Performance Portable Magnetohydrodynamics for the Exascale Era

Forrest W. Glines[1,2,3], Philipp Grete[1], Brian W. O'Shea[1,2]

[1]*Department of Physics and Astronomy,* [2]*Department of Computational Mathematics, Science and Engineering, Michigan State University,* [3]*Sandia National Laboratory.*

Email: glinesfo@msu.edu, grete@pa.msu.edu, oshea@msu.edu

SAND2019-6874C

Introduction

Computational plasma models such as magnetohydrodynamics (MHD) are essential tools in modern plasma physics. They can complement experiments, inform future research, and provide insight into plasma phenomena that are difficult to create in a lab. For simulations to more closely match reality and deepen our understanding of plasmas, we need the computational resources of increasingly larger supercomputers.

However, constraints in computer chip manufacturing are leading to new computer architectures such as many-core processors and graphics processing units (GPUs) to make up the majority of next generation computer clusters. Each new architecture can require a non-trivial rewrite of a simulation code. A current goal in supercomputing is the creation of paradigms for writing performance portable code: code that can run efficiently at high performance on many different architectures.

To explore the development of performance portable plasma codes, we created K-Athena [2], a performance portable, CPU and GPU performant conversion of the CPU-only astrophysical MHD code Athena++ [3] using Kokkos [1], a performance portability library.

Kokkos Programming Methodology

```
for( int k = ks; k < ke; k++){
  for( int j = js; j < je; j++){
    #pragma omp simd
    for( int i = is; i < ie; i++){
      /* Loop Body */
      u(k,j,i) = ...
}}}
```

Example triple `for` loop for a typical operation in a finite volume method on a structured mesh such as in a code like Athena++, where `ks`, `ke`, `js`, `je`, `is`, and `ie` are loop bounds and `u` is a field quantity. **When converting these stenciled operations to GPUs using Kokkos, the loop body is mostly unchanged**.

```
using namespace Kokkos;
parallel_for( MDRangePolicy<Rank<3>>
     ({ks,js,is},{ke,je,ie}),
  KOKKOS_LAMBDA(int k, int j, int i){
    /* Loop Body */
    u(k,j,i) = ...
});
```

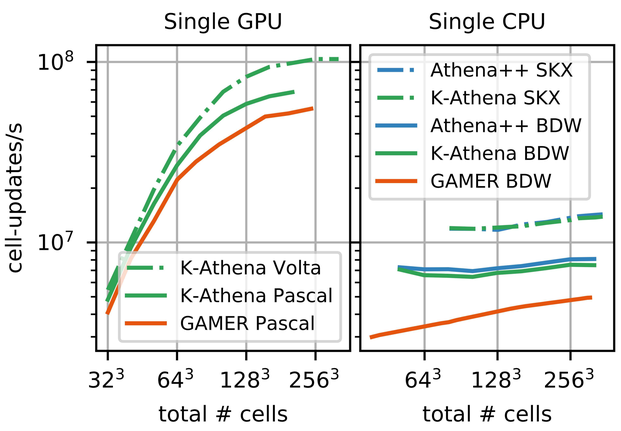
Example `for` loop using Kokkos. The loop body is reformulated into a lambda function and passed into `Kokkos::parallel_for` to execute on the target architecture. The class `Kokkos::MDRangePolicy` specifies the loop bounds. The array `u` is now a `Kokkos::View`, a Kokkos building block that allows transparent access to CPU and GPU memory.

```
using namespace Kokkos;
parallel_for(team_policy(nk*nj, AUTO),
  KOKKOS_LAMBDA(member_type team_mem){
    int lr = team_mem.league_rank();
    int k = lr / nj + ks;
    int j = lr % nj + js;
    parallel_for(
    TeamThreadRange<>(team_mem,is,ie),
      [&](int i) {
      /* Loop Body */
      u(k,j,i) = ...
});});
```

Another approach using Kokkos' team based parallelism through the class `Kokkos::TeamThreadRange`. This interface is closer to the underlying parallelism used by the backend such as CUDA blocks on GPUs and SIMD vectors on CPUs. We found that it was easier to optimize performance using team based parallelism.

3D Linear Wave Performance

In order to assess the performance of K-Athena, we conducted scaling tests using a 3D linear MHD wave as a proxy for research applications of MHD. All numbers shown here were obtained using a second order scheme consisting of a Van Leer integrator, piecewise linear reconstruction, and a Roe Riemann solver. We compare the performance against the base Athena++ CPU version as well as against GAMER, an MHD code written in CUDA using the same algorithms. [4].

