HPCToolkit Components for Measurement, Analysis and Presentation

John Mellor-Crummey, Mark Krentel, Laksono Adhianto
Mike Fagan, Gabriel Marin, Nathan Tallent

Department of Computer Science
Rice University

http://www.hipersoft.rice.edu/hpctoolkit
Outline

• **Brief overview of the HPCToolkit toolchain**

• Three new components
  — libmonitor
  — call stack sampling
  — hpcviewer

• New and notable

• Components
Performance Analysis Goals

• Accurate measurement of complex parallel codes
  — large, multi-lingual programs
  — fully optimized code: loop optimization, templates, inlining
  — binary-only libraries, sometimes partially stripped
  — complex execution environments
    – dynamic loading
    – SPMD parallel codes with threaded node programs
    – batch jobs

• Effective performance analysis
  — insightful analysis that pinpoints and explains problems
    – correlate measurements with code (yield actionable results)
    – intuitive enough for scientists and engineers
    – detailed enough for compiler writers

• Scalable to petascale systems
HPCToolkit Design Principles

• Binary-level measurement and analysis
  — observe fully optimized, dynamically linked executions
  — support multi-lingual codes with external binary-only libraries

• Sampling-based measurement (avoid instrumentation)
  — minimize systematic error and avoid blind spots
  — enable data collection for large-scale parallelism

• Collect and correlate multiple derived performance metrics
  — diagnosis requires more than one species of metric
  — derived metrics: “unused bandwidth” rather than “cycles”

• Associate metrics with both static and dynamic context
  — loop nests, procedures, inlined code, calling context

• Support top-down performance analysis
  — intuitive enough for scientists and engineers to use
  — detailed enough to meet the needs of compiler writers
HPCToolkit Workflow

compile & link

app. source → optimized binary

profile execution [hpcrun] → call stack profile

binary analysis [hpcstruct] → program structure

interpret profile correlate w/ source [hpcprof] → database

visualization [hpcviewer] → database
• Measure execution unobtrusively
  — launch unmodified, optimized application binaries
  — collect call path (or flat) profiles of events of interest
HPCToolkit Workflow

- Analyze binary to recover program structure
  - extract loop nesting & identify procedure inlining
  - map transformed loops and procedures to source

Visualization
[hpccviewer]

Database

Interpret profile correlate w/ source
[hpcprof]
HPCToolkit Workflow

- Combine multiple profiles
- Correlate dynamic metrics with static source structure
- Synthesize new metrics by combining metrics

Visualization

[hpctoolkit]

profile execution

[hpcrun]

optimized binary

compile & link

app. source

binary analysis

[hpcstruct]

call stack profile

program structure

database

interpret profile correlate w/ source

[hpcprof]

visualization

[hpcviewer]
• Visualization
  — support top-down analysis with interactive viewer
  — analyze results anytime, anywhere
Outline

• Brief overview of the HPCToolkit toolchain

• Three new components
  — libmonitor
  — call stack sampling
  — hpcviewer

• New and notable

• Components
libmonitor: A Profiling Substrate

A substrate for profiling statically and dynamically linked code

- Intercept and manage notable operations
  - Monitor gains control & performs bookkeeping operations
  - Invokes a client callback function
  - Calls real version

- Principal intercepts
  - Library initialization, finalization
  - Process creation, finalization
    - main, __libc_start_main, exit, _exit
    - fork and exec
  - Threads
    - Initialize thread support
    - Parent: thread pre-create, thread-post-create
    - Child: thread creation, finalization
  - Dynamic library open, close
  - Mpi initialize, finalize
libmonitor: A Profiling Substrate (part 2)

• Support functions
  — sigaction, signal
  — system
  — stack_bottom, process_bottom_frame
  — mpi_comm_size, mpi_comm_rank

• History: rewrite of Phil Mucci’s libmonitor (UTK)
  — Mucci’s libmonitor originally derived from HPCToolkit
Using libmonitor

- **Platforms**
  - GNU/Linux (dynamic, static)
  - BG/P, CNL, Catamount (static)
  - any Unix (static)

- **Repository host: SciDAC Outreach Center**
  - https://outreach.scidac.gov/projects/libmonitor
  - svn repository

