Profiling with HPCToolkit

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

http://hpctoolkit.org
HPCToolkit Basic Features

- Run application natively (optimized) 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 (hpcstruct).
- 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 POSIX timers and Hardware Performance Counters (Perfmon or PAPI 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, 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:    $(\text{time at } 8K \text{ cores and } 8x \text{ size}) - (\text{time at } 1K \text{ cores})$

• 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.

• Inline sequences — show full inline sequence for C++ templates.
New Features

- Spack — now build hpctoolkit and prereqs with spack and install with spack modules.
- Simplified use case for hpcstruct and hpcprof.
- Kernel Blocktime — use Perf Events to count time spent blocked inside kernel, eg, I/O, barriers, locks, etc. (requires kernel perf events paranoid level 0 or 1).
  
  ```
  hpcrun -e CYCLES -e BLOCKTIME …
  ```
- GPU — full support for Nvidia and AMD, in progress for Intel.
- Support for OpenMP parallel regions — splice thread call paths onto master thread and identify work and idle (requires libomp replacement library), part of OpenMP 5.
Call Path Profiling

Measure and attribute costs in context
sample timer or hardware counter overflows
gather calling context using stack unwinding

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, available as module hpctoolkit (includes hpcviewer on theta login nodes).
  
  module load hpctoolkit/2022.05.15  (theta)
  module load hpctoolkit/2022.05.15-gpu    (theta-gpu)

  See: /soft/perftools/hpctoolkit/workshop-2022 for build/run notes, example databases, etc.

- Source code on GitHub
  
  https://github.com/hpctoolkit
  git clone https://github.com/hpctoolkit/hpctoolkit
  spack build instructions: README.Install

- Send questions to:
  
  hpctoolkit-forum at mailman.rice.edu
Quickstart for theta-gpu

• On theta-gpu,
  — module load hpctoolkit/2022.05.15-gpu

• Run application as follows. The first example is low-overhead. The ‘pc’ option displays the internals of the gpu kernels but can cause high overhead (1.5x to 4x).
  — hpcrun [-t] -e REALTIME -e gpu=nvidia app …
  — hpcrun [-t] -e REALTIME -e gpu=nvidia,pc app …

• Post-run analysis.
  — hpcstruct hpctoolkit-measurements-directory
  — hpcprof hpctoolkit-measurements-directory

• Finally, run hpcviewer and select database directory in the File menu chooser.
Using OpenMP Tools Library

- Use hpctoolkit ompt module.
  
  ```
  module load hpctoolkit/2020.04.ompt
  ```

- Compile with `-fopenmp`, but on `hpclink` link line, replace `-fopenmp` with `libomp.a` from LLVM runtime. Supports GNU, Intel and Clang. On theta, 
  
  ```
  /projects/Tools/hpctoolkit/pkgs-theta/llvm-openmp/lib/libomp.a
  ```

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

- Workarounds on theta to turn off thread affinity.
  
  ```
  aprun --cc none ...
  export KMP_AFFINITY=none
  ```
Trace of hpcstruct analyzing 8 Gig .so file.
Code inside GPU kernel in quicksilver proxy app.
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
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, solves 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