Scaling HabaneroUPC++ on Heterogeneous Supercomputers

Vivek Kumar[†], Max Grossman[†], Hongzhang Shan[‡], and Vivek Sarkar[†]

[†]Rice University, [‡]Lawrence Berkeley National Laboratory

Accelerators/co-processors have made their way into supercomputing systems. These modern heterogeneous systems feature multiple layers of memory hierarchies, and produce a high degree of thread-level parallelism. To ensure that current and future applications perform well on these systems, it is important that users be able to cleanly express the various types of parallelism found in their applications while trusting that expertly-implemented runtime libraries will schedule this parallelism in a way that efficiently utilizes the memory hierarchies and computational resources of their system.

In this poster we present our work-in-progress, a distributed, heterogeneous programming model in HabaneroUPC++, which aims to target distributed, heterogeneous systems with multi-layered memory hierarchies through a distributed data-driven programming model that integrates all memory layers together through data-flow programming.

Keywords—Habanero; UPC++; PGAS; Heterogeneous computing; CUDA;

I. Introduction

Modern supercomputers are turning from multicore CPUs to accelerators/co-processors for the core of their computational power, thanks to better energy efficiency and space management when using accelerators [1], [2]. Because of this, both novel and existing distributed parallel programming models and languages have added, or are in the process of adding accelerator support. Some of these programming models are UPC [3], X10 [4], Phalanax [5], Kokkos [6] and Raja [7]. Compiler directives based programming models, such as OpenMP 4.0 [8] and OpenACC [9] support accelerators and can be used in hybrid programming models using MPI [10], [11] and UPC++ [12]. While some of these use compilers to translate source codes, others are implemented in the form of libraries. The main advantage of a library-based approach is that it avoids the need for any custom compiler support and allows easy integration of the new features provided by mainstream compilers.

Existing programming models allow the offload of compute kernels to accelerator, either using a synchronous model (OpenMP, OpenACC, Kokkos, Raja), or an asynchronous model (X10 and Phalanax). Launching a compute kernel as an asynchronous task allows parallel execution overlap with other asynchronous tasks. Phalanax takes one step further and allows the creation of dependencies among asynchronous tasks. A task executing on the GPU can signal the launch of another asynchronous task on the host or a remote node (not allowed in X10). This is similar to task dependencies in OpenMP. Declaring dependencies explicitly in this fashion, rather than waiting and computation-communication overlap directly in the calling thread, is essential for scalability. It allows the runtime to decouple waiting for task completion from the sequential execution of the calling thread and implement waiting efficiently. However, Phalanax does not employ threads within a PGAS (Partitioned Global Address Space) rank, limiting the overlap that is achievable. It has been well studied that assigning a MPI (or PGAS) rank per processing element is not scalable on modern supercomputers, which are placing more and more number of cores in each node [13].

Modern HPC systems contain deep memory hierarchies. Data accesses latencies dramatically increase as one progress higher in the memory hierarchy. Exploiting data locality in parallel programming on these complex systems is a challenge for users. X10 allows managing locality through the use of `places` and enables co-location of asynchronous tasks and shared mutable locations. However, this model is flat in structure and cannot capture vertical locality in a memory hierarchy. Phalanx employs a memory model closely related to X10. However, Phalanax provides a hierarchical rather than flat model of `places` by adopting a model of memory spaces that extends beyond the usual global/local divisions.

