Performance measurement of Eulerian kinetic code on the Xeon Phi KNL

Takayuki Umeda
Institute for Space-Earth Environmental Research,
Nagoya University
Nagoya, Aichi
umeda@isee.nagoya-u.ac.jp

Keiichiro Fukazawa
Academic Center for Computing and Media Studies,
Kyoto University
Kyoto, Kyoto
fukazawa@media.kyoto-u.ac.jp

ABSTRACT

The present study deals with the Eulerian kinetic simulation code as a high-performance application, which solves the first-principle kinetic equations known as the Boltzmann equation. A five-dimensional Boltzmann code with two spatial dimension and three velocity dimensions is parallelized with the MPI-OpenMP hybrid parallelism. The performance of the parallel Boltzmann code is measured on a single compute node with a Xeon Phi Knights Landing (KNL) processor. In the present performance measurement with different configurations of memory and cluster modes, the number of processes per node is varied. The result shows that the MPI-OpenMP hybrid parallelism outperforms the flat-MPI for any number of processes per node and that 4 or 16 processes are an optimum number of processes per node. It is also shown that the use of Multi-Channel Dynamic Random Access Memory (MCDRAM) as the "cache" mode gives higher performances than the "flat" mode.

CCS CONCEPTS

• **Computing methodologies** → **Massively parallel and high-performance simulations**; *Parallel algorithms*; • **Applied computing** → Physics; Astronomy; Earth and atmospheric sciences;

KEYWORDS

Xeon Phi; High performance computing; kinetic simulation; Eulerian-grid-based method; hybrid parallelism; performance measurement

ACM Reference Format:
Takayuki Umeda and Keiichiro Fukazawa. 2018. Performance measurement of Eulerian kinetic code on the Xeon Phi KNL. In *Proceedings of International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2018).* ACM, New York, NY, USA, 4 pages.

HPC Asia 2018, January 2018, Tokyo, Japan

1 INTRODUCTION

Manycore scalar processors are one of recent trends of CPUs in high-performance computing, which run at low clock frequency to reduce the power consumption but have a large number of compute cores with processing units for operating multiple data, such as Advanced Vector Extension (AVX) and Single Instruction Multi Data (SIMD) units. It is not easy for users of scientific applications to achieve a high performance (e.g., a computational efficiency of more than 30% to the theoretical peak performance) on recent manycore scalar processors with multiple data units.

As a high-performance application to scientific computing, the present study deals with a first-principle kinetic simulation based on the Eulerian grid. The first-principle kinetic simulation usually requires enormous computing resources since this solves time development of distribution functions defined in "hyper" dimensions (at most six dimensions: three spatial and three velocity dimensions). In Eulerian-grid-based simulations, such as fluid simulations and the present kinetic simulations, a bottleneck of the computational performance generally exists at the memory bandwidth.

Performance tuning of the Eulerian-grid-based codes on manycore scalar processors is an issue in high-performance computing. In the present study, performance measurement of the Eulerian-grid-based kinetic code is made on the recent processor Xeon Phi Knight Landing (KNL) with various configurations of memory and cluster modes.

2 OVERVIEW OF NUMERICAL SCHEMES

The present kinetic code solves the first-principle equation, which is known as the collisionless Boltzmann equation,

$$\frac{\partial f_s}{\partial t} + \mathbf{v}\cdot\frac{\partial f_s}{\partial \mathbf{x}} + \left[\frac{q_s}{m_s}\left(\mathbf{E}+\mathbf{v}\times\mathbf{B}\right)+\mathbf{g}\right]\cdot\frac{\partial f_s}{\partial \mathbf{v}} = 0, \qquad (1)$$

where f represents the distribution function at a given position $\mathbf{x}$, velocity $\mathbf{v}$, and time t, and $\mathbf{E}$, $\mathbf{B}$, $\mathbf{g}$, q, and m represent electric field, magnetic field, gravity, charge, and mass, respectively. The subscript s represents the species of singly charged particles (e.g., $s = i$ and e for ions and electrons, respectively). Here, the collisional term in the right hand side of the equation is set to be zero. The self-consistent electromagnetic and gravitational fields are obtained by coupling field equations such as the Maxwell equations and the gravitational field formula. This equation is also known as the

Vlasov equation. By taking moments of the Boltzmann equation (i.e., integrating over the velocity $\mathbf{v}$), fluid equations are obtained.

