>40059</td>
<td>589095</td>
<td>610775</td>
</tr>
<tr>
<td></td>
<td>lib2.12.so</td>
<td>5218 (100%)</td>
<td>37767</td>
<td>625809</td>
<td>322026</td>
</tr>
<tr>
<td></td>
<td>libCmalloc_minimal.so</td>
<td>12180 (100%)</td>
<td>5767</td>
<td>408070</td>
<td>389066</td>
</tr>
</tbody>
</table>

#### Function Name Offset Length

- **ElementLink, ForwardIndex**: 0x4f110, 0x107, libCaloEvent
- **CLHEP::HepSymMatrix::num**: 0x1cf50, 0x4, libCLHEP
- **operator new(unsigned long)**: 0x134b0, 0x3da, libCmalloc
- **CLHEP::operator*(CLHEP::H...**: 0x19070, 0x23d, libCLHEP
- **operator delete(void*)**: 0x12c10, 0x2da, libCmalloc
- **std::Rb_tree_increment(s...**: 0x69c00, 0x5a, libstdc
- **CaloEnergyCluster::getCell...**: 0x59360, 0x9b, libCaloEvent
- **CaloCellContainer::findIn...**: 0x6dec0, 0x99, libCaloEvent
MOTIVATION

• What we were looking for:
  ‣ Low overhead profiling
  ‣ Call counts statistics
  ‣ Microarchitectural insights
  ‣ User friendly GUI
  ‣ Open Source
CODE OPTIMIZATION

• Code optimization is minimizing CPU cycles
  ‣ nothing else matters

• Decisions of what code to work on must be based on reasonably accurate estimates of what can be gained... in cycles!

• Cycles can be grouped into architecture independent groups
  ‣ forms an hierarchical tree
CYCLE ACCOUNTING
CYCLE ACCOUNTING

- Total
- Halted
- Stalled
- Unhalted
- Unstalled

Diagram shows the relationships between total, halted, stalled, unhalted, and unstalled cycle accounting states.
CYCLE ACCOUNTING

- Total
- Halted
- Unhalted
- Stalled
- Unstalled
- Load Latency
- Instruction Latency

Diagram showing relationships between cycle accounting metrics.
CYCLE ACCOUNTING

Total Cycles

Halted Cycles

Unhalted Cycles

Stalled Cycles

Unstalled Cycles

Port Saturation

Function Call Overhead

Instruction Serialization

Exception Handling

Load Latency

Bandwidth Saturation

Instruction Starvation

Instruction Latency

Store Resource Saturation

Multithread Collisions

Branch Misprediction
HARDWARE EVENT COLLECTION

• Modern CPU's include a Performance Monitoring Unit (PMU)

• Provides the ability to count the occurrence of micro-architectural events, e.g.:
  ‣ Executed instructions
  ‣ Cache misses

• Events expose inner workings of the processor as it executes code
  ‣ hundreds of events per architecture

  ‣ **caveat**: events do not map consistently between different architectures
HARDWARE EVENT COLLECTION

• PMU interrupt mode: profile where events occur vs assembly and source
  ▶ Initialize counters to the sampling period
  ▶ An interrupt is triggered when counter is zero
  ▶ Capture IP, PID, TID, LBR, CPU and other data on interrupt

• How do we convert event samples to cycles?
CYCLE DECOMPOSITION

- Stalled/unstalled cycles are decomposed as a sum of count(event) * cost(event)
  - the cost is the penalty paid in cycles for a specific event
- Example: Load Latency:
  - Use exclusive hit events
  - Includes load accesses to caches and memory, load DTLB costs and blocked store forwarding... **lots of events**!
  - Latency depends on specific configuration that needs to be determined with micro benchmarks
CYCLE DECOMPOSITION

• Load Latency on Westmere
  ‣ 6 * mem_load_retired:l2_hit +
  ‣ 52 * mem_load_retired:l3_unshared_hit +
  ‣ 85 * (mem_load_retired:other_core_l2_hit_hitm - mem_uncore_retired:local_hitm) +
  ‣ 95 * mem_uncore_retired:local_hitm +
  ‣ 250 * mem_uncore_retired:local_dram_and_remote_cache_hit +
  ‣ 450 * mem_uncore_retired:remote_dram +
  ‣ 250 * mem_uncore_retired:other_llc_miss +
  ‣ 7 * (dtlb_load_misses:stlb_hit + dtlb_load_misses:walk_completed) + dtlb_load_misses:walk_cycles +
  ‣ 8 * load_block_overlap_store

