Profiling with HPCToolkit

Mark W. Krentel
Department of Computer Science
Rice University
krentel@rice.edu

http://hpctoolkit.org
HPCToolkit Basic Features

- Run application natively and every 100-200 times per second, interrupt program, unwind back to main(), record call stack, and combine these into a calling context tree (CCT).

- Combine sampling data with a static analysis of the program structure for loops and inline functions.

- Present top-down, bottom-up and flat views of calling context tree (CCT) and time-sequence trace view. Metrics are displayed per source line in the context of their call path.

- Can sample on Wallclock (itimer), POSIX timers and Hardware Performance Counter Events (PAPI preset and native events): cycles, flops, cache misses, etc.

- Note: always include -g in compile flags (plus optimization) for attribution to source lines.
**HPCToolkit Advanced Features**

- Finely-tuned unwinder to handle multi-lingual, fully-optimized code, no frame pointers, broken return pointers, stack trolling, etc.
- Derived metrics -- compute flops per cycle, or flops per memory reads, etc. and attribute to lines in source code.
- Compute strong and weak scaling loss, for example:
  - strong: $8 \times (\text{time at } 8K \text{ cores}) - (\text{time at } 1K \text{ cores})$
  - weak: $(\text{time at } 8K \text{ cores and } 8\times \text{ size}) - (\text{time at } 1K \text{ cores})$
- Load imbalance -- display distribution and variance in metrics across processes and threads.
- Blame shifting -- when thread is idle or waiting on a lock, blame the working threads or holder of lock.
Measure and attribute costs in context
- sample timer or hardware counter overflows
- gather calling context using stack unwinding

Call Path Profiling

Call path sample
- return address
- return address
- return address
- instruction pointer

Calling context tree

Overhead proportional to sampling frequency...
...not call frequency
Where to find HPCToolkit

- Home site: user’s manual, build instructions, links to source code, download viewers.
  http://hpctoolkit.org/

- On theta, add to PATH:
  /projects/Tools/hpctoolkit/pkgs-theta/hpctoolkit/bin/

- Source code now on GitHub.
  http://github.com/hpctoolkit
  git clone https://github.com/hpctoolkit/hpctoolkit
  git clone https://github.com/hpctoolkit/hpctoolkit-externals

- Send questions to:
  hpctoolkit-forum at mailman.rice.edu
• In Makefile, add hpclink to front of final link line.
  hpclink cc file.o ...

• Run job with HPCRUN environment variables.
  export HPCRUN_EVENT_LIST="event@period,…"
  export HPCRUN_TRACE=1

• Run hpcstruct on program binary (for loops and inline).
  hpcstruct program

• Run hpcprof to produce database.
  hpcprof -S program.hpcstruct -l /path/to/source/tree/+ \hpctoolkit-measurements-directory

• View results with hpcviewer and hpctraceviewer.
Running on Theta

- Add to PATH:
  `/projects/Tools/hpctoolkit/pkgs-theta/hpctoolkit/bin/`

- On KNL, set sampling period to limit interrupts to about 100 per second. For example,
  `REALTIME@10000`
  `PAPI_TOT_CYC@14000000`

- For large node counts (more than 50-100 nodes), reduce the process count for profiling with the following (or some other fraction).
  `export HPCRUN_PROCESS_FRACTION=0.1`
Using OpenMP Tools Library

- Use hpclink from hpctoolkit-ompt. On theta, 
  /projects/Tools/hpctoolkit/pkgs-theta/hpctoolkit-ompt/bin/hpclink

- Compile with -fopenmp, but on hpclink link line, replace -fopenmp with libomp.a from LLVM runtime. On theta, 
  /projects/Tools/hpctoolkit/pkgs-theta/openmp-runtime/lib/libomp.a

- Add event OMP_IDLE (no number) plus time-based event: REALTIME or PAPI_TOT_CYC.

- Workarounds on theta to turn off thread affinity. 
  aprun —cc none …
  export KMP_AFFINITY=none
HPCToolkit Capabilities at a Glance

- **Attribute Costs to Code**
- **Pinpoint & Quantify** Scaling Bottlenecks
- **Assess Imbalance** and Variability
- **Analyze Behavior** over Time
- **Shift Blame from** Symptoms to Causes
- **Associate Costs with Data**

- HPCToolkit.org
Understanding Temporal Behavior

- Profiling compresses out the temporal dimension
  —temporal patterns, e.g. serialization, are invisible in profiles
- What can we do? Trace call path samples
  —sketch:
  - N times per second, take a call path sample of each thread
  - organize the samples for each thread along a time line
  - view how the execution evolves left to right
  - what do we view?
    assign each procedure a color; view a depth slice of an execution
OpenMP loop in hypre_BoomerAMGRelax using static scheduling has load imbalance; threads idle for a significant fraction of their time
Code-centric view: `hypre_BoomerAMGRelax`

Note: The highlighted OpenMP loop in `hypre_BoomerAMGRelax` accounts for only 4.6% of the execution time for this benchmark run. In real runs, solving using this loop are a dominant cost across all instances of this OpenMP loop in `hypre_BoomerAMGRelax`. 19.7% of time in this loop is spent idle idle w.r.t. total effort in this loop.
Serial Code in AMG2006 8 PE, 8 Threads

7 worker threads are idle in each process while its main MPI thread is working.
Pinpointing and Quantifying Scalability Bottlenecks

\[ P \times 600K - Q \times 400K = Q \times 200K \]

coefficients for analysis of strong scaling