Performance Portable Finite Volume Magnetohydrodynamics for the Exascale Era

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

MICHIGAN STATE UNIVERSITY

[1]Department of Physics and Astronomy, Michigan State University, [2]Department of Computational Mathematics, Science and Engineering, Michigan State University, [3] For further information, email glinesfo@msu.edu, grete@pa.msu.edu, oshea@msu.edu

Introduction

Computational plasma models such as magneto-hydrodynamics (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 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 modified the adiabatic ideal MHD solver in Athena++[2], a finite volume astrophysical MHD code that is highly optimized for CPUs, to run efficiently on both CPUs and GPUs, 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) = ...
}}}
```

Listing 1: 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) = ...
});
```

Listing 2: 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 architecuture. 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) = ...
});});
```

Listing 3: 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 our Athena++ version using Kokkos (K-Athena) we conducted scaling tests for a 3D linear MHD wave as a proxy for research applications of MHD. All numbers shown have been 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 original Athena++ CPU version as well as against GAMER, a comparable MHD code using the same numerical algorithms but written directly in CUDA and tuned to achieve high performance [3].

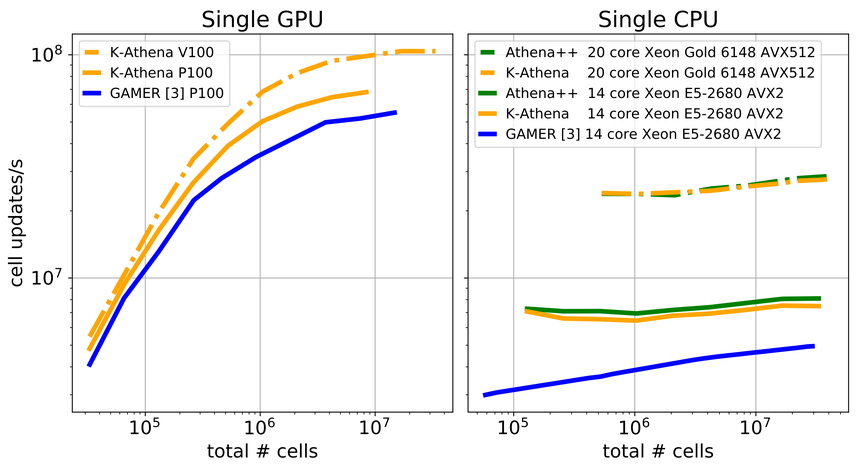

Figure 1: Number of cell updates per second vs. problem size (number of cells) for Athena++, K-Athena, and GAMER, on GPUs (**left**) and CPUs (**right**).

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 [3]. All other results were generated for this work.

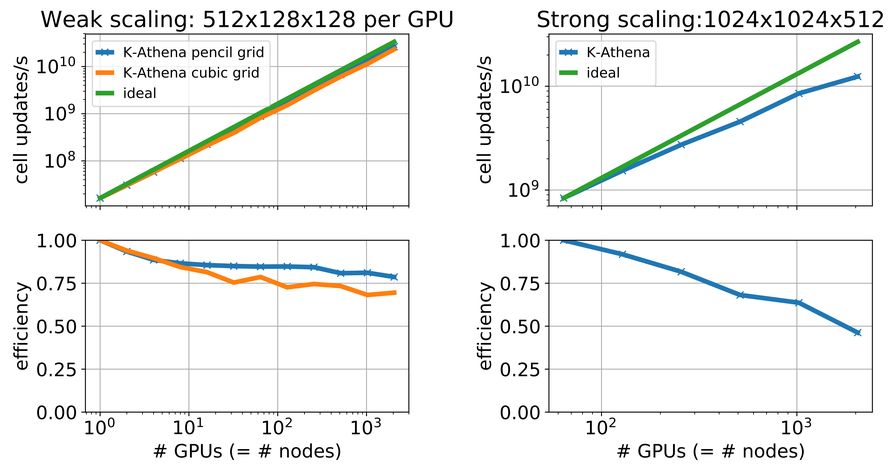

Figure 2: Number of cell updates per second (**top**) and parallel efficiency, (the relative performance per GPU) vs. the number of GPUs used, for weak scaling (**left**) and strong scaling (**right**). Scaling tests were performed on Titan at Oak Ridge National Laboratory, which has one Nvidia Tesla K20x GPU per node.

For weak scaling the problem size per GPU remains constant. We tested grids of 512x128x128 arranged in both a cubic domain and along a line (pencil), in which case communications are minimized. **K-Athena achieves a parallel efficiency of 70-80% up to 2048 GPUs.**

For strong scaling the total problem size (1024x1024x512) remains constant. **K-Athena achieves ∼50% parallel efficiency in strong scaling with 64 to 2048 GPUs.**

Discussion

Porting the CPU optimized MHD code 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.

We are still evaluating different loop structures for optimal performance, which is, for example, related to the (dis)ability of different compilers to automatically vectorize inner loops.

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.

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

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