Accelerating CHAMPS on GPUs

Akihiro Hayashi
ahayashi@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

Sri Raj Paul
sriraj.paul@intel.com
Intel Corporation
Austin, Texas, USA

Vivek Sarkar
vsarkar@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

KEYWORDS

Chapel, Champs, GPUs, GPUIterator, GPUAPI

ACM Reference Format:
Akihiro Hayashi, Sri Raj Paul, and Vivek Sarkar. 2022. Accelerating CHAMPS on GPUs. In *Proceedings of CHIUW '22: The 9th Annual Chapel Implementers and Users Workshop (CHIUW '22).* ACM, New York, NY, USA, 3 pages. https://doi.org/XXXXXXX.XXXXXXX

EXTENDED ABSTRACT

There has been a growing interest in utilizing GPUs in large-scale systems. While we believe PGAS languages such as Chapel [1] are suitable not only for homogeneous nodes but also for heterogeneous nodes, GPU programming with PGAS languages in practice is still limited since there is still a big performance gap between compiler-generated GPU code and hand-tuned GPU code. Additionally, hand-optimization of CPU-GPU data transfers is an important contributor to this performance gap.

In our past work, we proposed the `GPUAPI` module [4], which includes a wide variety of Chapel-level GPU API that allows the user to write device memory (de)allocation and device-to-host/host-to-device data transfer in Chapel. While the user still has to write GPU kernels manually, the use of the `GPUAPI` module facilitates hand-optimization of CPU-GPU data transfers.

Furthermore, the `GPUAPI` module is designed to comply with Chapel's multi-resolution concept, where the user has the option of providing a high-level specification and also of diving into lower-level details to incrementally evolve their implementations for improved performance on multiple CPUs+GPUs nodes particularly when it is used with the `GPUIterator` module [4]. Specifically, we proposed to introduce the following spectrum of GPU programming abstraction in Chapel [3] and the `GPUAPI` module is mainly responsible for the MID-level/MID-LOW-level part:

- **HIGH-level/HIGH-MID-level**: The compiler compiles `forall` / `reduce` constructs to GPUs and generates all the host part required for GPU execution (HIGH). For the host part, the user has the option of using our GPUAPI to optimize device memory allocation and data transfer (HIGH-MID).
- **MID-level/MID-LOW-level**: The user writes 1) GPU kernels in a low-level GPU language, and 2) the host part in Chapel in either/both of two levels of abstraction: a Chapel programmer-friendly version (MID), and a thin wrapper version of raw GPU API routines (MID-LOW).
- **LOW-level**: The user writes a full GPU program in a low-level GPU language and call it from Chapel using the C interoperability feature.

In this talk, we discuss our experience in accelerating a real-world Chapel application, namely CHAMPS (CHApel Multi-Physics Simulation) [5], on GPUs with the `GPUIterator` and `GPUAPI` modules. Please note that, this year's talk focuses on discussing more details of the application part (e.g., profiling, results, CUDA code implementations, and providing a interface between and the CUDA and the Chapel layer) and new data transfer optimizations, whereas the last year's GPUAPI paper [4] mainly focused on the design and implementation of the module and its talk briefly introduced initial CHAMPS numbers as an example.

CHAMPS is a 3D unstructured finite-volume Reynolds Average Navier-Stokes (RANS) flow solver developed at Polytechnique Montréal and is written fully in Chapel. We use the potential flow solver in CHAMPS.

Our key contributions include:

(1) implementing the following GPU variants of CHAMPS's potential solver.
(2) discussing preliminary performance numbers on three CPU-GPU platforms and productivity evaluation results in terms of source lines of code (SLOC).

- **Chapel-CPU (Original)**: Implemented in Chapel using a `forall` with the default parallel iterator that is executed on CPUs.
- **Chapel-GPU**: Implemented using a `forall` with the `GPUAPI` and `GPUIterator` modules.
  - **MID-level**: All the GPU part except for GPU kernels is implemented using the MID-level API, which is a Chapel class based abstraction of GPU arrays.
  - **MID-LOW-level**: All the GPU part except for GPU kernels is implemented using the MID-LOW-level API, which is a set of thin wrappers for raw GPU API routines.
  - **LOW-level**: The GPU part is fully implemented in CUDA (on NVIDIA GPUs) or HIP (on AMD GPUs).

In CHAMPS, there is a function that contains two `forall` loops that are invoked multiple times, which accounts for 89.1% of the end-to-end execution time based on our CPU profiling on an Intel Xeon Gold 6148 machine. Essentially, we prepared a CUDA kernel for each `forall` loop and invoke it using the `GPUIterator` module[1]. For device memory allocations and data transfers, as above, we prepared MID-level, MID-LOW-level, and LOW-level versions. Also, we manually applied a data transfer optimization to 1) hoist CPU to GPU transfers that transfer read-only data to the beginning of the

[1]We used the `hipify` tool to convert the CUDA implementation to HIP.

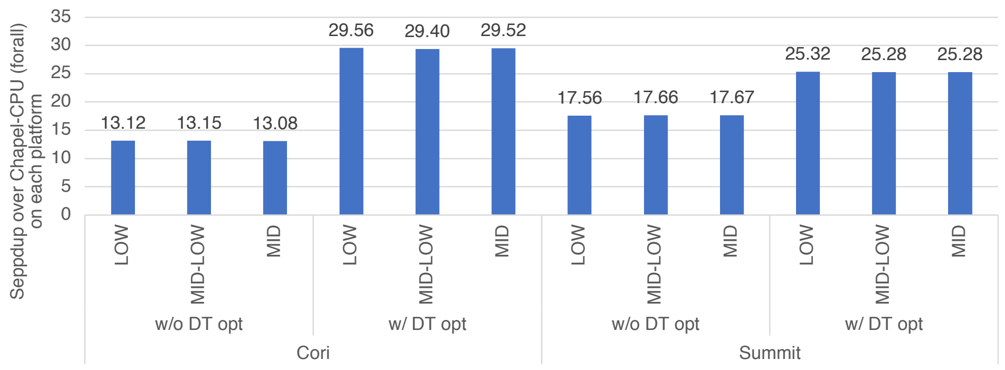

Figure 1: End-to-end performance improvements over Chapel-CPU on Cori-GPU and Summit (single-node, 1GPU/node).

