Numerical Modeling of Space Plasma Flows: ASTRONUM-2012
ASP Conference Series, Vol. 474
N. V. Pogorelov, E. Audit, and G. P. Zank, eds.

Hybrid Parallelization of Adaptive MHD-Kinetic Module in Multi-Scale Fluid-Kinetic Simulation Suite

Sergey N. Borovikov[1], Jacob Heerikhuisen[1,2], and Nikolai V. Pogorelov[1,2]

[1]*Center for Space Plasma and Aeronomic Research, University of Alabama in Huntsville, AL 35805*

[2]*Physics Department, University of Alabama in Huntsville, AL 35805*

Abstract. The Multi-Scale Fluid-Kinetic Simulation Suite has a computational tool set for solving partially ionized flows. In this paper we focus on recent developments of the kinetic module which solves the Boltzmann equation using the Monte-Carlo method. The module has been recently redesigned to utilize intra-node hybrid parallelization. We describe in detail the redesign process, implementation issues, and modifications made to the code. Finally, we conduct a performance analysis.

1. Introduction

Multi-Scale Fluid-Kinetic Simulation Suite (MS-FLUKSS) is a state-of-the-art tool for modeling partially ionized plasma flows. Charged particles are modeled by solving the ideal MHD equations whereas neutral particles are modeled either by solving the Boltzmann equation using Monte-Carlo method (Baranov & Malama 1993; Heerikhuisen et al. 2006) or using a multi-fluid approach (Zank et al. 1996). In the latter method, different populations of neutral atoms are assumed to have Maxwellian distribution functions and each fluid is described by the Euler equations. The multi-fluid approach, although computationally inexpensive, works best when the regions of origins of different fluids are well defined. That is not the case in simulations of the heliosphere for interstellar magnetic fields above 3μG or for the long heliotail simulation where the population of neutrals born in the heliosheath has very different properties close to the termination shock from those in the distant tail. As a result, modeling neutrals by solving the Boltzmann equation becomes a necessity in many cases even if it is more computationally expensive.

The Boltzmann equation for neutrals and the ideal MHD equations for plasma are solved together via global iterations. For the current configuration of plasma, the Monte Carlo code is run until appropriate statistics are accumulated to calculate the source terms for the plasma. After that, the plasma code is run for some time. This procedure is repeated until a steady state is reached. A Monte Carlo method for solving the Boltzmann equations is based on injecting test particles into a computational region and gathering statistics for the charge-exchange source terms instead of solving the 6D Boltzmann equation directly. The kinetic module was successfully developed and tested (Heerikhuisen et al. 2006; Pogorelov et al. 2009) and its 1D and 2D versions were successfully integrated into the MS-FLUKSS (Borovikov et al. 2008). 3D simulations, however, have been performed until recently without adaptive mesh refinement

(AMR) capabilities of the MS-FLUKSS (Heerikhuisen & Pogorelov 2011). This created a lot of inconveniences in practical usage including complicated job scripts to run the code on parallel computers and unnecessary utilization of file systems for data exchange. However, a main drawback of the kinetic code was the fact it was using solely MPI for parallelization. Each MPI task had its own copy of plasma data which means that a 12-core node had 12 allocated identical arrays. This led to serious limitations on the physical problems that could be solved with the code. For example, high resolution simulations of the solar wind (SW) interaction with the local interstellar medium (LISM) or astrotail modeling could not be conducted with the old version of the code. Therefore, the following list of improvements was created after a careful evaluation of the existing code.

- Plasma data and arrays storing the source terms for the MHD code must be shared among the cores of a single node. This can be done by switching parallelization approach from a pure MPI to a hybrid (MPI+OpenMP) method, where particles are distributed between MPI tasks and threads. The major challenge is to update the source term arrays when charge exchange occurs between neutrals and ions preventing data racing and performance deterioration.

- Load balancing problem. The load balancing should be a 2-level algorithm that guarantees an even workload between nodes and threads within a single node.

