A Tool for Performance Analysis of GPU-Accelerated Applications

Keren Zhou
Department of Computer Science
Rice University
Houston, Texas, USA
keren.zhou@rice.edu

John Mellor-Crummey
Department of Computer Science
Rice University
Houston, Texas, USA
johnmc@rice.edu

***Abstract*—Architectures for High-Performance Computing (HPC) now commonly employ accelerators such as Graphics Processing Units (GPUs). High-level programming abstractions for accelerated computing include OpenMP as well as RAJA and Kokkos—programming abstractions based on C++ templates. Such programming models hide GPU architectural details and generate sophisticated GPU code organized as many small procedures. For example, a dot product kernel expressed using RAJA atop NVIDIA's thrust templates yields 35 procedures. Existing performance tools are ill-suited for analyzing such complex kernels because they lack a comprehensive profile view. At best, tools such as NVIDIA's nvvp provide a profile view that shows only limited CPU calling contexts and omits both calling contexts and loops in GPU code. To address this problem, we extended Rice University's HPCToolkit to build a complete profile view for GPU-accelerated applications.**

I. Approach

We extended HPCToolkit [1], [2] with a wait-free measurement subsystem to attribute costs of a GPU kernel measured using instruction samples back to the worker thread that launched the kernel. GPU instruction samples are collected using NVIDIA's CUPTI API, which spawns a background CUPTI thread at program launch. When an application thread T launches a GPU kernel, it tags the kernel with a correlation ID C and notifies the CUPTI thread that C belongs to T; when samples associated with C are collected, the CUPTI thread uses this information to attribute the samples back to thread T. The CUPTI thread and application threads coordinate to attribute performance information using wait-free queues. Using this strategy, HPCToolkit can monitor long-running applications with low memory and time overhead.

To attribute costs to GPU code, HPCToolkit recovers loops and calling contexts in GPU machine code. HPCToolkit reconstructs GPU control flow graphs by parsing branch targets and analyzes the graphs to identify loop nests. To understand calling contexts in GPU code, HPCToolkit recovers static call graphs and transforms them into calling context trees by splitting call edges and cloning called procedures.

To analyze performance data for each GPU calling context, HPCToolkit first attributes costs to inline functions, loop nests, and individual source lines in GPU procedures. Next, it apportions costs of each GPU procedure its call sites. If necessary, HPCToolkit uses the sample count of each call instruction to divide a GPU procedure's costs among its multiple call sites. To cope with recursive calls, HPCToolkit merges all GPU procedures in the same strongly connected component group into a "supernode" and apportions costs for the "supernode."

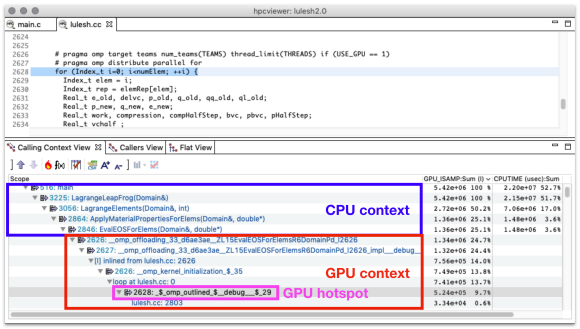

Fig. 1. HPCToolkit's hpcviewer attributing performance metrics to a GPU context that includes a call chain and a loop

II. Initial Results

We have used HPCToolkit to analyze accelerated applications written using RAJA and OpenMP on the Summit supercomputer, whose compute nodes are equipped with IBM POWER9 processors and NVIDIA Volta GPUs. Figure 1 shows HPCToolkit's detailed analysis of calling contexts and loops in GPU code, which enables precise attribution of costs for executions of GPU-accelerated applications and helps users identify hotspots and bottlenecks.

Acknowledgments

This research was partially supported by LLNL Subcontract B626605 of DOE Prime Contract DE-AC52-07NA27344 and by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration).

References

[1] L. Adhianto *et al.*, "HPCToolkit: Tools for performance analysis of optimized parallel programs," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 6, pp. 685–701, 2010.

[2] Rice University, "HPCToolkit performance tools project," http://hpctoolkit.org.