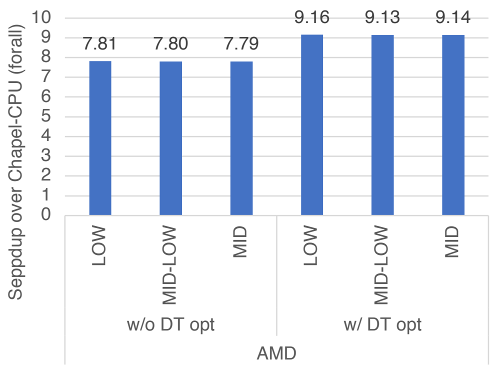

Figure 2: End-to-end performance improvements over Chapel-CPU on the AMD server (single-node, 1GPU/node).

most time-consuming function, and 2) defer GPU to CPU transfers until the end of the function. It is worth noting that the second most time consuming part is a linear solver part, for which we prepared a cuSOLVER version of it (NVIDIA GPUs only).

We present the performance results on three platforms. The first platform is the Cori GPU cluster at NERSC, each node of which consists of two sockets of 20-core Intel Xeon Gold 6148 running at 2.40 GHz with a total main memory size of 384GB and 8 NVIDIA Tesla V100 GPUs, each with 16 GB HBM2 memory, connected via PCIe 3.04. The second platoform is the Summit supercomputer at ORNL, which consists of the IBM Power System AC922 nodes. Each node contains two IBM POWER9 running at 3.45GHz with a total main memory size of 512GB and 6 NVIDIA Tesla V100 GPUs, each with 16GB HBM2 memory, connected via NVLink. On both platforms, in this preliminary experiment, we used a single GPU of a single node. The third platform is a single-node AMD server, which consists of 12-core Ryzen9 3900X running at 3.8GHz and a Radeon RX570 GPU with 8GB memory.

Preliminary performance results: Figure 1 and Figure 2 show the end-to-end performance improvements over Chapel-CPU on the three platform. As shown in Figure 1, our Chapel-GPU variants give significant speedups of up to 29.5x on Cori and 25.3x on Summit. The result also shows the data transfer optimization gives 2.2x and 1.4x performance improvements on Cori and Summit respectively. Similarly, Figure 2 shows our Chapel-GPU variants gives 9.1x performance improvement and the data transfer optimization gives 1.1x improvement. More importantly, the results show that there is no significant performance difference between the LOW, MID-LOW, and MID versions[2], which means that our modules are able to provide higher- yet Chapel-level GPU API with no performance loss.

Level	Host (Chapel)	Host (CUDA)	Kernel (CUDA)
LOW	172	117	361
MID-LOW	265	5	361
MID	161	5	361

Table 1: Source code additions and modifications required for using the `GPUAPI` module in CHAMPS's potential solver in terms of source lines of code (SLOC).

Preliminary productivity results: Table 1 shows source code additions and modifications required for using the GPUAPI. We measure the productivity in term of source lines of code[3]. The goal of this productivity experiment is to demonstrate SLOC for the host part in Chapel and CUDA are reduced when the MID-level API is used. Note that the CUDA kernel part is out of the scope of this paper. The results show 1) the use of MID/MID-LOW level API significantly reduces the SLOC for Host (CUDA) part (117⟶5), and 2) the use of the MID-level API further decreases the SLOC for the Host (Chapel) part thanks to our higher-level abstraction of device memory (such as `GPUArray` and `GPUJaggedArray`, for more details please see [2]). Let us reiterate that the MID-level API simplifies the Host (Chapel) part more than what it appears as the lines of code reduction because it avoids the explicit manipulation of raw C pointers.

[2]While there are slight differences, such differences are mostly from fluctuations in an initialization part on CPUs, which is out of our focus.
[3]Our definitions of source code "lines" is based on common usage.

ACKNOWLEDGMENTS

The authors would like to thank Éric Laurendeau, Matthieu Parenteau, and Anthony Bouchard for allowing us to obtain the source code of CHAMPS and giving us valuable feedback on our GPU implementation.

This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. Also, this research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

REFERENCES

[1] Bradford L. Chamberlain. 2011. Chapel (Cray Inc. HPCS Language). In *Encyclopedia of Parallel Computing*. 249–256. https://doi.org/10.1007/978-0-387-09766-4_54

[2] Akihiro Hayashi et al. 2021. GPUIterator and GPUAPI Documentation. https://ahayashi.github.io/chapel-gpu/.

[3] A. Hayashi, S. Raj Paul, and V. Sarkar. 2020. Exploring a multi-resolution GPU programming model for Chapel. In *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 675–675. https://doi.org/10.1109/IPDPSW50202.2020.00117

[4] A. Hayashi, S. Raj Paul, and V. Sarkar. 2021. GPUAPI: Multi-level Chapel Runtime API for GPUs. In *The 8th Annual Chapel Implementers and Users Workshop(CHIUW)*.

[5] Matthieu Parenteau, Simon Bourgault-Cote, Frédéric Plante, Engin Kayraklioglu, and Eric Laurendeau. 2021. Development of Parallel CFD Applications with the Chapel Programming Language. In *AIAA Scitech 2021 Forum*. https://doi.org/10.2514/6.2021-0749