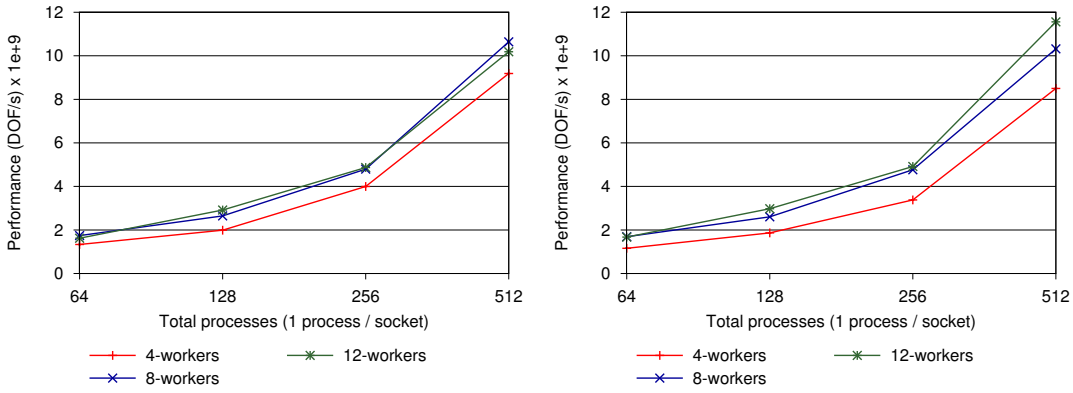
II. HabaneroUPC++

HabaneroUPC++ [14] is a compiler-free PGAS library that supports a tighter integration of intra-process and inter-process parallelism than standard hybrid programming approaches. It uses the UPC++ library for PGAS communication and function shipping, and the Habanero-C++ library to provide work-stealing and multi-threaded scheduling within a UPC++ process. HabaneroUPC++ uses C++11 lambda-based user interfaces for launching asynchronous tasks, such as: a) intra-process asynchronous tasks — **`async`**, **`asyncAwait`**, **`asyncPhased`** and **`forasync`** (currently not supported to run across accelerators); b) inter-process asynchronous tasks — **`asyncAt`** for asynchronous remote function invocation and **`asyncCopy`** for asynchronous local/remote copy; and c) a barrier style **`finish_spmd`** to join all the asynchronous tasks (both intra and inter-process).

A. Motivation to use HabaneroUPC++

We plan to use HabaneroUPC++ to implement our ideas for a highly scalable runtime for parallel programming on heterogeneous supercomputers. In this section, we demonstrate

To appear in IEEE PGAS conference as extended abstract, September 2015, Washington, D.C., USA.



(a) UPC++ using OpenMP

(b) HabaneroUPC++

Fig. 1: HPGMG performance comparison on Edison supercomputer.

the performance of our current implementation of HabaneroUPC++ on the Edison supercomputer at NERSC, using the HPGMG [15] benchmark. HPGMG intends to be a more balanced, robust, and diverse benchmark than the traditional HPL [16] benchmark. It provides a better representation of a supercomputer's real-world performance. The Edison supercomputer is composed of homogeneous processing elements, but comparable performance of HabaneroUPC++ relative to a highly optimized reference implementation will augment our choice of HabaneroUPC++ as HPC infrastructure.

The reference implementation of HPGMG uses UPC++ and OpenMP. In the HabaneroUPC++ version of HPGMG, we replace OpenMP parallel **`for`** loops with asynchronous **`forasync`** calls and often overlap computations and communications. Each node of Edison has two sockets and each socket has 12 cores. In each experiment we run one process (one PGAS rank) per socket. We vary the number of threads per process in each experiment. Figure 1 shows the results of our experiments, where we see that HabaneroUPC++ performs similarly to the reference implementation.

III. Planned Implementation

A. Launching Tasks across GPUs

As a first step, we plan to extend HabaneroUPC++ asynchronous tasks to execute on GPGPU accelerators. Programmers will define a GPGPU task using C++ `functors`. One or more dedicated accelerator management threads will acquire and launch work on the GPGPUs in the system. These will be special-purpose Habanero-C++ threads which prioritize SIMD tasks. We plan to use the `hwloc` [17] hardware locality package to associate threads with GPUs in a way that minimizes latency.

Because the address spaces of GPGPUs and their host systems are both logically and physically separate, an address translation step is necessary immediately prior to launching tasks on the GPU. Variables passed by reference must be translated into the GPGPU address space. If necessary, space is allocated and populated for these reference variables. This work will also investigate using CUDA Unified Memory to reduce programmer burden.

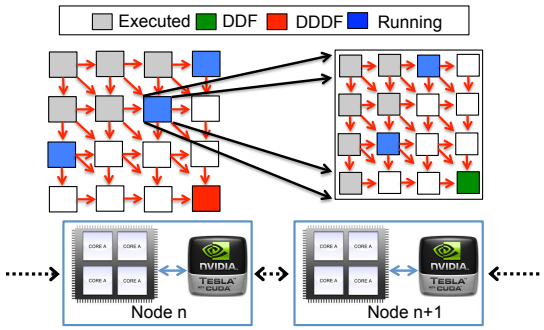


Fig. 2: Smith-Waterman dependency graph, its hierarchical tiling and execution.

On the GPGPU, dynamic task creation is possible through the instantiation of additional task instances, which are stored in a concurrent queue on the GPGPU. When the GPGPU kernel completes, these task objects are transferred back to the host where they are inserted in to the Habanero-C++ runtime for scheduling.

