HPCToolkit Performance Tools
Performance analysis of CPU and GPU-accelerated applications

John Mellor-Crummey
Professor, Rice University
HPCToolkit Funding Acknowledgments

• Government
  — Exascale Computing Project 17-SC-20-SC
  — Lawrence Livermore National Laboratory Subcontract B645220
  — Argonne National Laboratory Subcontract 9F-60073

• Corporate
  — Advanced Micro Devices
  — Intel Corporation
  — TotalEnergies EP Research & Technology USA, LLC.
Rice University’s HPCToolkit Performance Tools

Measure and analyze performance of CPU and GPU-accelerated applications

• Easy: profile unmodified application binaries
• Fast: low-overhead measurement
• Informative: understand where an application spends its time and why
  ├── call path profiles associate metrics with application source code contexts
  │     └── optional hierarchical traces to understand execution dynamics
• Broad audience
  └── application developers
  └── framework developers
  └── runtime and tool developers
HPCToolkit’s Workflow for CPU Applications

- **Source Files** → **Optimized Binary** → **Compile & Link**
  - **hpcrun**
    - Profile execution on CPUs
  - **hpcstruct**
    - Analyze CPU program structure
  - **hpcviewer**
    - Present trace view and profile view
  - Database
  - **hpcprof/hpcprof-mpi**
    - Interpret profile
    - Correlate w/ source

- **Profile Files** → **Trace Files** → **Program Structure**
HPCToolkit’s Workflow for GPU-accelerated Applications

- Source Files
  - Compile & Link
  - Optimized Binary

- hpcrun
  - Profile execution on CPUs and GPUs
  - Profile Files
  - Trace Files
  - Program Structure

- hpcstruct
  - Analyze CPU/GPU program structure

- hpcviewer
  - Present trace view and profile view

- Database

- hpcprof/hpcprof-mpi
  - Interpret profile
  - Correlate w/ source
HPCToolkit’s Workflow for GPU-accelerated Applications

Step 1:
- Ensure that compilers record line mappings
- host compiler: -g
- nvcc: -lineinfo

- **Source Files** ↠ **Optimized Binary** [Compile & Link]
  - **hpcrun**
    - Profile execution on CPUs and GPUs
    - Profile Files
  - **hpcstruct**
    - Analyze CPU/GPU program structure
    - Trace Files
    - Program Structure
  - **hpcviewer**
    - Present trace view and profile view
  - **Database**
  - **hpcprof/hpcprof-mpi**
    - Interpret profile
    - Correlate w/ source
HPCToolkit’s Workflow for GPU-accelerated Applications

Step 2:
• **hpcrun** collects call path profiles of events of interest

![Diagram showing the workflow for GPU-accelerated applications](image-url)
Measurement of CPU and GPU-accelerated Applications

• Sampling using timers and hardware counter overflow on the CPU
• Callbacks when GPU operations are launched and (sometimes) completed
• GPU event stream for GPU operations; PC Samples (NVIDIA)
Call Stack Unwinding to Attribute Costs in Context

- Unwind when timer or hardware counter overflows
  —measurement overhead proportional to sampling frequency rather than call frequency
- Unwind to capture context for events such as GPU kernel launches

Call path sample

Calling context tree
hpcrun: Measure CPU and/or GPU activity

- GPU profiling
  \[\text{hpcrun} -e \text{gpu=xxx} \ <\text{app}> \ ... \ \text{xxx} \in \{\text{nvidia, amd, opencl, level0}\}

- GPU instrumentation (Intel GPU only)
  \[\text{hpcrun} -e \text{gpu=level0,inst=count, latency} \ <\text{app}>\]

- GPU PC sampling (NVIDIA GPU only)
  \[\text{hpcrun} -e \text{gpu=nvidia, pc} \ <\text{app}>\]

- CPU and GPU Tracing (in addition to profiling)
  \[\text{hpcrun} -e \text{CPUTIME} -e \text{gpu=xxx} -t \ <\text{app}>\]

- Use hpcrun with job launchers
  \[\text{jsrun} -n 32 -g 1 -a 1 \text{hpcrun} -e \text{gpu=xxx} \ <\text{app}>\]
  \[\text{srun} -n 1 -G 1 \text{hpcrun} -e \text{gpu=xxx} \ <\text{app}>\]
  \[\text{aprun} -n 16 -N 8 -d 8 \text{hpcrun} -e \text{gpu=xxx} \ <\text{app}>\]

Profiles: aggregated on the fly
- a calling context tree per thread
- a calling context tree per GPU stream
- instruction level measurements

CPU traces
- trace of call stack samples

GPU traces
- trace of call stacks that initiate GPU operations
HPCToolkit’s Workflow for GPU-accelerated Applications

Step 3:
• *hpcstruct* recovers program structure about lines, loops, and inlined functions
hpcstruct: Analyze CPU and GPU Binaries Using Multiple Threads

