Cosmic Ray Influence in Multiphase Environments Scaling and Timing

PI: Chad Bustard, Kavli Institute for Theoretical Physics
Co-PI: S. Peng Oh, University of California - Santa Barbara

Computational Method

Our code of choice is Athena++ [1], a higher-order directionally unsplit Godunov scheme that solves the ideal magnetohydrodynamics (MHD) equations. This public code is a rebuild of the popular Athena code, used for many applications in astrophysics over the last two decades, but not optimized for modern computing architectures. Athena++, on the other hand, uses a high percentage of vectorized instructions: according to the recent Athena++ methods paper [1], about 85% of the MHD code is vectorized using AVX/AVX2/AVX512, and compared to other prominent astrophysical MHD codes, the per-core performance of Athena++ is as much as a factor of 10 faster when tested on the Intel Skylake processors.

The version of Athena++ that we use has additional physics modules, most notably a cosmic ray module [2] that evolves additional equations for cosmic ray energy and flux via a computationally efficient two-moment method adapted from simulations of radiative transfer [3, 4]. This cosmic ray algorithm is more accurate and vastly speeds up our simulations compared to other cosmic ray implementations [5], as the stable timestep scales linearly instead of quadratically with the cell size.

Performance And Scaling Of Athena++

Athena++ gives us substantial control over mesh decomposition for parallel simulations. The computing domain is decomposed into small units called MeshBlocks that have the same logical size and are stored on a tree structure. We find that the best performance is achieved when each MeshBlock contains either 32^3 or 64^3 cells, which minimizes communication and overhead compared to MeshBlocks with fewer cells. We also benefit from years of experience in our group fine-tuning Athena and Athena++ for optimal use. We have experimented with compiler optimization, finding that the Intel C++ compiler combined with compiler flags **-O3 -std=c++11 -xMIC-AVX512 -qopenmp-simd -qopt-prefetch=4** give the best performance to-date. While not as important, we also take care to use the HLLD Riemann solver as our MHD default, which is slighly faster than the Roe solver.

Athena++ scales extremely well on Stampede2. To characterize the scalability, we run a set of 3D MHD simulations with and without cosmic rays on the Knights Landing nodes, each utilizing 64 cores per node, and we track the number of cell updates per CPU per second – a metric of the code performance. We run each simulation for between 50 and 600 timesteps (this choice makes very little difference). The left panel of Figure 1 shows the strong scaling for domain sizes of 256^3 and 512^3, each utilizing 32^3 cells per MeshBlock. We find incredibly good scaling, with virtually no drop-off in performance until we run the 512^3 simulation with 4096 processors. Figure 2 shows weak scaling; in this case, all simulations include cosmic rays and use MeshBlocks of 64^3 cells. The scaling is almost perfect when the 64 processor simulation is considered as a baseline, while the