- **License: BSD (3-clause)**
libmonitor: Building and Running

- **Autoconf and Automake**
- **Dynamic (libmonitor.so)**
  - LD_PRELOAD override __libc_start_main()
  - works with unmodified, optimized binary
  - limited to GNU/Linux with __libc_start_main
- **Static (libmonitor_wrap.a)**
  - ld --wrap main
  - define __wrap_main(), refer to __real_main()
  - works on most any Unix
  - requires re-linking application
Use libmonitor as glue between application and profiler

```c
void *
monitor_init_process(int argc, char **argv, void *data)
{
    initialize_profiling();
    start_profiling();
    return NULL;
}

void
monitor_fini_process(int how, void *data)
{
    stop_profiling();
    print_results();
}
```
libmonitor Technical Points

- Lazy dlsym of library functions (e.g. pthread_create)
- Installs signal handler for every signal
  — offers signal to client first, then application
- Catches all types of exit, _exit, signals, exec, pthread_exit, pthread_cancel
- Keeps list of threads for thread shoot down
  — mechanism to get into thread at exit (via signal)
- Provides a thread-local pointer for each thread
libmonitor To-do List

• Revisit some callback functions
  — more general access to library functions
  — provide real (unmonitored) versions of overrides

• Handle system, compiler quirks
  — when pthread_create() called before main().
  — libc system() calls hidden fork()
  — ia64 __libc_start_main() misbehaves

• Better MPI support
Outline

• Brief overview of the HPCToolkit toolchain
• Three new components
  — libmonitor
  — call stack sampling
  — hpcviewer
• New and notable
• Components
Measurement Challenges

Performance often depends upon context

- Layered design
  - application frameworks, math libraries, communication libraries
- Generic programming, e.g. C++ templates
  - both data structures and algorithms
- Context-sensitive optimization
  - e.g. inlining
- Goals
  - identify and quantify context-sensitive behavior
  - differentiate between types of performance problems
    - cheap procedure called many times
    - expensive procedure called few times
Call Path Profiling

- No instrumentation
  - statistical sampling of hardware performance counter overflows
  - gather calling context information using stack unwinding
  - overhead proportional to sampling frequency
    - not calling frequency

- Capture samples in full calling context
  - attribute sample to individual PC and source line
  - associate costs with full calling context
    - call sites too, not just callers
Novel Aspects of Our Approach

- Unwind fully-optimized and even stripped code
- Cope with dynamically loaded shared libraries
- Integrate static & dynamic context information in presentation
- Differentiate between frequent and long calls
A Call Path Profile

A call path sample

- return address
- return address
- return address

Calling Context Tree (CCT)
Unwinding Optimized Code

• Optimized code presents challenges for unwinding
  — optimized code often lacks frame pointers
  — no compiler information about epilogues
  — routines may have multiple epilogues, multiple frame sizes
  — code may be partially stripped: no info about function bounds

• Difficulties
  — where is the return address of the current frame?
    – a register, relative to SP, relative to BP
  — where is the FP for the caller’s frame?
    – a register, relative to SP, relative to BP

• Approach: use binary analysis to support unwinding
cursor = initialize_cursor(machine_context)

do {
    ui = lookup_unwind_interval(PC)  // splay tree
    if (ui is NULL)
        pb = lookup_procedure_bounds(PC)  // binary search table
        if (pb is NULL)
            sg = lookup_segment_bounds(PC)  // populate sg list
            if (sg is NULL) fail
            pb = compute_procedure_bounds(sg, PC)  // populate pb table
            if (pb is NULL) fail
            ui = compute_unwind_interval(pb, PC)  // populate ui splay tree
            if (ui is NULL) fail
            cursor = unwind_cursor(cursor, ui)  // move cursor to caller
} while (more_frames_left(cursor))
Unwind Cursor

- SP: stack pointer register for current frame
- BP: base pointer register for current frame
- IP: instruction pointer for the current frame
Unwind Interval Information