It is not easy to integrate the "hyper-"dimensional equation numerically in time in terms of both computational accuracy and computational resources. In order to simplify the numerical operation, the Boltzmann equation (1) is separated into two advection equations (2,3) and a rotation equation (4) based on an operator splitting method [1, 2].

$$\frac{\partial f_s}{\partial t} + \mathbf{v} \cdot \frac{\partial f_s}{\partial \mathbf{x}} = 0, \tag{2}$$

$$\frac{\partial f_s}{\partial t} + \left[\frac{q_s}{m_s} \mathbf{E} + \mathbf{g} \right] \cdot \frac{\partial f_s}{\partial \mathbf{v}} = 0, \tag{3}$$

$$\frac{\partial f_s}{\partial t} + \frac{q_s}{m_s} (\mathbf{v} \times \mathbf{B}) \cdot \frac{\partial f_s}{\partial \mathbf{v}} = 0. \tag{4}$$

Equations (2) and (3) are scalar (linear) advection equations in which $\mathbf{v}$ and $q_s\mathbf{E}/m_s + \mathbf{g}$ are independent of $\mathbf{x}$ and $\mathbf{v}$, respectively. For solving these multi-dimensional advection equations, we adopt a multidimensional conservative semi-Lagrangian scheme [2] together with a fifth-order positive and non-oscillatory limiter [3, 4].

Equation (4) is a multi-dimensional rotation equation which follows a circular motion of a profile at constant speed by a centripetal force. For stable (solid body) rotation of the profile on the Cartesian grid system, the "back-substitution" scheme [5] together with the positive and non-oscillatory scheme [3, 4] are adopted.

The distribution function is defined by a five-dimensional array $f_s(v_x, v_y, v_z, x, y)$. The velocity dimensions are placed at inner loops because the density and the momentum are computed by the integration (sum) of the distribution function over the velocity directions. For solving the advection equation (2), however, the array of the distribution function is transposed to $f'_s(x, y, v_x, v_y, v_z)$ for accelerating the computation.

A hyper-dimensional simulation requires a huge computer resource. For an example, a computation with 40^6 grid cells requires ∼160 GB memory, so a limited number of systems with large shared memory can handle it. In the present five-dimensional Boltzmann code, a computation with 40^5 grid cells requires ∼4 GB memory, which can be handled on various supercomputer systems with a small shared memory. For applications to practical scientific computing, however, massively parallel computation with multiple compute nodes is necessary, since more than 1 TB memory is usually used for practical scientific computing. Hyper-dimensional Boltzmann simulations are practically in use (but not so widely) in plasma sciences, especially for laser plasma [1], tokamak plasma in thermonuclear fusion devices [6], and collisionless space plasma [7, 8].

We adopt the "domain decomposition" with the standard message passing interface (MPI) library as the first-level process parallelism as standard Eulerian-grid-based methods do. However, we decompose the computational domain only in the position dimensions [9]. The velocity dimensions are not decomposed because there arise some additional communications overhead due to a reduction operation in the computation of the density and momentum. It is well-known that the domain decomposition involves the exchange of halo layers for the distribution function and electromagnetic field data at boundaries of each computational sub-domain. The present non-oscillatory and conservative scheme [3, 4] uses six grid cells for numerical interpolation. Thus, three halo grids are exchanged by using the "`MPI_Sendrecv`" subroutine in MPI for simplicity, portability and stability. As a second-level thread parallelism, we use the "`OMP (PARALLEL) DO`" directive together with the "`COLLAPSE`" clause to parallelize most outer multiple loops with less threading overhead [10].

Table 1: Configurations of the Xeon Phi processor for the present performance measurement.