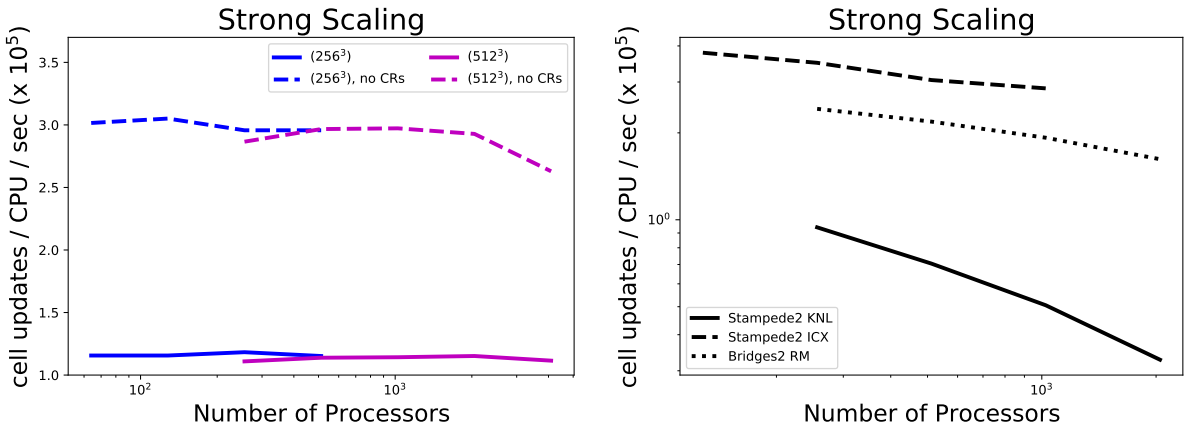

Figure 1: *Left:* Strong scaling of code performance, measured in cell updates per CPU per second, for 3D MHD simulations on Athena++ with cosmic rays included (solid lines) and without cosmic rays (dashed lines). Each MeshBlock has 32^3 cells, allowing Athena++ to run close to its peak performance. For both grid setups tested (256^3 and 512^3 cells), strong scaling is > 90 % out to 4096 processors (64 nodes), consistent with tests done in the Athena++ methods paper ([1]). Including cosmic rays increases computational cost by a factor of 2-3, about the same cost increase as going from pure hydro to MHD. *Right:* Scaling study of our fiducial 512^3 turbulent box simulations on Stampede2's KNL and ICX nodes, as well as the Bridges2 RM nodes. Decreased efficiency with increasing number of processors is due to using fewer cells per Meshblock than the optimal 32^3. To balance wall-clock time and efficiency, we run most of our published simulations on 1024 cores.

efficiency is $\approx 83\%$ compared to the single processor run. Very similar results are achieved when we instead use MeshBlocks of 32^3 cells.

We also compare the Knights Landing (KNL) performance to that of new ICX nodes on Stampede2 and the RM nodes on PSC-Bridges2. The right panel of Figure 1 shows a scaling study specific to our fiducial resolution 512^3 turbulent box simulations. The stirring module is essential for our turbulence simulations but can represent a slow-down if not done carefully. Trial-and-error has shown us that we achieve the greatest computational efficiency when we use pencil decomposition; that is, when we decompose our grid of e.g. N^3 cells into MeshBlocks each with N x 8 x 8 cells. This is the approach we've taken in previous simulations and the tests shown in the right panel of Figure 1. The new ICX nodes on Stampede2 are 3-4x faster than the KNL nodes. The Bridges2 RM processors are intermediate but almost as fast as the ICX processors (just 1.3x slower), making them a very viable option for our proposed simulations. Note the decreased efficiency of these runs as we increase the number of cores; unlike in the left panel of Figure 1, where we use cubic Meshblocks of 32^3 cells, we here assign each processor to one Meshblock and therefore decrease the number of cells per Meshblock as we increase the number of processors. The runs using 256 processors are most efficient because they near the sweet-spot of 64^3 cells per Meshblock, while runs on more cores have far fewer (and less efficient) cells per Meshblock. This exhibits some of the nuances we've taken into consideration when running our simulations. **To strike a balance between efficiency and simulation wall-clock time, we've chosen so far to run these 512^3 simulations with 1024 cores.**

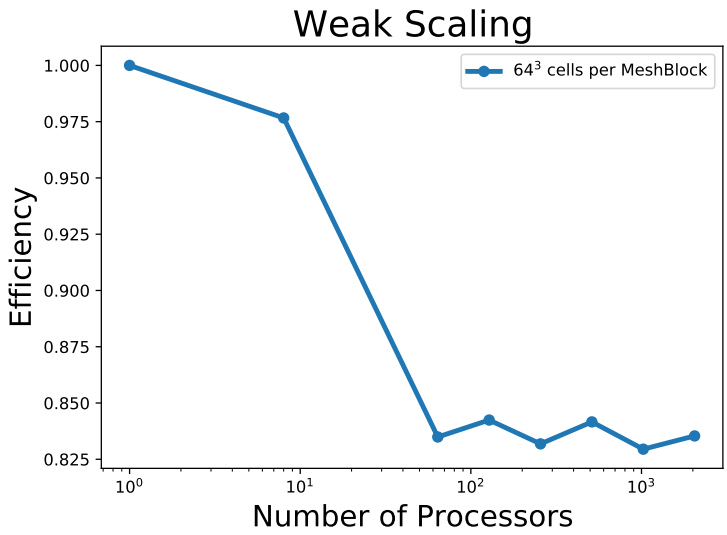

Figure 2: Weak scaling, with efficiency measured relative to a single-core simulation with one MeshBlock capturing a grid of 64^3 cells. Simulations utilizing 64 cores per node run at an efficiency of $\approx 83\%$ compared to the serial simulation. Scaling to more nodes is essentially perfect, as also noted in [1].

Additional Overhead

I/O: Even for our largest simulations run on a 512^3 grid, we find that the input/output (I/O) cost is negligible (of order 1%) compared to the actual computation cost. This is largely because we generate HDF5 output and checkpoint files very infrequently; they are primarily used for visualization rather than analysis purposes, and we instead perform the most computationally-expensive analysis during the simulation and output bulk quantities of interest at much finer time resolution to a tabular data file. For visualization, we use the yt software [6], an open-source, python-based toolkit for visualization and analysis, tailored towards astrophysical applications. yt was developed to process large 3D datasets, such as the HDF5 files produced by Athena++, and has been parallelized with MPI to speed-up post-processing. We have run yt extensively for a number of research projects on Stampede2.