- [start_addr, end_addr)
- ra_loc
  - SP_RELATIVE: unwind using SP
    - no BP initialized (yet); SP mods = only add/subtract constant
  - STD_FRAME: unwind using SP or BP
    - might set up BP as base pointer, but SP manipulation is transparent
  - BP_FRAME: unwind using BP
    - e.g. if BP overwritten in the routine
- bp_loc
  - BP_UNCHANGED: BP on entry is still in BP
  - BP_SAVED: BP on entry is saved in stack
  - BP_OVERWRITTEN: BP is not useful for unwinding
- sp_ra_pos, sp_bp_pos: SP and BP offsets relative to SP
- bp_ra_pos, bp_bp_pos: SP and BP offsets relative to BP
- next, prev: pointers for doubly-linked list of intervals and splay tree edges
Unwinder Analyzer Details

Instructions tracked

- **call**: set high watermark interval for end of prologue
- **enter**: set up BP frame; adjust offsets accordingly
- **leave**: tear down BP frame
- **push**, **pop**: note SP change, check for BP save or restore
- **mov**: BP save/restore to memory; SP save/restore to/from BP
- **add**, **sub**: note if modify SP
- **conditional branch**: set high watermark interval for prologue
- **ret**: reset to canonical interval at next instruction
- **jmp**
  - set high watermark interval for end of prologue
  - reset to canonical interval at next instruction
Complications

• **Invisible `alloca`**
  — PGI compiler uses support routines that move SP as side effect
  — binary analysis may indicate STD_FRAME
  — only unwind with BP will succeed
  — approach needed:
    – backtracking to use BP instead of SP when necessary

• **Register-to-register moves of frame-relevant values**
  — `mov %rbp,%rax`
  — `mov %rax,0xb8(%rsp)`
  — must track register equivalences for frame relevant registers

• **Unconditional control transfers**
  — reset to “canonical interval” interval for following instruction
Finding Procedure Bounds

- Unwind interval analyzer requires function start and end
- Normally, obtain these from the symbol table
- If symbol table is partially stripped, need to recover them

Approach
  - seed process with dynamic symbols
  - segment boundaries (PLT, INIT, FINI, TEXT)
  - scan code segments (PLT, INIT, FINI, TEXT)
    - build candidate set
      - note every instruction that is a target of a call
      - note every instruction that follows an unconditional control transfer and pad bytes
    - build filter set
      - every instruction that is within the span of a conditional branch
  - output filtered candidate set
Dynamically Loaded Code

- Issue: new code may be loaded/unloaded at any time

- When a new module is loaded
  - indicate that a module is being loaded
  - load the module (and any of its dependents)
  - note new code segment mappings
  - build table of new procedure bounds

- When a module is unloaded
  - mark end of profiler epoch: code addresses no longer apply
  - flush stale cached information
Call Stack Unwinding Effectiveness

• Test cases using SPEC CPU 2006 benchmarks
  — combination of spec train and ref tests
  — compiled with intel 10.0.23 compiler
  — compiled with pathscale 3.1 compiler
  — compiled with PGI 7.0.3 compiler

• 11M samples, dropped 234 samples
  — we know the issues and expect to reduce this further
## Overhead on Opteron