Configuration	Memory mode	Cluster mode
#1	Flat	All2All
#2	Cache	All2All
#3	Cache	Hemisphere
#4	Cache	Quadrant

3 PERFORMANCE MEASUREMENT

3.1 System descriptions

Our system has 9 6GB of DDR4 shared memory and a single Xeon Phi 7250 (Knights Landing) processor on one compute node. The processor has 16 GB of Multi-Channel Dynamic Random Access Memory (MCDRAM) and 68 compute cores. A total of 272 processes are executable with the hyper threading (HT) technology. The Intel Parallel Studio XE Cluster Edition Ver.17.0.1.132 is installed. The compiler option used in the present performance measurement is "`-ipo -ip -O3 -xMIC-AVX512`."

The MCDRAM has a high bandwidth of over 400 GB/s which is about five times larger than the bandwidth of DDR4 main memory (∼90 GB/s). However, the latency MCDRAM is slightly higher than that of the DDR4 main memory at low loads (but is much less at high loads). In the present study, we change the configuration of the Xeon Phi processor as shown in Table 1. The memory mode is chosen to be the "Flat" or "Cache" mode with the "All to All" cluster mode. Here, the Flat mode uses both of the MCDRAM and DDR4 main memory as shared memory, and the Cache mode uses the MCDRAM as a kind of (level three) cache of the main memory. The cluster mode is chosen to be the "All to All," "Hemisphere," or "Quadrant" mode with the "Cache" memory mode. The All to All cluster mode distributes memory addresses uniformly on the chip. The Hemisphere and Quadrant modes separate compute-core tiles into two and four groups, respectively, and memory addresses are distributed only to each tile groups.

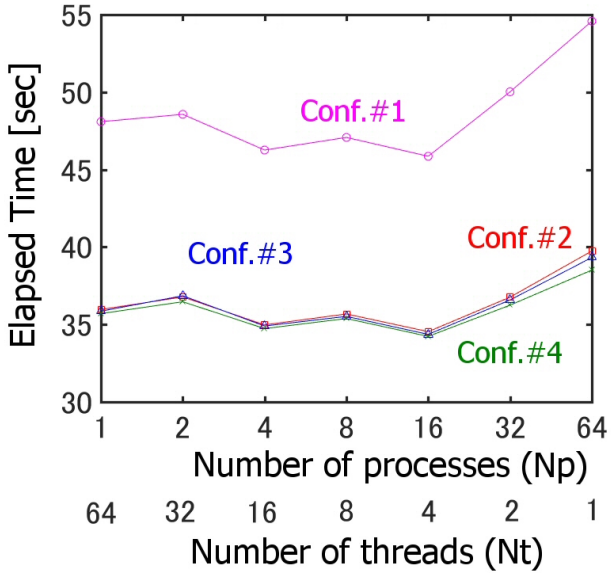

Figure 1: Performance measurement result of the five-dimensional Eulerian kinetic code on the KNL Xeon Phi processor with 64 compute cores. The vertical axis shows the elapsed time for five time steps. The horizontal axis shows the number of processes N_p (or the number of threads given by $N_t = 64/N_p$). The circles, squares, triangles, and x-marks correspond to the results with configurations #1, #2, #3, and #4 in Table 1, respectively.

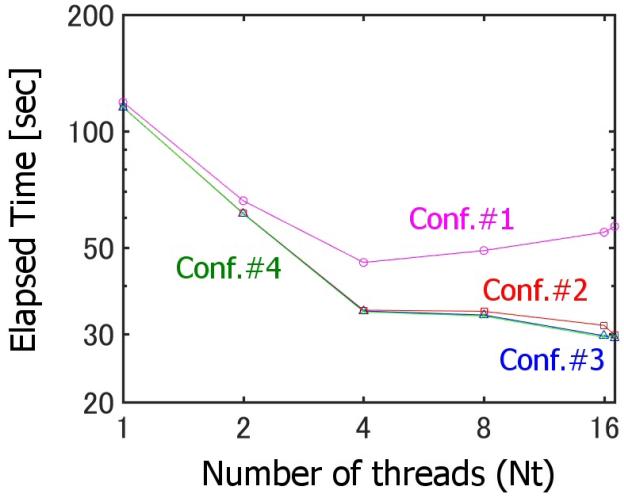

Figure 2: Strong scaling of the five-dimensional Eulerian kinetic code on the KNL Xeon Phi processor with 16 processes per compute node. The vertical axis shows the elapsed time for five time steps. The horizontal axis shows the number of threadsN_t. The circles, squares, triangles, and x-marks correspond to the results with configurations #1, #2, #3, and #4 in Table 1, respectively.