- Input/output (IO) modifications. The kinetic module is written in Fortran and uses Fortran binary files to output data. These files are not easily portable between different machines or programming languages because Fortran adds record markers to files which are compiler and system specific. If one needs to read a file using a different language, e.g., C++, these markers must be treated manually by skipping certain fragments of a file. Since the size of the markers is not regulated by the Fortran standard, usage of Fortran files is very inconvenient.

- Full 64-bit support. The code must be able to handle more than 2 billion particles.

2. Modifications of the kinetic code

The plasma module in the MS-FLUKSS has domain-driven parallelization, i.e., a computational region is split between processors and every MPI task computes only in the specified region. The kinetic code splits the workload between ranks by ensuring approximately even distribution of particles among cores. The plasma data is gathered into a set of arrays (each array represents a level of refinement in the plasma code) which is passed from the main C++ code to the kinetic code using a specially created interface. A similar set of arrays is created and passed to the kinetic code for collection of the source terms and the neutral atom distribution. The arrays are shared between threads that are spawned in the main subroutine of the kinetic code.

When a neutral particle experience charge exchange, the information about this event must be stored in the arrays for source terms. This can be done, for example, by creating a critical section that prevents simultaneous updates of the arrays. However, this approach will lead to a significant performance deterioration because the arrays are updated frequently and the threads will spend a lot of time waiting to enter the critical region. To solve this problem, a special buffer is created which is an array of special

data structure representing a single charge exchange event. This user-defined data structure has the following fields: a) index of the finest level in nested grids hierarchy where charge exchange occurs; b) position where charge exchange happens; c) physical quantities of the event (mass loss, momentum and energy data). When charge exchange occurs, the corresponding data is written to the buffer, not to the global arrays. The buffer is an array of the data structure and its size is chosen on a random basis for each thread to make simultaneous requests to update the arrays highly unlikely. Its typical size varies from 200,000 to 600,000 elements. When the buffer is full it is flushed to the main array. This is done via a critical section in the code.

The load balancing algorithm was also completely rewritten. The old version used an iterative approach when two MPI processors with the greatest and the least number of particles are identified. The processors perform particle exchange to have even number of particles. This procedure is repeated until disbalance between particles is less than a predefined threshold. This approach is not optimal because each iteration requires several collective communications, which are expensive. The new approach uses a similar idea, but its implementation is completely different. An MPI task that has the largest number of particles sends excess particles to a task with the least number of particles. If the excess is too big, some particles are sent to a task having the second least number of particles, etc. On the other hand, if deficit of particles is too big on some processor, it can receive particles from different MPI tasks. Each MPI task receives information about the number of particles the other tasks have. This is the only global collective communication. This information allows each rank to calculate the average number of particles per core and determine the communication scheme for load balancing. Of more interest, is that each rank knows the whole communication pattern (which of them will send/receive and how many particles will be transfered). Each MPI rank initiates a series of non-blocking send or receive calls. When the communication is complete the load balancing within node is performed using a similar approach.

Input/output is now performed using the hdf5 format. We take full advantage of parallel IO subroutines and parallel file systems. The chosen file format is easily transferable and manageable.

Initially, the new code demonstrated a very poor hybrid performance. It did not scale with the number of threads, moreover, the performance noticeably deteriorated as number of threads increased. Performance analysis revealed that the code spent most of its time generating random numbers. Eventually, it was discovered that the GNU Fortran random number generator shares seeds among threads and uses mutexes to regulate their update. This significantly deteriorated the overall performance. We have implemented our own random number generator where each thread has its own set of seeds, which has resolved this problem.

