A tool for top-down performance analysis of **GPU-accelerated** applications



Keren Zhou, Mark Krentel, and John Mellor-Crummey

Department of Computer Science, Rice University

Abstract

We extended Rice University's HPCToolkit to measure and analyze GPU-accelerated applications. Our tool has the following key innovations:

• A measurement substrate that employs a wait-free data structure to coordinate measurement and attribution between each application thread and a

Wait-free Communication

Case Studies

• We designed a data structure that supports wait-free communication of measurement and correlation records between the GPU monitor thread and each application thread.

• We have used our tool to analyze HPC applications on the Summit supercomputer whose compute nodes are equipped with IBM POWER9 processors and NVIDIA Volta GPUs.

• We identified three costly memory copies in Laghos from

- GPU monitor thread.
- An approach to reconstruct an approximate calling context tree for GPU computations from flat GPU PC samples.
- A method to derive GPU performance metrics from PC samples and attribute them at all levels in a heterogeneous calling context.

Background

- A *trace view* shows a series of events that happen over time on each process, thread, and GPU stream.
- A *profile view* collapses out the time dimension and correlates performance metrics with program contexts.
- Most GPU performance tools, including nvprof, Nsight Systems, Nsight Compute, TAU, and Allinea Map, only provide a trace view for GPU API calls and/or a flat profile view.



Figure: Interactions between the GPU monitor thread and application threads. pc0, pc1, and pc2 denote PC samples

> **GPU Calling Context Tree** Reconstruction

- We first construct a static call graph based on function symbols and call instructions.
- We transform the call graph into a calling context tree by splitting call edges and cloning called procedures.
- We use a heuristic method to apportion samples to

the bottom-up profile view.

• Applying optimizations based on calling context and performance increases Laghos' GPU code by 25%.

Scope	GXCOPY:IMPORTANCE
<pre> <</pre>	10.96 %
# 73: mfem::rmemcpy::rDtoD(void*, void const*, unsigned long, bool)	5.77 %
4 34: mfem::CudaVector::alloc(unsigned long)	5.77 %
109: mfem::CudaVector::operator=(mfem::CudaVector const&)	5.77 %
49: mfem::CudaProlongationOperator::MultTranspose(mfem::CudaVector const&, mfem::C	1.86 %
> 🕫 86: mfem::CudaRAPOperator::Mult(mfem::CudaVector const&, mfem::CudaVector&) co	1.81 %
ase 1 > @ 245: mfem::hydrodynamics::LagrangianHydroOperator::Mult(mfem::CudaVector const&	0.05 %
4 29: mfem::CudaProlongationOperator::Mult(mfem::CudaVector const&, mfem::CudaVector	1.86 %
> 🕫 84: mfem::CudaRAPOperator::Mult(mfem::CudaVector const&, mfem::CudaVector&) co	1.81 %
> 🕫 256: mfem::hydrodynamics::LagrangianHydroOperator::Mult(mfem::CudaVector const&	0.05 %
ase 2 > 🖷 130: mfem::hydrodynamics::CudaMassOperator::Mult(mfem::CudaVector const&, mfem::Cu	1.81 %
> 🕫 212: mfem::hydrodynamics::LagrangianHydroOperator::Mult(mfem::CudaVector const&, m	0.13 %
> 🕫 39: mfem::CudaCGSolver::h_Mult(mfem::CudaVector const&, mfem::CudaVector&) const	0.10 %
> 🕫 436: main	0.01 %
> 495: mfem::CudaMassIntegrator::SetOperator(mfem::CudaVector&)	0.00 %
> 🕫 146: mfem::hydrodynamics::LagrangianHydroOperator::LagrangianHydroOperator(int, mfe	0.00 %
ase 3, 425: main	0.00 %
> 🕫 61: cuVectorDot(unsigned long, double const*, double const*)	5.14 %

Figure: Laghos is a DOE mini-app that solves the time-dependent Euler equation of compressible gas dynamics.

• By combining PC sampling measurements with instruction mix analysis, our tool provides the metrics necessary to construct a Roofline model. • We optimized Nekbone to 84% of the peak performance limited by memory bandwidth.

