Scaling HabaneroUPC++ on Heterogeneous Supercomputers

Vivek Kumar[1], Max Grossman[1], Hongzhang Shan[2], and Vivek Sarkar[1]
[1] Rice University, [2] Lawrence Berkeley National Laboratory

1 Introduction

Modern supercomputers are turning from multicore CPUs to accelerators/co-processors for the core of their computational power, thanks to better performance, energy efficiency, and space management when using accelerators.

Existing programming models for these architectures offload computational kernels to accelerators, either synchronously (e.g., OpenMP [1] and OpenACC [2]), or asynchronously (e.g., Phalanx [3] and X10 [4]).

Scaling on future exascale distributed, heterogeneous supercomputers requires an efficient integration of communication, memory management, and work scheduling at all layers (inter-accelerator, host-accelerator, inter-node).

2 Motivation

Distributed data-driven programming enables computation-communication overlap for improved scalability:

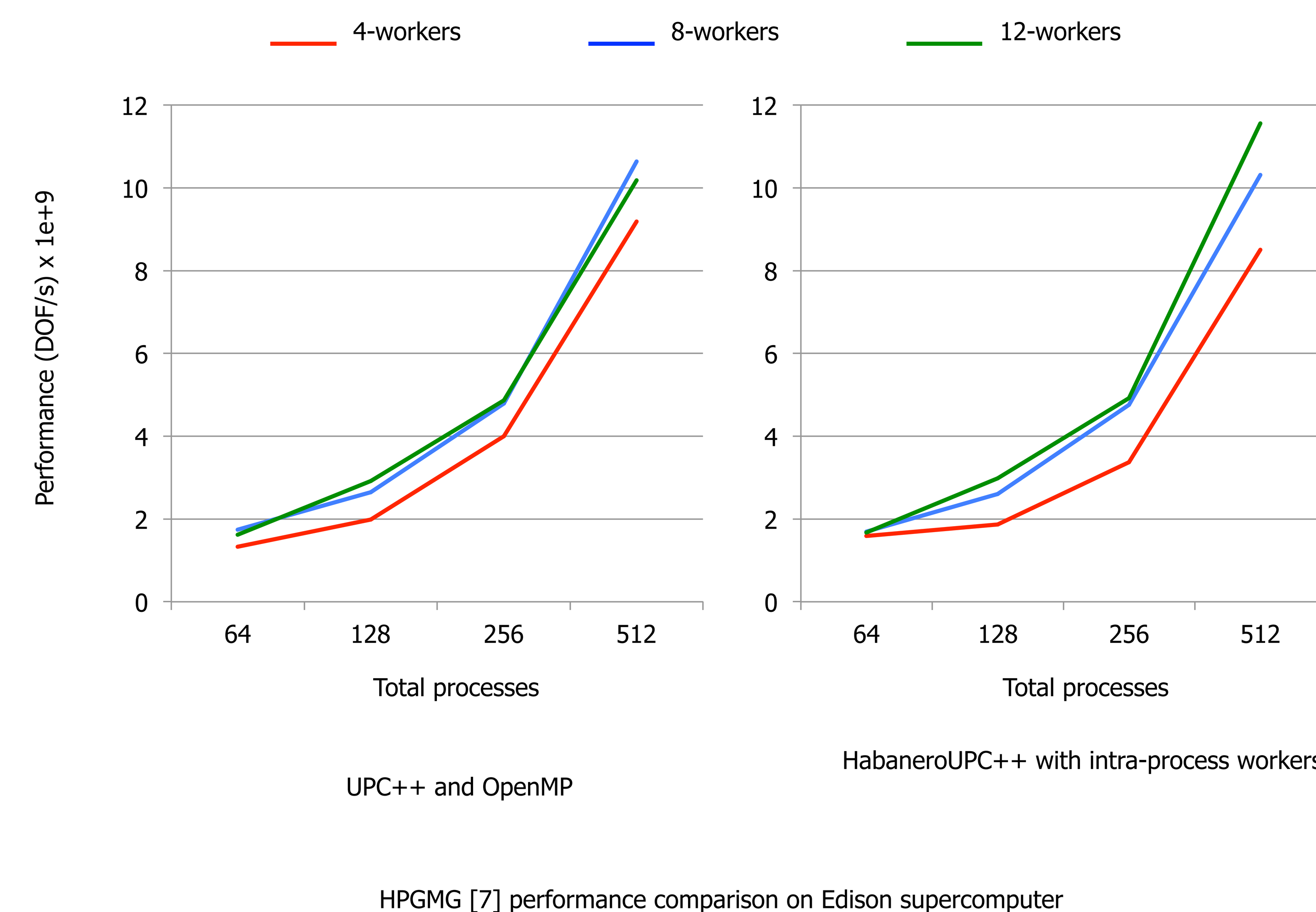
- Phalanx allows creating dependencies between asynchronous tasks executing either at GPU or host but lacks an intra-process work-stealing runtime.
- X10 supports intra-process (place) work-stealing but does not allow creating dependencies between host and accelerator tasks as Phalanx does.
- HCMPI [5] supports both distributed data-driven programming model as well as intra-process work-stealing but does not support accelerators.