• Tools needs to know methodology so users don’t!
HOW GOODA WORKS

CPU
PMU

Kernel
Perf Events

Collector
PerfTool

Analyzer
GOoDA

Visualizer
GOoDA Visualizer
PERF EVENTS

- Performance monitoring interface introduced in the kernel in 2009
- Unified interface to access hardware performance counters, kernel software counters and tracepoints
- System call interface that exposes an high level abstraction known as event
- Events are manipulated via file descriptor obtained through the perf_event_open system call
- Samples are saved into a kernel buffer which is made visible to tools via the mmap system call
PERF TOOL

• User space tool which allows counting and sampling of events
• Many events can be sampled at the same time
• Used by the GOoDA collection scripts to collect samples into a data file
ANALYZER

• Reads and parses a perf data file

• Implements the cycle accounting methodology
  ‣ depends on the underlying architecture!

• Generates spreadsheets for:
  ‣ hot processes and functions
  ‣ source and assembly for the N hottest functions

• Generates SVG’s of the Call Graph and the Control Flow Graph
VISUALIZER

- HTML5, CSS3 & Javascript based GUI
- Reads, parses and displays the spreadsheets generated by the Analyzer
- Can be deployed on a webservice or on a client machine
- A modern browser is the only dependency
## In Action: Hot Processes

Processes ordered by hotnessness

<table>
<thead>
<tr>
<th>process path</th>
<th>module path</th>
<th>unhalted_core_cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>athena.py</td>
<td></td>
<td>473185 (100%) 266508</td>
</tr>
<tr>
<td>vmlinux</td>
<td></td>
<td>463031 (100%) 246143</td>
</tr>
<tr>
<td>gnome-settings-</td>
<td></td>
<td>9006 (100%) 19529 (0)</td>
</tr>
<tr>
<td>irqbalance</td>
<td></td>
<td>328 (100%) 156</td>
</tr>
<tr>
<td>khugepaged</td>
<td></td>
<td>253 (100%) 142</td>
</tr>
<tr>
<td>perf</td>
<td></td>
<td>164 (100%) 142</td>
</tr>
<tr>
<td>flush-253:0</td>
<td></td>
<td>134 (100%) 85</td>
</tr>
<tr>
<td>ksoftirqd/3</td>
<td></td>
<td>45 (100%)</td>
</tr>
</tbody>
</table>
IN ACTION: HOT MODULES

<table>
<thead>
<tr>
<th>process path</th>
<th>module path</th>
<th>unhalted_core_cycles</th>
<th>unhalted_cpu_cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>athena.py</td>
<td></td>
<td>473185 (100%)</td>
<td>266508</td>
</tr>
<tr>
<td>libCaloEvent.so</td>
<td></td>
<td>28434 (100%)</td>
<td>15320</td>
</tr>
<tr>
<td>libtcmalloc_minimal.so</td>
<td></td>
<td>28897 (100%)</td>
<td>14966</td>
</tr>
<tr>
<td>libm-2.12.so</td>
<td></td>
<td>41526 (100%)</td>
<td>22066</td>
</tr>
<tr>
<td>libBFieldStand.so</td>
<td></td>
<td>28792 (100%)</td>
<td>15122</td>
</tr>
<tr>
<td>libstdc++.so.6.0.10</td>
<td></td>
<td>22440 (100%)</td>
<td>14030</td>
</tr>
<tr>
<td>libCLHEP-Matrix-1.9.4.7.so</td>
<td></td>
<td>12644 (100%)</td>
<td>5017</td>
</tr>
<tr>
<td>ld-2.12.so</td>
<td></td>
<td>11451 (100%)</td>
<td>4478</td>
</tr>
<tr>
<td>libTrkAlgebraUtils.so</td>
<td></td>
<td>13464 (100%)</td>
<td>5456</td>
</tr>
</tbody>
</table>

Modules ordered by hotness
IN ACTION: CALLGRAPH

No instrumentation required
## IN ACTION: HOT FUNCTIONS

Dive into assembly and source code...