### SPEC CPU2006 Benchmarks

Opteron 246

200 samples/sec ref runs

### Compilers

- pathscale 3.1
- intel 10.0.23

### Note

- not yet memoizing
- path prefixes

### Table

<table>
<thead>
<tr>
<th>benchmark</th>
<th>intel</th>
<th>intel-cs</th>
<th>intel-cs %qe</th>
<th>path</th>
<th>path-cs</th>
<th>path-cs %qe</th>
</tr>
</thead>
<tbody>
<tr>
<td>perlbench</td>
<td>1037.64</td>
<td>1049.35</td>
<td>1.13</td>
<td>1078.79</td>
<td>1104.68</td>
<td>2.40</td>
</tr>
<tr>
<td>bzip2</td>
<td>1397.57</td>
<td>1318.45</td>
<td>-5.66</td>
<td>1328.97</td>
<td>1336.70</td>
<td>0.58</td>
</tr>
<tr>
<td>gcc</td>
<td>1320.43</td>
<td>1209.24</td>
<td>-8.42</td>
<td>1247.53</td>
<td>1208.62</td>
<td>-3.12</td>
</tr>
<tr>
<td>mcf</td>
<td>1687.41</td>
<td>1703.71</td>
<td>0.97</td>
<td>1084.54</td>
<td>1100.16</td>
<td>1.44</td>
</tr>
<tr>
<td>gobmk</td>
<td>907.38</td>
<td>929.10</td>
<td>2.39</td>
<td>1140.31</td>
<td>1159.96</td>
<td>1.72</td>
</tr>
<tr>
<td>hammer</td>
<td>1098.40</td>
<td>1107.32</td>
<td>0.81</td>
<td>867.04</td>
<td>879.87</td>
<td>1.48</td>
</tr>
<tr>
<td>sjeng</td>
<td>1218.29</td>
<td>1228.91</td>
<td>0.87</td>
<td>1307.24</td>
<td>1314.02</td>
<td>0.52</td>
</tr>
<tr>
<td>libquantum</td>
<td>1850.02</td>
<td>1861.00</td>
<td>0.59</td>
<td>1738.72</td>
<td>1750.45</td>
<td>0.67</td>
</tr>
<tr>
<td>rnglib</td>
<td>1897.79</td>
<td>1909.53</td>
<td>0.62</td>
<td>1544.49</td>
<td>1596.84</td>
<td>3.39</td>
</tr>
<tr>
<td>omnet</td>
<td>990.72</td>
<td>1002.84</td>
<td>1.22</td>
<td>993.98</td>
<td>1003.14</td>
<td>0.92</td>
</tr>
<tr>
<td>astar</td>
<td>1069.19</td>
<td>1076.91</td>
<td>0.72</td>
<td>1026.62</td>
<td>1032.83</td>
<td>0.60</td>
</tr>
<tr>
<td>xalancbmk</td>
<td>906.09</td>
<td>1019.04</td>
<td>12.47</td>
<td>920.67</td>
<td>1063.77</td>
<td>15.54</td>
</tr>
<tr>
<td>specrand</td>
<td>0.23</td>
<td>1.79</td>
<td>665.99</td>
<td>0.22</td>
<td>1.47</td>
<td>559.62</td>
</tr>
<tr>
<td>bwaves</td>
<td>1277.41</td>
<td>1296.05</td>
<td>1.46</td>
<td>1169.10</td>
<td>1191.95</td>
<td>1.95</td>
</tr>
<tr>
<td>gamsess</td>
<td>1769.74</td>
<td>1777.28</td>
<td>0.43</td>
<td>2099.06</td>
<td>2122.12</td>
<td>1.10</td>
</tr>
<tr>
<td>milc</td>
<td>1137.34</td>
<td>1113.90</td>
<td>-2.06</td>
<td>1071.40</td>
<td>1067.53</td>
<td>-0.36</td>
</tr>
<tr>
<td>zeusmp</td>
<td>1064.95</td>
<td>1096.93</td>
<td>3.00</td>
<td>1092.61</td>
<td>1094.73</td>
<td>0.19</td>
</tr>
<tr>
<td>gromacs</td>
<td>808.66</td>
<td>814.71</td>
<td>0.75</td>
<td>824.94</td>
<td>829.14</td>
<td>0.51</td>
</tr>
<tr>
<td>cactusADM</td>
<td>1276.82</td>
<td>1320.08</td>
<td>3.39</td>
<td>1229.49</td>
<td>1245.18</td>
<td>1.28</td>
</tr>
<tr>
<td>testbed</td>
<td>1217.68</td>
<td>1236.99</td>
<td>1.59</td>
<td>1172.38</td>
<td>1194.29</td>
<td>1.87</td>
</tr>
<tr>
<td>namd</td>
<td>922.21</td>
<td>925.75</td>
<td>0.38</td>
<td>951.64</td>
<td>953.07</td>
<td>0.15</td>
</tr>
<tr>
<td>dealII</td>
<td>1057.41</td>
<td>1037.34</td>
<td>-1.90</td>
<td>854.29</td>
<td>861.18</td>
<td>0.81</td>
</tr>
<tr>
<td>sox</td>
<td>1083.53</td>
<td>1107.86</td>
<td>2.25</td>
<td>1169.66</td>
<td>1161.72</td>
<td>-0.68</td>
</tr>
<tr>
<td>povray</td>
<td>437.44</td>
<td>443.40</td>
<td>1.36</td>
<td>442.66</td>
<td>443.51</td>
<td>0.19</td>
</tr>
<tr>
<td>calcix</td>
<td>1404.73</td>
<td>1494.84</td>
<td>0.68</td>
<td>1574.34</td>
<td>1584.51</td>
<td>0.65</td>
</tr>
<tr>
<td>GemsFRTE</td>
<td>1883.90</td>
<td>1444.66</td>
<td>-23.32</td>
<td>1448.85</td>
<td>1404.32</td>
<td>-3.07</td>
</tr>
<tr>
<td>tonto</td>
<td>1031.49</td>
<td>1029.54</td>
<td>0.19</td>
<td>1005.57</td>
<td>969.97</td>
<td>-3.54</td>
</tr>
<tr>
<td>bjam</td>
<td>1526.78</td>
<td>1507.01</td>
<td>-1.30</td>
<td>1463.73</td>
<td>1372.36</td>
<td>-6.24</td>
</tr>
<tr>
<td>wcf</td>
<td>1218.33</td>
<td>1027.26</td>
<td>-15.68</td>
<td>1068.75</td>
<td>1078.20</td>
<td>-0.97</td>
</tr>
<tr>
<td>sphinx3</td>
<td>2014.36</td>
<td>2058.90</td>
<td>2.21</td>
<td>1988.31</td>
<td>1842.07</td>
<td>-7.35</td>
</tr>
<tr>
<td>specrand</td>
<td>0.23</td>
<td>1.66</td>
<td>626.90</td>
<td>0.28</td>
<td>1.47</td>
<td>423.30</td>
</tr>
</tbody>
</table>