3. Performance of the new code

Our first test is to compare the performance with a different number of threads per MPI task. This test was performed on Cray XT5 Kraken with two six-core AMD Istanbul processors per node. We use 16 nodes and 50 million particles in our test case. The runs ranged from 192 MPI tasks and no threads, to 16 MPI tasks each spawning 12 threads. The results are summarized in Table 1. If hybrid parallelization is not used, the run time is 180 seconds. The code works faster when two threads are used, because the number of MPI calls is reduced by a factor of two. When the number of threads is further

	All MPI	2 threads	3 threads	6 threads	12 threads
Time (sec)	180	167	170	181	208

Table 1. Performance comparison of the kinetic code with different numbers of threads per MPI task.

increased, the code slows down because synchronization between the threads requires more time. The case with 12 threads per MPI task gave the worst performance. This happened because of the Non-Uniform Memory Access (NUMA) design of the Kraken nodes. Each processor has its own local memory. Although processor can access the other processor memory, the memory bandwidth in this case is 20% smaller than the access time to local memory. For this reason, we observed the performance decrease by 15%.

To perform scaling tests, we used the BlueWaters supercomputer recently installed at the National Center for Supercomputing Applications (NCSA), located at the University of Illinois at Urbana-Champaign. This is Cray a XE6/XK7 machine. The XE6 nodes have two AMD Interlagos processors, whereas XK7 nodes contain one AMD processor and the NVIDIA GK110 "Kepler" accelerator. We used the XE6 nodes for the tests. Each Interlagos processor has 16 integer cores. There are 8 floating point units (FPU) per processors, e.g. a FPU is shared between two integer cores. Eight integer cores form a NUMA domain. To perform our runs, we used 1 MPI task per NUMA domain to keep shared data in one domain. Totally, we had 4 MPI tasks per node and each one spawned additional 8 threads. For the strong scaling tests we used 12 billion particles and the physical run time was 800 years. The strong scaling results are almost perfect (see Figure 1) and there is even a superscaling effect when we switch from 20k to 40k cores. This is explained by a better cache fitting speedup that overpasses the extra communication load. Figure 2 shows the weak scaling results when there is approximately an even load of 100,000 particles per core in each test. The simulation time is equal in all cases, but for some small insignificant fluctuations.

Number of cores	Time (sec)	Speed up	Ideal
20,000	1003		
40,000	484	2.07	2
80,000	251	1.93	2
96,000	209	1.20	1.2
120,000	167	1.25	1.25

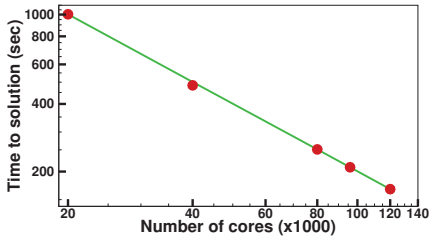

Figure 1. Strong scaling results of the kinetic code. The green line shows ideal performance. The red circles are measured time.

The IO performance of the code is also surprisingly good. A 650Gb data file containing 10 billion particles can be written as fast as 32 seconds on Lustre file system if it is striped over 100 Object Storage Targets (OSTs).

We also explored the opportunities to speed up the plasma code using available cores. Since number of grid blocks (patches) is less than the number of available cores, we needed to focus on how to speed up calculations on a single patch. We tried two different approaches: (a) auto-parallelization of low-level loops that are done by com-

Number of cores	Time (sec)
20,000	164
40,000	159
80,000	168
96,000	177
120,000	167

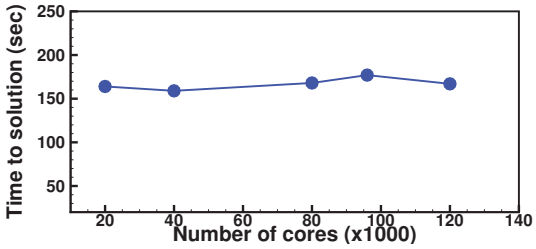

Figure 2. Weak scaling results of the kinetic code.

piler and (b) dimension splitting parallelization when reconstruction and solving of the Riemann problem is done in parallel by creating one thread for each dimension. Both approaches gave approximately the same speedup, close to a factor of two. There are some methods to speed up the code even more, e.g. nested parallelism, which will be explored in the future.