• Usage
  
hpcstruct [--gpucfg yes] <measurement-directory>

• What it does
  • Recover program structure information
    • Files, functions, inlined templates or functions, loops, source lines
  • In parallel, analyze all CPU and GPU binaries that were measured by HPCToolkit
    —default: use size(CPU set)/2 threads
    —analyze large application binaries with 16 threads
    —analyze multiple small application binaries concurrently with 2 threads each
  • Cache binary analysis results for reuse when analyzing other executions

NOTE: --gpucfg yes needed only for analysis of GPU binaries when NVIDIA PC samples were collected
Step 4:
- \texttt{hpcprof/hpcprof-mpi} combines profiles from multiple threads and correlate metrics to static & dynamic program structure
hpcprof/hpcprof-mpi: Associate Measurements with Program Structure

- Analyze data from modest executions sequentially
  ```
  hpcprof <measurement-directory>
  ```

- Analyze data from large executions in parallel
  ```
  jsrun -n 32 -a 1 hpcprof-mpi <measurement-directory>
  srun -n 32 hpcprof-mpi <measurement-directory>
  aprun -n 128 -N 8 hpcprof-mpi <measurement-directory>
  ```
HPCToolkit’s Workflow for GPU-accelerated Applications

Step 4:
- *hpcviewer* - interactively explore profile and traces for GPU-accelerated applications

```
<table>
<thead>
<tr>
<th>Step</th>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>hpcviewer</td>
<td>Present trace view and profile view</td>
</tr>
<tr>
<td></td>
<td>hpcrun</td>
<td>Profile execution on CPUs and GPUs</td>
</tr>
<tr>
<td></td>
<td>hpcstruct</td>
<td>Analyze CPU/GPU program structure</td>
</tr>
<tr>
<td></td>
<td>hpcprof/hpcprof-mpi</td>
<td>Interpret profile Correlate w/ source</td>
</tr>
</tbody>
</table>
```
Code-centric Analysis with hpcviewer

- function calls in full context
- inlined procedures
- inlined templates
- outlined OpenMP loops
- loops

### Source Pane

- Function calls in full context
- Inlined procedures
- Inlined templates
- Outlined OpenMP loops

### Metric Pane

- View control
- Metric display
- Navigation pane
Understanding Temporal Behavior

- Profiling compresses out the temporal dimension
  - Temporal patterns, e.g. serial sections and dynamic load imbalance are invisible in profiles

- What can we do? Trace call path samples
  - 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
Time-centric Analysis with hpcviewer

The color at a particular point in a timeline indicates the CPU procedure or GPU kernel active at that time at the selected call stack depth.

Call stack pane shows full calling context for the cursor.

A depth view showing the history of calling contexts for the thread with the cursor.

Minimap indicates part of execution trace shown.

A multi-level call stack based view of execution over time.
Time-centric Analysis with hpctraceviewer

Experimental version of QMCPack
• 32 ranks
• 32 threads each

Summary view summarizes activity across threads at each point in time
hpcstruct Example: Analyze 7.7GB TensorFlow library (170MB text) in 77s
Improved Tracing to Show Blocking on CPU Threads

Miniqmc: OpenMP on 32 CPU threads
Coarse- and Fine-grain Measurement on NVIDIA GPUs: LLNL’s Quicksilver

Compute Node
- 2xPower9 + 4xNVIDIA GPUs

• Optimized (-O2) compilation with nvcc
• Detailed measurement and attribution using PC sampling
• Attribute information to heterogeneous calling context

• Key Metrics
  - instructions executed
  - instruction stalls and reasons
  - GPU utilization

Analysis of PeleC using PC Sampling on an NVIDIA GPU

Cause: passed udata structure pointer to lambda capture

Improvement: pass udata components as scalars

https://github.com/AMReX-Combustion/PelePhysics/pull/192

4% speedup on PeleC PMF drm19 test case
HPCToolkit Trace of WarpX (16 ranks + 16 GPUs)

- Monitor HIP operations
  - kernel execution, memory copies, memset
- Collect profiles of HIP operations attributed to calling contexts where they are initiated
- Collect traces of HIP operations attributed to calling contexts where they are initiated
- Support measurement of multiple processes (e.g. MPI ranks) across multiple nodes

GPU imbalance: significantly less GPU work on last 6 ranks
Measure and Attribute OpenMP Offloading
HPCToolkit Status on GPUs

• NVIDIA
  • heterogeneous profiles, including GPU instruction-level execution and stalls using PC sampling
  • traces

• AMD
  • heterogeneous profiles; no GPU instruction-level measurements within kernels
  • measure OpenMP offloading using OMPT interface
  • traces

• Intel
  • heterogeneous profiles, including GPU instruction-level measurements with kernel instrumentation and heuristic latency attribution to instructions
  • traces
Coming Attraction: Improved Scalability of Post-mortem Analysis

- Exploit natural sparsity in performance data
  - Reduce storage requirements, efficiently use available I/O
- Use multithreading to process performance data
  - Reduce memory footprint and communication cost, efficiently use available compute
- Empirical results of improvements in HPCToolkit
  - Practical benefits: process data from 1000s of nodes with <10, in minutes!
Storing Mountains of Performance Data from Extreme-Scale Executions

Exploit natural sparsity to reduce storage and I/O

“1254x compression: 14TB → 11GB for PeleC (turbulent combustion) @ 2K threads + 2K GPUs”
Analyzing Mountains of Performance Data from Extreme-Scale Executions

Highly-efficient multithreaded parallelism!

