Modeling Performance of Graph Programs on GPUs in a Compiler

Sreepathi Pai and Keshav Pingali
The University of Texas at Austin

Graph processing primitives like Breadth-First-Search (BFS), Single-Source Shortest Path (SSSP) and Minimum-Spanning Tree (MST) are well-known examples of *irregular* data-parallel programs. These programs require extensive use of memory indirection and data-dependent control flow, so it is impossible to ignore the *values* of inputs (as opposed, to say, size) when building performance models for these programs. Currently, the only consensus about the behaviour of these programs, after several characterization studies, is that memory bandwidth is *not* a bottleneck since it is never fully utilized in these programs. However, no study has yielded a performance model. This lack of a sound performance model has hindered efforts to improve compilers, runtimes and computer architecture for these programs. In this work, we describe our motivation for building a sound performance model, our methodology, our results and potential avenues for future exploration enabled by our infrastructure.

Our primary motivation stems from our experiences with the Galois GPU compiler. We built this compiler to generate CUDA code for GPUs from high-level representations of graph algorithms. Using several automatic optimizations tailored to GPU graph programs, our compiler produces code that meets or exceeds the performance of 8 well-known hand-optimized third-party implementations. However, automatically deciding *which* optimizations to apply to graph programs proved exceedingly difficult. Some optimizations led to speedups on some inputs, but slowed down on other inputs. The continuing rapid evolution of GPUs also exacerbated this problem as optimizations changed in relevance with new generations of hardware.

To develop a clear understanding of overall performance, we created a model for graph programs that would generalize well to other graph algorithms consuming a variety of inputs and running on various GPUs. We developed a representation of graph programs called the Operator Machine that uses queuing network models to represent graph programs. This representation splits any graph program into multiple stages, the majority of which are shared by all graph programs. Amenable to both measurement and analysis, our characterization of the measured throughputs at each of these shared stages led to several general insights.

We found that graph programs on GPUs suffer largely from high TLB miss rates and are therefore unable to fully utilize memory bandwidth. Contradicting previous studies, we discovered that the large number of iterations in iterative graph programs operating on high-diameter graphs is not a performance concern; instead the *rate* of iterations in these programs overwhelms the CUDA runtime, leading to GPU under-utilization. The performance gains of widely-used dynamic scheduling techniques, often prescribed to load-balance inputs with widely-varying degrees (such as social networks), were found to be bottlenecked by barrier throughput. Additionally, the gains from these techniques were linked to the graph algorithm rather than the type of graph. Finally, we were able to show that for graph programs, no one set of optimizations suffices and *hybrid* implementations featuring multiple variants must be generated by the compiler.

Since the number of variants is quite large, the compiler has to reason about the performance of each variant so that it can generate the minimal number of variants necessary for a hybrid implementation as well as the code to switch between them at runtime. While code generation influenced by data characteristics and driven by performance models is not without precedent – SQL query optimizers come to mind – compilers have traditionally not incorporated complex performance models such as the one we have proposed. For now, our compiler provides the only infrastructure to systematically probe the performance of graph programs on GPUs. As programs with input-dependent behaviour become increasingly important workloads and hardware platforms become increasingly diverse, we believe that compilers augmented with these kinds of performance models will become critical tools in order to exploit the performance of these systems.