**Total:**

|       | 36594.19 | 36151.36 | -1.21    | 34926.18 | 34969.85 | 0.13    |
Outline

• Brief overview of the HPCToolkit toolchain
• Three new components
  — libmonitor
  — call stack sampling
  — hpcviewer
• New and notable
• Components
hpcviewer

• Open Source

• Built on top of Eclipse platform
  — Independent application: Rich client platform

• Available on most platforms
  — x86 (32 and 64), PPC
  — Linux/GTK, Mac, Windows
  — Ongoing work: Itanium

• Requirements
  — Java 1.5
  — GTK for Linux
Derived Metrics

• Allow users to define new metrics
  — Use a formula to compose existing metrics
    – floating point waste: \( (2 \times \text{Cycle}) - \text{FP}_\text{Ins} \)
    – performance losses: \( \min(\$1 - \$2, 0) \)

• Ongoing work
  — predefined derived metrics
    – performance losses, bandwidth consumed, ...
  — storing derived metrics into a database
Hot Call Paths

- Account for cost of performance hot-spots
- Show the chain of responsibility for costs
- How long is the chain?
  - compare parent and child values
  - if the difference is greater than a threshold (50%)
    - continue the path through that child
Demo
Outline

• Brief overview of the HPCToolkit toolchain
• Three new components
  — libmonitor
  — call stack sampling
  — hpcviewer
• New and notable
• Components
New and Notable

• First-generation unwinder for BG/P
  — needs binary analysis to identify routine prologues for completeness

• Pinpointing bottlenecks in multithreaded code
  — insufficient parallelism
  — parallel overhead

• Detailed modeling of performance bottlenecks
  — provide insight into why performance is bad
Integrated View of Multiple Threads

Thread 0

Thread 1

Thread 2
Outline

- Brief overview of the HPCToolkit toolchain
- Three new components
  - libmonitor
  - call stack sampling
  - hpcviewer
- New and notable
- Components
Components We Use

- Symtab API - function bounds recovery
- Xed2 - first-party binary analysis of x86, function bounds recovery
- binutils - binary analysis for structure recovery
- OpenAnalysis - CFG construction, interval analysis
Components We Want

- First-party binary analysis of PowerPC instructions
- Saving performance data from large-scale runs to disk
- Storing, indexing, and accessing performance data @ 100K
- Visualization components
  - integrate into Eclipse RCP