3.2 Results

For overall performance measurement, the total number of grid cells was fixed to $N_x \times N_y \times N_{vx} \times N_{vy} \times N_{vz} = 128 \times 64 \times 40 \times 40 \times 40$ for two particle species ($s = i$ and e for positively-charged ions and electrons, respectively), which corresponds to a job size of ∼28 GB including temporary work arrays. Hence, the size of the job is larger than the memory size of the MCDRAM. It should be noted that the length of a loop (number of iterations) for one dimension is small in hyper-dimensional simulations due to the limited size of shared memory. Then, the number of threads sometimes exceed the length of the loop, which causes a performance loss of the thread parallelism. Hence, the loop collapsing is necessary in hyper-dimensional simulations.

We first measured the performance by changing the number of processes per compute node. The total number of cores used in this performance measurement is fixed to 64 compute cores. Figure 1 shows the performance measurement result for the five-dimensional Eulerian kinetic code. The vertical axis shows the elapsed time for five time steps. The horizontal axis shows the number of processes N_p (or the number of threads given by $N_t = 64/N_p$). The circles, squares, triangles, and x-marks correspond to the results with configurations #1, #2, #3, and #4 in Table 1, respectively.

It is shown that the elapsed time for the flat-MPI (i.e., 64 processes and 1 thread per compute node) is longest for all of the configurations. The hybrid parallelism with 16 processes (4 threads) per compute node is fastest and the hybrid parallelism with 4 processes (16 threads) per compute node is second-fastest for all of the configurations.

It is also clearly shown that the performance of the "Flat" memory mode is worse. Although the "Quadrant" cluster mode with 16 processes (4 threads) per compute node is fastest in Fig.1, the difference of performance among "All to All," "Hemisphere," and "Quadrant" cluster modes is ∼ 1%.

We next conducted strong-scaling tests with different number of threads by changing the "`OMP_NUM_THREADS`" environment variables. Figure 2 shows the strong scaling of the five-dimensional Eulerian kinetic code with 16 processes per compute node. The vertical axis shows the elapsed time for five time steps. The horizontal axis shows the number of threads N_t. The circles, squares, triangles, and x-marks correspond to the results with configurations #1, #2, #3, and #4 in Table 1, respectively.

With the "Flat" memory mode, the performance does not scale linearly with a larger number of threads. The performance becomes worse with number of threads larger than 4, suggesting that HT is not effective with the configuration #1.

With the "Cache" memory mode, the performance scales linearly until 4 threads per compute node. The performance becomes slightly better with a larger number of threads. The

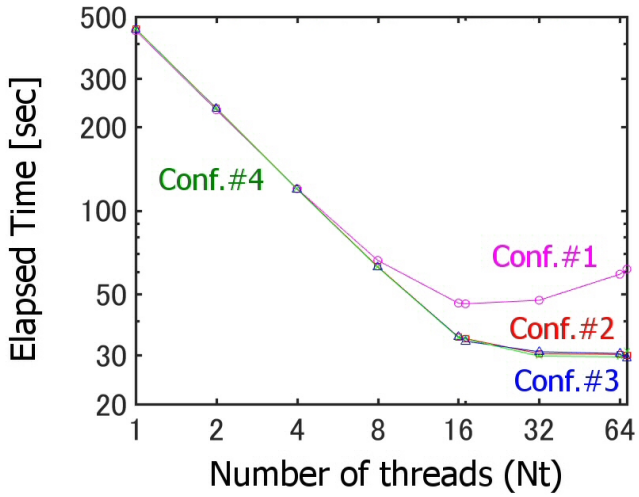

Figure 3: Strong scaling of the five-dimensional Eulerian kinetic code on the KNL Xeon Phi processor with 4 processes per compute node with the same format as Fig.2.

performance is improved by $\sim$ 15% from 4 threads to 17 threads with HT. Although the "Hemisphere" cluster mode with 17 threads per compute node is fastest in Fig.2, the difference of performance among "All to All," "Hemisphere," and "Quadrant" cluster modes is $\sim$ 1%.