4. Simulation results

A new version of the MHD-kinetic code was used to perform the calculations of the SW-LISM problem. We used the following SW and LISM parameters, which were prposed by McComas et al. (2012). At 1 AU the SW number density $n_{\rm E} = 7.4\,{\rm cm}^{-3}$, the radial velocity $u_{\rm E} = 450\,{\rm km\,s}^{-1}$, the temperature $T_{\rm E} = 51100\,{\rm K}$, and the radial component of the magnetic field is $B_{\rm r} = 37.5\,\mu{\rm G}$. The LISM proton number density, velocity, temperature and magnetic field strength are $n_\infty = 0.13\,{\rm cm}^{-3}$, $u_\infty = 23.2\,{\rm km\,s}^{-1}$, $T_\infty = 6200\,{\rm K}$, $B_\infty = 2\,\mu{\rm G}$, respectively. The neutral H density is $n_{{\rm H}\infty} = 0.22\,{\rm cm}^{-3}$. The velocities and temperatures of hydrogen atoms and ions in the LISM coincide in thermodynamic equilibrium. The LISM vector $\mathbf{V}_\infty$ is aligned with the neutral He velocity vector, which is $(\lambda,\beta)_{\rm He} = (254^\circ, 5^\circ)$ in the Heliocentric Aries Ecliptic ($\rm HAE_{J2000}$) coordinates (Lallement et al. 2010). The interstellar magnetic field (ISMF) direction was chosen to fit the ribbon observed by the IBEX (Heerikhuisen et al. 2013). The normalized ISMF vector points toward $(\lambda,\beta) = (221^\circ, 39^\circ)$.

The calculations are performed on quadrilateral Cartesian meshes in the computational region $[-1000, -850, -850] \times [680, 830, 830]$ AU in the x-, y-, and z-directions, respectively. The base grid is 168^3 cells. Mesh refinement for plasma is performed by introducing three additional grid levels. The grid size at each upper level being half of the next level down. These meshes are nested as they approach the Sun. The results of the simulation are shown in Figure 3. These results correctly reproduce the major features of the SW-LISM interaction (the termination shock, the heliopause, the hydrogen wall, the weakened bow shock) and are exactly the same as obtained by the previous version of the code. This validates the correctness of the new code (Heerikhuisen et al. 2013).

5. Conclusion

The new implementation of the MHD-kinetic simulations in MS-FLUKSS is a significant step forward in our capability to model the SW-LISM interaction. It significantly