Scalable parallelism: multithreading + MPI
hpcprof-mpi: Analyze Measurements of LAMMPS @ 2K threads + 2K GPUs

Analysis on 8 nodes using 504 threads!

Completes in 41s!
HPCToolkit Resources

• Documentation
  • User manual
  • Tutorial videos
    • [http://hpctoolkit.org/training.html](http://hpctoolkit.org/training.html)

• Software
  • Download hpcviewer GUI binaries for your laptop, desktop, cluster, or supercomputer
    • OS: Linux, Windows, MacOS
    • Processors: x86_64, aarch64, ppc64le
    • [http://hpctoolkit.org/download.html](http://hpctoolkit.org/download.html)
  • Install HPCToolkit on your Linux desktop, cluster, or supercomputer using Spack
    • [http://hpctoolkit.org/software-instructions.html](http://hpctoolkit.org/software-instructions.html)
HPCToolkit Hands-On Directions

Performance analysis of CPU and GPU-accelerated applications

John Mellor-Crummey
Professor, Rice University
Sample Performance Databases for You to Explore

- Where can you find the databases: theta:/grand/ATPESC2022/hpctoolkit/data
  - hpctoolkit-gamess.makefp.crusher.db
    - General Atomic and Molecular Electronic Structure System (GAMESS) is a general ab initio quantum chemistry package
    - Fortran; MPI + OpenMP offloading (Cray CCE); AMD GPUs
    - 110s; 40MB; 16 MPI ranks x (5 CPU threads + 2 GPU streams)
  - hpctoolkit-qmcpack-database-dmc-S16-cpu-n32-t32-d2-BGQ
    - An early prototype distributed-memory implementation of QMCPACK - a many-body ab initio Quantum Monte Carlo code for computing the electronic structure of atoms, molecules, 2D nanomaterials & solids
    - C++; MPI + OpenMP; Blue Gene Q
    - 155s; 3.2GB; 32 MPI ranks x 32 threads
  - hpctoolkit-PeleC-PMF-96GPU.d
    - PeleC is an adaptive-mesh compressible hydrodynamics code for reacting flows
    - C++; AMReX framework using CUDA; Power9 + NVIDIA GPUs
    - 1.4GB; 96 MPI ranks x (3 threads + 5 GPU streams)
  - hpctoolkit-PeleC3d.dpcpp.ex-skylake-gpu.d
    - C++; AMReX framework using SYCL; Intel Skylake with integrated GPU cores
    - Instruction-level measurements within GPU kernels
    - 10s; 44MB; Single process + GPU offloading
Profiling Quicksilver with HPCToolkit on Theta-gpu

- module swap cobalt/cobalt-knl cobalt/cobalt-gpu  # if cobalt/cobalt-gpu is loaded
- ssh thetagpusn1
- qsub -I -q single-gpu -t 60 -n 1 --attrs filesystems=grand -A ATPESC2022
- source /grand/ATPESC2022/hpctoolkit/scripts/setup-proxy.sh
- cd /grand/ATPESC2022/usr/${LOGNAME}
- git clone https://github.com/hpctoolkit/hpctoolkit-tutorial-examples
- cd hpctoolkit-tutorial-examples/examples/gpu/quicksilver
- source setup-env/theta-gpu.sh
- make build
- make run
- make run-pc
- exit  # your compute node
- exit  # thetagpusn1
- cd /grand/ATPESC2022/usr/${LOGNAME}
- cd hpctoolkit-tutorial-examples/examples/gpu/quicksilver
- module load hpctoolkit
- hpcviewer hpctoolkit-qs-gpu-cuda.d
- hpcviewer hpctoolkit-qs-gpu-cuda-pc.d
Profiling AMG2013 with HPCToolkit on Theta

- module swap cobalt/cobalt-gpu cobalt/cobalt-knl  # if cobalt/cobalt-gpu is loaded
- cd /grand/ATPESC2022/usr/${LOGNAME}
- git clone https://github.com/hpctoolkit/hpctoolkit-tutorial-examples
- cd hpctoolkit-tutorial-examples/examples/cpu/mpi+openmp/amg2013
- export HPCTOOLKIT_TUTORIAL_RESERVATION=<queue name>
- export HPCTOOLKIT_TUTORIAL_PROJECTID=ATPESC2022
- source setup-env/theta.sh
- make build
- make run
- # wait for $COBALT_JOBID.done to appear in your directory

Alternatives

- make analyze
- make analyze-parallel
- # wait for $COBALT_JOBID.done to appear in your directory

Alternatives

- make view
- hpcviewer hpctoolkit-amg2013.d