Exploiting locality at each level of a NUMA memory hierarchy is crucial to scaling to highly parallel systems:

- Unlike X10, Phalanx exposes a hierarchical memory model (places)

3 Why use HabaneroUPC++ ?

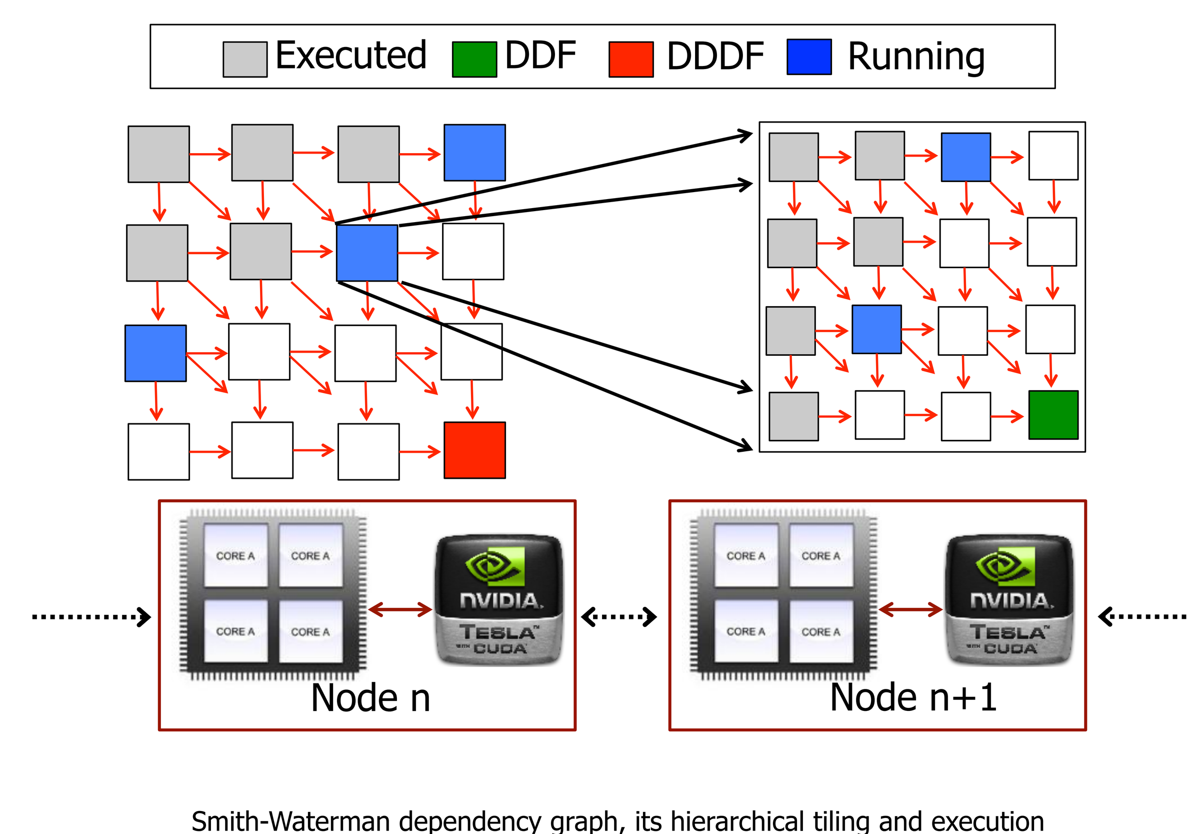
A highly scalable, compiler-free, distributed, multi-core, task-parallel PGAS library [6].



HPGMG [7] performance comparison on Edison supercomputer

4 Proposed Execution Model

Dataflow scheduling across all computational units within a node. Use dedicated communication and accelerator worker threads to manage resources.



Smith-Waterman dependency graph, its hierarchical tiling and execution

5 Launching Tasks Across GPUs

HabaneroUPC++ accelerator runtime will focus on an efficient implementation of :

- Multi-GPU management, load balancing across all GPUs and CPU cores
- Efficient host-device and inter-device communication, overlap with compute on work-stealing CPU and GPU threads
  - Automated by default, enable programmer hints
- Use of C++ functors to communicate computation to GPUs, variadic function templates to communicate data

```
class task1 : public cuda_task<task1,
    task2, int *, int> {
  __host__ __device__ void apply()(
      int i, int *A, int val) override
{
      async(test_functor(), A, val + 1);
   }
};

forasync(N, task1(), h_out, val);
```

References

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

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

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

[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] 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.

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

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