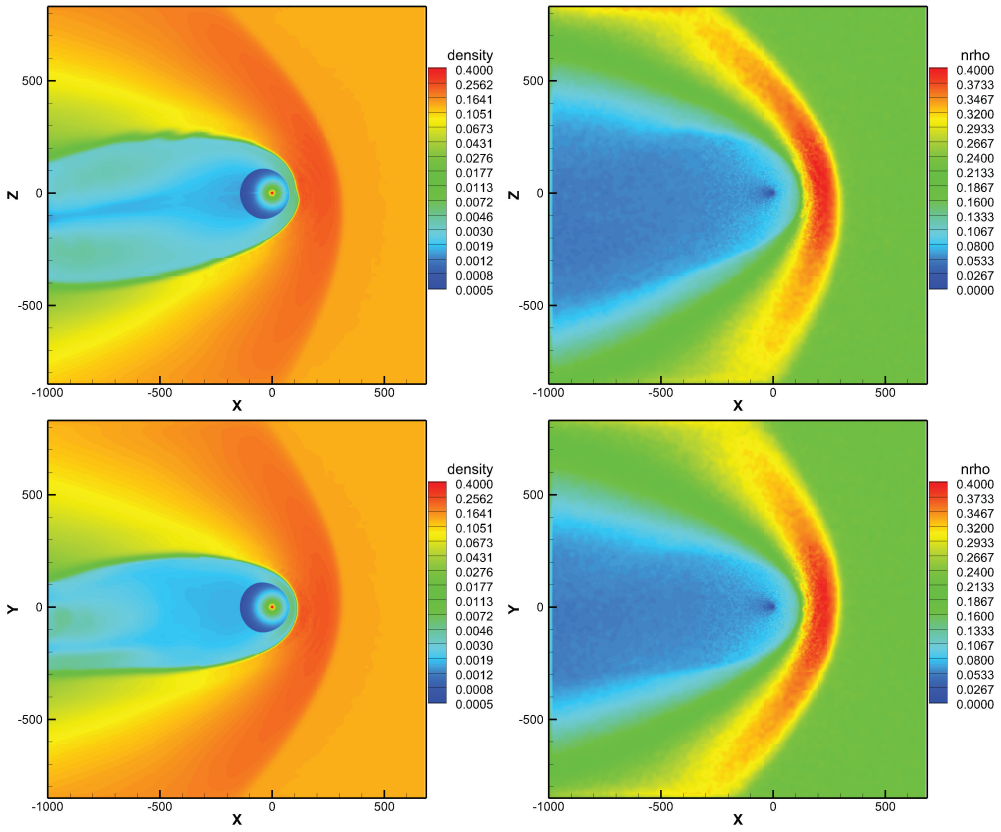

Figure 3. Simulation results. Top row: plasma density and neutral density distributions in the meridional plane. Bottom row: plasma density and neutral density distributions in the equatorial plane.

reduces the time necessary to perform the simulations. Even more interesting is the capability it gives us to tackle previously intractable problems. Long tail simulation results will be used in the future to analyze Lyman-α absorption in the directions of nearby stars.

Acknowledgments. The work presented here was supported in part by NASA grants[1], and DOE grant DE-SC0008334. Supercomputer time allocations were provided on SGI Pleiades by NASA High-End Computing Program award SMD-12-3071, Cray XT5 Kraken by NSF XSEDE project MCA07S033. This research is part of the Blue Waters sustained-petascale computing project, which is supported by the NSF (award number OCI 07-25070) and the state of Illinois. This work is also part of the "Modeling Heliophysics and Astrophysics Phenomena with a Multi-Scale Fluid-Kinetic Simulation Suite" PRAC allocation support by the NSF (award number 1144120).

[1]NNH09AM47I, NNX09AP74A, NNX10AE46G, NNX12AB30G

References

Baranov, V. B., & Malama, Y. G. 1993, J. Geophys. Res., 98, 15157
Borovikov, S. N., Heerikhuisen, J., Pogorelov, N. V., Kryukov, I. A., & Zank, G. P. 2008, in Numerical Modeling of Space Plasma Flows, edited by N. V. Pogorelov, E. Audit, & G. P. Zank, vol. 385 of Astronomical Society of the Pacific Conference Series, 197
Heerikhuisen, J., Florinski, V., & Zank, G. P. 2006, J. Geophys. Res., 111, A06110
Heerikhuisen, J., & Pogorelov, N. V. 2011, ApJ, 738, 29
Heerikhuisen, J., Zirnstein, E., Pogorelov, N. V., & Zank, G. P. 2013, ApJ, in preparation
Lallement, R., Quémerais, E., Koutroumpa, D., Bertaux, J., Ferron, S., Schmidt, W., & Lamy, P. 2010, Twelfth International Solar Wind Conference, 1216, 555
McComas, D. J., Alexashov, D., Bzowski, M., Fahr, H., Heerikhuisen, J., Izmodenov, V., Lee, M. A., Möbius, E., Pogorelov, N., Schwadron, N. A., & Zank, G. P. 2012, Science, 336, 1291
Pogorelov, N. V., Heerikhuisen, J., Zank, G. P., & Borovikov, S. N. 2009, Space Sci.Rev., 143, 31
Zank, G. P., Pauls, H. L., Williams, L. L., & Hall, D. T. 1996, J. Geophys. Res., 101, 21639