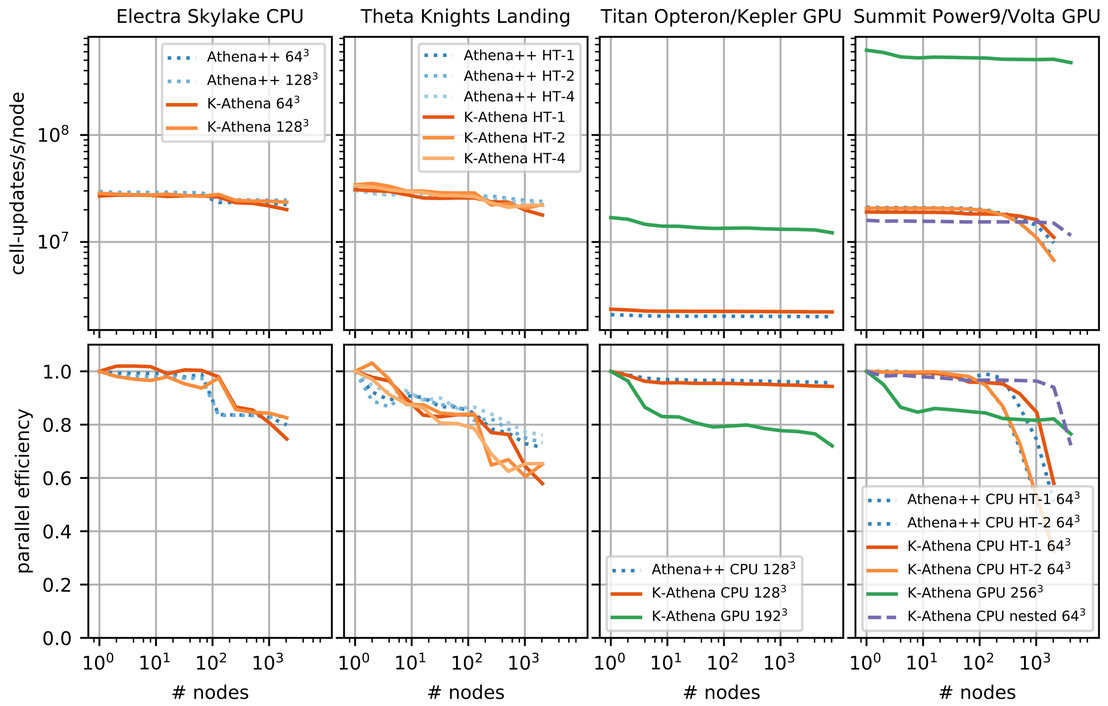


Number of cell updates per second vs. problem size (number of cells) for Athena++, K-Athena, and GAMER, on GPUs (**left**) and CPUs (**right**).

On Nvidia Tesla P100 (Pascal) GPUs K-Athena performs comparably to GAMER, which is optimized for CUDA and uses more hardware features such as CUDA streams. **K-Athena reaches a peak performance of 1.04×10^8 cell-updates/s on a single Nvidia Tesla V100 (Volta) GPU.**

On CPUs K-Athena retains virtually the same performance as Athena++, i.e., reaching 1.43×10^7 cell-updates/s on a single CPU (20 core Intel Xeon Gold 6148).

GAMER timings from [4]. All other results were generated for this work.
SAND No. XXXXXXXXXX



Weak scaling on different supercomputers for K-Athena and Athena++. The top row shows cell-updates per second per node to directly compare performance on different systems, the bottom row shows parallel efficiency normalized to single node performance, on the Electra, Theta, Titan, and Summit. **Running on 4,096 nodes with 24,576 GPUs on Summit, K-Athena achieved 1.94 trillion cell updates per second.**

System Details: NASA Electra: two 20-core Intel Xeon Gold 6148 CPUs per node ALCF's Theta: one 64-core Intel Xeon Phi 7230 (per node, OLCF's Titan: one AMD Opteron 6274 16-core CPU and one Nvidia K20X GPU per node, OLCF's Summit: two 21-core IBM POWER9 CPUs and six Nvidia V100 (Volta) GPUs per node.

Discussion

Porting Athena++ to GPUs using Kokkos required only a moderate effort in code developement. The ease of development is largely due to the good design of Athena++ and consistent use of data structures.

Our main goal was to enable GPU-accelerated simulations while maintaining performance on CPUs using a single code base. We achieved this goal by reaching 93-100% of the original Athena++ performance on CPUs, and reaching comparable performance to GAMER (a specialized CUDA code) on GPUs.

With it's high efficiency of computing resources, K-Athena will allow us to study magnetized turbulence, galaxy clusters, and other plasmas at higher resolutions than what was previously possible.

Acknowledgments

We thank the Kokkos developers, particulary Christian Trott and Steve Bova, and the organizers of the 2018 Performance Portability with Kokkos Bootcamp for their help using Kokkos in Athena++. We thank Kristian Beckwith for inspiring discussions on Kokkos. We thank the Athena++ team for making their code public and for their well designed code. We acknowledge funding by NASA Astrophysics Theory Program grant #NNX15AP39G. Code development, testing, and benchmarking was made possible through various computing grants including allocations on NASA Pleiades (SMD-16-7720), OLCF Titan (AST133), XSEDE Comet (TG-AST090040), and Michgian State University's High Performance Computing Center. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

References

[1] H. Carter Edwards, C. R. Trott, D. Sunderland, *Journal of Parallel and Distributed Computing*, Domain-Specific Languages and High-Level Frameworks for High-Performance Computing **2014**, *74*, 3202–3216.

[2] P. Grete, F. W. Glines, B. W. O'Shea, *submitted to IEEE Transactions on Parallel and Distributed Systems* **2019**, arXiv:1905.04341 [cs.DC].

[3] C. J. White, J. M. Stone, C. F. Gammie, *The Astrophysical Journal Supplement Series* **2016**, *225*, 22.

[4] U.-H. Zhang, H.-Y. Schive, T. Chiueh, *The Astrophysical Journal Supplement Series* **2018**, *236*, 50.