B. Distributed Data Driven Futures

HCMPI [13] introduced Distributed Data-Driven Futures (DDDF) object that carries a globally unique identifier to help tasks in communicating data in a global name space. This allows overlapping of computation and communication in applications. HabaneroUPC++ provides intra-process Data-Driven Future (DDF) [18] but does not allow DDDFs. We plan to extend HCMPI's DDDF support in HabaneroUPC++ using the RDMA `put` and `get` provided by UPC++. Unlike HCMPI, DDDF in HabaneroUPC++ would be able to satisfy task dependencies at all levels of memory hierarchies including accelerators. Figure 2 shows the working of DDDFs and DDFs using Smith-Waterman local sequencing alignment algorithm [13]. Each cell in the matrix represents an asynchronous task in different possible execution states.

C. Hierarchical Place Trees

Hierarchical Place Trees (HPT) [19] are supported in Habanero-C [20] and Habanero-Java [21]. HPTs model complex memory hierarchies of various computing systems and provide an abstraction powerful enough to exploit locality at each level in the memory hierarchy, without compromising performance. We plan to extend HPT support in HabaneroUPC++ to provide a simple way to control locality.

REFERENCES

[1] "A slice of green with accelerators on top," http://www.hpcwire.com/2014/07/01/slice-green-accelerators-top/, July 2014.

[2] "Top500 highlights," http://www.top500.org/lists/2014/11/highlights/, November 2014.

[3] Y. Zheng, C. Iancu, P. H. Hargrove, S.-J. Min, and K. Yelick, "Extending unified parallel C for GPU computing," in *SIAMPP*, 2010.

[4] D. Cunningham, R. Bordawekar, and V. Saraswat, "GPU programming in a high level language: Compiling X10 to CUDA," in *X10 Workshop*, 2011, p. 8.

[5] M. Garland, M. Kudlur, and Y. Zheng, "Designing a unified programming model for heterogeneous machines," in *SC*, 2012, pp. 1–11.

[6] D. S. H. Carter Edwards, Christian Trott, "Kokkos tutorial," in *Trilinos User Group Meeting*, 2013.

[7] R. Hornung and J. Keasler, "RAJA portability layer," https://e-reports-ext.llnl.gov/pdf/782261.pdf.

[8] OpenMP Architecture Review Board, "The OpenMP API specification for parallel programming," http://openmp.org/.

[9] "OpenACC directives for accelerators," http://www.openacc-standard.org/.

[10] M. Snir, S. W. Otto, D. W. Walker, J. Dongarra, and S. Huss-Lederman, *MPI: The Complete Reference*. MIT press, 1995.

[11] A. S. Bland, J. C. Wells, O. E. Messer, O. R. Hernandez, and J. H. Rogers, "Titan: Early experience with the Cray XK6 at Oak Ridge National Laboratory," in *CUG*, 2012.

[12] Y. Zheng, A. Kamil, M. Driscoll, H. Shan, and K. Yelick, "UPC++: A PGAS extension for C++," in *IPDPS*, 2014, pp. 1105–1114.

[13] S. Chatterjee, S. Tasirlar, Z. Budimlic, V. Cave, M. Chabbi, M. Grossman, V. Sarkar, and Y. Yan, "Integrating asynchronous task parallelism with MPI," in *IPDPS*, 2013, pp. 712–725.

[14] V. Kumar, Y. Zheng, V. Cavé, Z. Budimlić, and V. Sarkar, "HabaneroUPC++: A compiler-free PGAS library," in *PGAS*, 2014, p. 5.

[15] "High-performance geometric multigrid," https://hpgmg.org/.

[16] A. Petitet, "HPL-a portable implementation of the high-performance linpack benchmark for distributed-memory computers," http://www.netlib-. org/-benchmark/hpl/, 2004.

[17] The Open MPI Project, "Portable hardware locality (hwloc)," http://www.open-mpi.org/projects/hwloc/.

[18] S. Tasirlar and V. Sarkar, "Data-driven tasks and their implementation," in *ICPP*, 2011, pp. 652–661.

[19] Y. Yan, J. Zhao, Y. Guo, and V. Sarkar, "Hierarchical place trees: A portable abstraction for task parallelism and data movement," in *LCPC*, 2010, pp. 172–187.

[20] "Habanero-C Overview," https://wiki.rice.edu/confluence/display/HABANERO/Habanero-C, Rice University, 2013.

[21] V. Cavé, J. Zhao, J. Shirako, and V. Sarkar, "Habanero-Java: the new adventures of old X10," in *PPPJ*, 2011, pp. 51–61.