Figure 3 shows the strong scaling of the five-dimensional Eulerian kinetic code with 4 processes per compute node with the same format as Fig.2. The result with 4 processes shows the same tendency as the result with 16 processes. With the "Flat" memory mode, the performance becomes worse with number of threads larger than 17, suggesting that HT is not effective with the configuration #1.

With the "Cache" memory mode, the performance scales linearly until 17 threads per compute node. The performance becomes slightly better with a larger number of threads. The performance is improved by $\sim$ 13% from 17 threads to 68 threads with HT. Although the "Hemisphere" cluster mode with 68 threads per compute node is fastest in Fig.3, the difference of performance among "All to All," "Hemisphere," and "Quadrant" cluster modes is $\sim$ 1%.

The best performance is obtained with 4 processes and 68 threads per compute nodes on the configuration #3 (Cache and Hemisphere modes). The performance on the KNL Xeon Phi is slightly higher than on the Haswell Xeon processor by $\sim$ 13%.

4 CONCLUSIONS

In the present study, we have made performance measurement of the five-dimensional Eulerian kinetic code on the KNL Xeon Phi processor. It is shown that the hybrid parallelism with the loop collapsing of OpenMP outperforms the flat-MPI parallelism, as seen in other scalar processors.

The performance of the MCDRAM with "Cache" mode is higher than with "Flat" mode if the size of job is larger than the memory size of the MCDRAM. It is suggested that HT is not effective with the "Flat" memory mode of MCDRAM. The effect of the cluster mode to the performance is small, although the "All to All" mode is slightly slower than the "Hemisphere" and "Quadrant" modes.

ACKNOWLEDGMENTS

This work was supported by MEXT/JSPS Grant-In-Aid (KAKENHI) Nos.JP26287041 and JP15K13572.

REFERENCES

[1] A. Ghizzo, F. Huot, and P. Bertrand, A non-periodic 2D semi-Lagrangian Vlasov code for Laser-plasma interaction on parallel computer, *J. Comput. Phys.*, 186(1):47–69, March 2003.

[2] T. Umeda, K. Togano, and T. Ogino, Two-dimensional full-electromagnetic Vlasov code with conservative scheme and its application to magnetic reconnection, *Comput. Phys. Commun.*, 180(3):365–374, March 2009.

[3] T. Umeda, A conservative and non-oscillatory scheme for Vlasov code simulations, *Earth Planets Space*, 60(7):773–779, August 2008.

[4] T. Umeda, Y. Nariyuki, and D. Kariya, A non-oscillatory and conservative semi-Lagrangian scheme with fourth-degree polynomial interpolation for solving the Vlasov equation, *Comput. Phys. Commun.*, 183(5):1094–1100, May 2012.

[5] H. Schmitz and R. Grauer, Comparison of time splitting and backsubstitution methods for integrating Vlasov's equation with magnetic fields, *Comput. Phys. Commun.*, 175(2):86–92, March 2006.

[6] Y. Idomura, M. Ida, T. Kano, N. Aiba, and S. Tokuda, Conservative global gyrokinetic toroidal full-f five-dimensional Vlasov simulation, *Comput. Phys. Commun.*, 179(6):391–403, September 2008.

[7] S. Von Alfthan, D. Pokhotelov, Y. Kempf, S. Hoilijoki, I. Honkonen, A. Sandroos, and M. Palmroth, Vlasiator: First global hybrid-Vlasov simulations of Earth's foreshock and magnetosheath, *J. Atmos. Sol.-Terr. Phys.*, 120: 24-35, December 2014.

[8] T. Umeda and K. Fukazawa, A high-resolution global Vlasov simulation of a small dielectric body with a weak intrinsic magnetic field on the K computer, *Earth Planets Space*, 67(1):49, April 2015.

[9] T. Umeda, K. Fukazawa, Y. Nariyuki, and T. Ogino, A scalable full electromagnetic Vlasov solver for cross-scale coupling in space plasma, *IEEE Trans. Plasma Sci.*, 40(5):1421–1428, March 2012.

[10] T. Umeda and K. Fukazawa, Hybrid parallelization of hyper-dimensional Vlasov code with OpenMP loop collapse directive, *Adv. Parallel Comput.*, 27:265–274, April 2016.