HPCToolkit: Performance Tools for GPU-Accelerated Computing

John Mellor-Crummey
Rice University

December 14, 2022
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 → Compile & Link → Optimized Binary → hpcrun

Profile execution on CPUs → Profile Files

Trace Files → hpcstruct

Analyze CPU program structure → Program Structure

hpcprof/hpcprof-mpi

Interpret profile Correlate w/ source → Database

hpcviewer

Present trace view and profile view
HPCToolkit’s Workflow for GPU-accelerated Applications

Source Files → Compile & Link → Optimized Binary

hpcrun
Profile execution on CPUs and GPUs

GPU Binary → hpcstruct
Analyze CPU/GPU program structure

Profile Files → hpcprof/hpcprof-mpi
Interpret profile Correlate with source

Trace Files → Program Structure

Database → hpcviewer
Present trace view and profile view
HPCToolkit’s Workflow for GPU-accelerated Applications

Step 1:
- Ensure that compilers record line mappings
- host compiler/hipcc: -g
- nvcc: -lineinfo
HPCToolkit’s Workflow for GPU-accelerated Applications

Step 2:
- *hpcrun* collects call path profiles (and optionally, traces) of events of interest.
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:
- return address
- return address
- return address
- instruction pointer

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

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

- GPU instrumentation (Intel GPU only)
  - hpcrun -e gpu=level0,inst=count, latency <app>

- GPU PC sampling (NVIDIA GPU only)
  - hpcrun -e gpu=nvidia, pc <app>

- CPU and GPU Tracing (in addition to profiling)
  - hpcrun -e CPUTIME -e gpu=xxx -t <app>

- Use hpcrun with job launchers
  - jsrun -n 32 -g 1 -a 1 hpcrun -e gpu=xxx <app>
  - srun -n 1 -G 1 hpcrun -e gpu=xxx <app>
  - aprun -n 16 -N 8 -d 8 hpcrun -e gpu=xxx <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
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
HPCToolkit’s Workflow for GPU-accelerated Applications

Step 4:

- `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 2 -a 1 -c 22 -b packed hpcprof-mpi <measurement-directory>
  srunt -N 2 -n 2 -c 126 hpcprof-mpi <measurement-directory>
HPCToolkit’s Workflow for GPU-accelerated Applications

Step 4:
- `hpcviewer` - interactively explore profile and traces for GPU-accelerated applications
Code-centric Analysis with hpcviewer

- **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 timeline
  - View how the execution evolves left to right

- **What do we view?**
  - Assign each procedure a color; view a depth slice of an execution

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

A multi-level call stack based view of execution over time.

Minimap indicates part of execution trace shown.
Case Study: GAMESS

- General Atomic and Molecular Electronic Structure System (GAMESS)
  - general *ab initio* quantum chemistry package
- Calculates the energies, structures, and properties of a wide range of chemical systems

- Experiments
  - GPU-accelerated nodes at a Perlmutter hackathon
    - Single node with 4 GPUs
    - Five nodes with 20 GPUs
  - GPU-accelerated node on Crusher
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter

GAMESS improved
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter

GAMESS improved
Time-centric Analysis: GAMESS 4 ranks, 4 GPUs on Perlmutter
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter

GAMESS improved
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter

GAMESS improved with better manual distribution of work in input
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter

GAMESS improved adding Rank 0 Thread 0 to GPU streams
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter

```fortran
C   IJ=1-INC
DO 150 I=2,NA
   IJ=IJ+INC
   IM1=I-1
   DO 140 J=1,IM1
      IJ=IJ+INC
      AIJ=A(IJ)
      IF(AIJ.EQ.ZERO) GO TO 140
      CALL DAXPY(MB, AIJ, B(I,1), NA, AB(J,1), NAB)
      CALL DAXPY(MB, AIJ, B(J,1), NA, AB(I,1), NAB)
140   CONTINUE
150   CONTINUE
RETURN
END
```
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter

GAMESS improved with PC Sampling
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter
Time-centric Analysis: GAMESS 5 nodes, 40 ranks, 20 GPUs on Perlmutter

136 !$omp atomic update
FA(IJ,IXYZ) = FA(IJ,IXYZ) + VAL4*DA(KL,IXYZ)
137 !$omp atomic update
FA(KL,IXYZ) = FA(KL,IXYZ) + VAL4*DA(IJ,IXYZ)
138 !$omp atomic update
FA(IK,IXYZ) = FA(IK,IXYZ) - VAL1*DA(JL,IXYZ)
139 !$omp atomic update
FA(JL,IXYZ) = FA(JL,IXYZ) - VAL1*DA(IK,IXYZ)
140 !$omp atomic update
FA(IL,IXYZ) = FA(IL,IXYZ) - VAL1*DA(JK,IXYZ)
141 !$omp atomic update
FA(JK,IXYZ) = FA(JK,IXYZ) - VAL1*DA(IL,IXYZ)
142 !$omp atomic update
ENDDO
Time-centric Analysis: GAMESS 16 ranks, 8 GPUs on Crusher
Time-centric Analysis: GAMESS 16 ranks, 8 GPUs on Crusher
Case Study: PeleC

- An adaptive mesh refinement solver for compressible reacting flows

- Experiment
  - PC Sampling on Summit node with NVIDIA GPU
Analysis of PeleC using PC Sampling on an NVIDIA GPU

Improvement:
pass udata components as scalars
https://github.com/AMReX-Combustion/PelePhysics/pull/192
4% speedup on PeleC PMF drm19 test case

Cause:
passed udata structure pointer to lambda capture
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
Using HPCToolkit at OLCF

- Summit (NVIDIA GPUs)
  - module use /gpfs/alpine/csc322/world-shared/modulefiles/ppc64le
  - module load hpctoolkit

- Crusher (AMD GPUs)
  - module use /gpfs/alpine/csc322/world-shared/modulefiles/x86_64
  - module load hpctoolkit

- Join our ECP Engagement Slack and ask questions
  - https://join.slack.com/t/hpctoolkit-ecp/shared_invite/zt-1lgrzjt93-aEB43o3SVgKFy1yFCjwlyA
HPCToolkit Resources

- Documentation
  - User manual
  - Tutorial videos
    - [http://hpctoolkit.org/training.html](http://hpctoolkit.org/training.html)
    - recorded demo of GPU analysis: [https://youtu.be/vixa3hGDuGg](https://youtu.be/vixa3hGDuGg)
- Cheat sheet
  - [https://gitlab.com/hpctoolkit/hpctoolkit/-/wikis/home](https://gitlab.com/hpctoolkit/hpctoolkit/-/wikis/home)
- 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)