<table>
<thead>
<tr>
<th>function name</th>
<th>offset</th>
<th>length</th>
<th>module</th>
<th>process</th>
<th>unhalted_core_cycles</th>
<th>uop</th>
</tr>
</thead>
<tbody>
<tr>
<td>operator new(unsigned lon...)</td>
<td>0x134b0</td>
<td>0x3da</td>
<td>libtcmallocMinimal.so</td>
<td>athena.py</td>
<td>12927 (100%)</td>
<td>5442</td>
</tr>
<tr>
<td>master.0.gbmagz_</td>
<td>0xfb80</td>
<td>0x4a0b</td>
<td>libBFIELDStand.so</td>
<td>athena.py</td>
<td>13882 (100%)</td>
<td>5995</td>
</tr>
<tr>
<td>operator delete(void*)</td>
<td>0x12c10</td>
<td>0x2da</td>
<td>libtcmallocMinimal.so</td>
<td>athena.py</td>
<td>7619 (100%)</td>
<td>3741</td>
</tr>
<tr>
<td>std::_Rb_tree_increment(s...)</td>
<td>0x69c00</td>
<td>0x5a</td>
<td>libstdc++.so.6.0.10</td>
<td>athena.py</td>
<td>8633 (100%)</td>
<td>5697</td>
</tr>
<tr>
<td>get_bsfld_</td>
<td>0xed60</td>
<td>0xe16</td>
<td>libBFIELDStand.so</td>
<td>athena.py</td>
<td>11407 (100%)</td>
<td>7809</td>
</tr>
<tr>
<td>Trk::STEP Propagator::propagate(...)</td>
<td>0x2b230</td>
<td>0x18e2</td>
<td>libTrkEXSTEPPropagator.so</td>
<td>athena.py</td>
<td>6337 (100%)</td>
<td>2792</td>
</tr>
<tr>
<td>Trk::RungeKuttaPropagator::propagate</td>
<td>0x250e0</td>
<td>0x1051</td>
<td>libTrkEXRungeKuttaPropagator</td>
<td>athena.py</td>
<td>7589 (100%)</td>
<td>4478</td>
</tr>
<tr>
<td>ma27od_</td>
<td>0x22000</td>
<td>0x26ee</td>
<td>libTrkAlgebraUtils.so</td>
<td>athena.py</td>
<td>6397 (100%)</td>
<td>2083</td>
</tr>
<tr>
<td>Trk::FitMatrices::solveEq(...)</td>
<td>0x108a0</td>
<td>0x49a</td>
<td>libTrkIPatFitterUtils.so</td>
<td>athena.py</td>
<td>4935 (100%)</td>
<td>1701</td>
</tr>
<tr>
<td>deflate_slow</td>
<td>0x6850</td>
<td>0x976</td>
<td>libz.so.1.2.3</td>
<td>athena.py</td>
<td>5189 (100%)</td>
<td>2395</td>
</tr>
</tbody>
</table>
### IN ACTION: SOURCE

Pinpoint hot source lines

| line number | source | unhalted_core_cycles | uops-
|-------------|--------|----------------------|---|
| 1050        | numSf++; | 6337 (100%) | 2792 (44%)
| 1051        | } else { | 45 (100%) | 14 (31%)
| 1052        | // save the nearest distance to surface | 641 (100%) | 184 (28%)
| 1053        | m_currentDist.push_back( std::pair<int, std::pair<double, double>> >(-1) | 641 (100%) | 184 (28%)
| 1054        | } | | |
| 1055        | } | | |
| 1056        | | | |
| 1057        | if (distanceToTarget == maxPath || numSf == 0 ) { | | |
| 1058        | //std::cout << "propagateWithJacobian: initial distance estimate faile... | | |
| 1059        | if( m_currentDist.capacity() > m_maxCurrentDist ) m_currentDist.reserv... | | |
## In Action: Assembly

Pinpoint hot basic blocks
Branches can be expanded and explored
CONCLUSION

• Low overhead profiler
• Implements a novel cycle accounting methodology
• Visualization of reports require only a browser
• Open Source Tool (contributions welcome!)
RESOURCES

GOoDA
http://code.google.com/p/gooda/

GOoDA Visualizer
http://code.google.com/p/gooda-visualizer/