- HPCToolkit attributes performance metrics to calling contexts that span both CPUs and GPUs.
- In the Figure below, we show HPCToolkit's heterogeneous calling context for a **fully optimized** GPU kernel that has 28 device functions.

NuclearData.cc 246// Return the total cross section for this energy grou 247 HOST DEVICE 248 double NuclearData::getReactionCrossSection(int reactIndex, unsigned int isotopeIndex, unsigned int group) qs assert(isotopeIndex < isotopes.size());</pre> assert(reactIndex < isotopes[isotopeIndex]. species[0]. reactions.size())</pre> 255 HOST_DEVICE_END Top-down view 🖾 🐟 Bottom-up view 📊 Flat view 1 🕂 🕂 🧑 😥 🕅 📰 🖓 🔚 🖓 👘 🗸

Scope		GINS:Sum (I)	GINS:STL_ANY:Sum (I)
loop at main.cc: 159		1.07e+11 100 %	9.83e+10 100 %
loop at main.cc: 159		1.07e+11 100 %	9.83e+10 100 %
loop at main.cc: 163		1.07e+11 100 %	9.83e+10 100 %
193: [I] CycleTrackingKernel(MonteCarlo*, int, ParticleVault*, ParticleVault*)		1.07e+11 100 %	9.83e+10 100 %
127:device_stub_Z19CycleTrackingKernelP10MonteCarloiP13ParticleVaultS2_(MonteCarlo*, in		1.07e+11 100 %	9.83e+10 100 %
CPU Calling Context 14: [I] cudaLaunchKernel < char>		1.07e+11 100 %	9.83e+10 100 %
GPU API Node		1.07e+11 100 %	9.83e+10 100 %
174: CycleTrackingKernel(MonteCarlo*, int, ParticleVault*, ParticleVault*)		1.07e+11 100 %	9.83e+10 100 %
132: CycleTrackingGuts(MonteCarlo*, int, ParticleVault*, ParticleVault*)		1.06e+11 100.0	9.82e+10 100.0
loop at CycleTracking.cc: 118		8.90e+10 83.5%	8.08e+10 82.2%
B⇒ 63: CollisionEvent(MonteCarlo*, MC_Particle&, unsigned int)		4.99e+10 46.9%	4.49e+10 45.7%
[1] ir	lined from QS_Vector.hh: 94	3.76e+10 35.3%	3.34e+10 34.0%
la	pop at QS_Vector.hh: 94	3.61e+10 33.9%	3.20e+10 32.5%
	[I] inlined from CollisionEvent.cc: 71	3.58e+10 33.6%	3.17e+10 32.3%
GPU Loops and Inline Functions	loop at CollisionEvent.cc: 71	3.42e+10 32.1%	3.03e+10 30.9%
	73: macroscopicCrossSection(MonteCarlo*, int, int, int, int, int)	3.11e+10 29.2%	2.78e+10 28.3%
	[I] inlined from MacroscopicCrossSection.cc: 45	2.57e+10 24.1%	2.31e+10 23.5%
	➡ 41: NuclearData::getReactionCrossSection(unsigned int, u	1.69e+10 15.9%	1.56e+10 15.9%
GPU Calling Context	[I] inlined from NuclearData.cc: 194	9.36e+09 8.8%	8.70e+09 8.9%
GPU Hotspot	NuclearData.cc: 253	9.06e+09 8.5%	8.44+09 8.6%

procedures called from multiple call sites.



Figure: A procedures's costs are computed using the number of samples at each call site

Metrics Derivation

- Some performance metrics cannot be collected in the same pass with PC samples.
- Nsight-compute runs nine passes to collect all of these metrics for a GPU kernel.
- Our tool estimates important performance metrics based on PC samples and kernel statistics, including SM



Figure: Nekbone is a lightweight subset of Nek5000 that mimics the computational characteristics of Nek5000, a high-order Navier-Stokes solver based on the spectral element method.

Next Steps

• We are continuing work on our tools with the aim of turning measurement data and metrics into high-level guidance for performance tuning.

• We have begun to apply our tool to MPI-based HPC







instruction throughput, etc.