MHD and the Cosmic Ray Module: Our results, both in terms of scalability and absolute speed, are consistent with tests outlined in the Athena++ methods paper ([1]). For 3D MHD, we find our simulations are about a factor of 2 slower than the pure-hydro performance. As shown in Figure 1, including cosmic rays generally slows the performance by another factor of 2-3, but this slowdown is vastly outweighed by the benefits of this method compared to other existing cosmic ray algorithms. Namely, the timestep of this approach scales as $\mathrm{O}(\Delta x)$ rather than traditional explicit schemes that scale as $\mathrm{O}(\Delta x^2)$, and it has improved accuracy at extrema in cosmic ray energy.

As one can also see from Figures 1 and 2, the cosmic ray method does not impact the excellent scaling of Athena++. While implicit schemes that also have $\mathrm{O}(\Delta x)$ time-stepping require matrix inversion over the entire simulation domain, thereby affecting parallelization, this method avoids this setback by explicitly differencing the transport and source terms. *This maintains excellent weak scaling and opens the door to efficiently simulate cosmic ray transport at unprecedented resolution on modern computing architectures.*

Turbulent Stirring: For all simulations tested, each with pencil decomposition, we find that including the FFT unit responsible for turbulent driving decreases code performance by less than 2%.

Radiative Cooling: Additionally, we implemented an exact cooling routine [7], only relevant for our proposed low-cost simulations of cloud evolution. Compared to other explicit and implicit cooling algorithms, this routine is competitive in terms of speed and achieves greater accuracy, making it a natural choice for our cloud acceleration simulations with radiative cooling. Overall, when this routine is turned on, the added computational expense is $\approx 10\%$.

Simulation Timing For Estimation Of SUs

Because we have already run simulations on a 512^3 grid with our fiducial and thoroughly-tested choices of domain size, simulation duration, and v_m (see the Justification section in the Main Document), **we already know how many SUs each 512^3 simulation consumes on Stampede2's ICX nodes**:

- For $\mathcal{M}_s \sim 0.5$, we need ~ 1600 SUs per simulation
- For $\mathcal{M}_s \sim 0.15$, we need ~ 5333 SUs per simulation

The larger number of SUs for lower Mach number turbulence reflects that the eddy turnover time (and hence simulation duration) is longer by a factor of 0.5/0.15. Our convergence test simulation on a 1024^3 grid will be run with $\mathcal{M}_s \sim 0.5$ and cost $16 \times 1600 = 25{,}600$ SUs.

For clarity, we also include here a conversion between SUs on Stampede2's ICX nodes and the Bridges2 RM nodes, based on our tests shown in Figure 1: **1 SU on Stampede2** ≈ 83.2 **SUs on Bridges2**, taking into account that the ICX processors are on average 1.3x faster than the Bridges2 processors and that SUs on Bridges2 are calculated by number of processors used instead of number of nodes used.

References

[1] James M. Stone, Kengo Tomida, Christopher J. White, and Kyle G. Felker. The Athena++ Adaptive Mesh Refinement Framework: Design and Magnetohydrodynamic Solvers. , 249(1):4, July 2020.

[2] Yan-Fei Jiang and S. Peng Oh. A New Numerical Scheme for Cosmic-Ray Transport. , 854(1):5, Feb 2018.

[3] Yan-Fei Jiang, James M. Stone, and Shane W. Davis. A Godunov Method for Multidimensional Radiation Magnetohydrodynamics Based on a Variable Eddington Tensor. , 199(1):14, March 2012.

[4] Yan-Fei Jiang, James M. Stone, and Shane W. Davis. An Algorithm for Radiation Magnetohydrodynamics Based on Solving the Time-dependent Transfer Equation. , 213(1):7, July 2014.

[5] Prateek Sharma, Phillip Colella, and Daniel F. Martin. Numerical Implementation of Streaming Down the Gradient: Application to Fluid Modeling of Cosmic Rays and Saturated Conduction. *arXiv e-prints*, page arXiv:0909.5426, Sep 2009.

[6] Matthew J. Turk, Britton D. Smith, Jeffrey S. Oishi, Stephen Skory, Samuel W. Skillman, Tom Abel, and Michael L. Norman. yt: A Multi-code Analysis Toolkit for Astrophysical Simulation Data. , 192(1):9, January 2011.

[7] R. H. D. Townsend. An Exact Integration Scheme for Radiative Cooling in Hydrodynamical Simulations. , 181(2):391–397, April 2009.