Sep. 8, 2017

MHDシミュレーションコードの Xeon Phi KNLでの性能評価

Keiichiro FUKAZAWA[1], Takayuki UMEDA[2], Takeshi NANRI[3]

1. Academic Center for Computing and Media Studies, Kyoto University, Japan
2. Institute for Space-Earth Environmental Research at Nagoya University, Japan
3. Research Institute for Information Technology, Kyushu University, Japan

Introduction

What is the planetary magnetosphere?

No less than 99.9 % of the matter in the visible universe is in the plasma state which is solar/stellar plasma.

The planetary magnetosphere is formed by the interaction between the solar wind which is the plasma from the Sun and the planetary magnetic field.

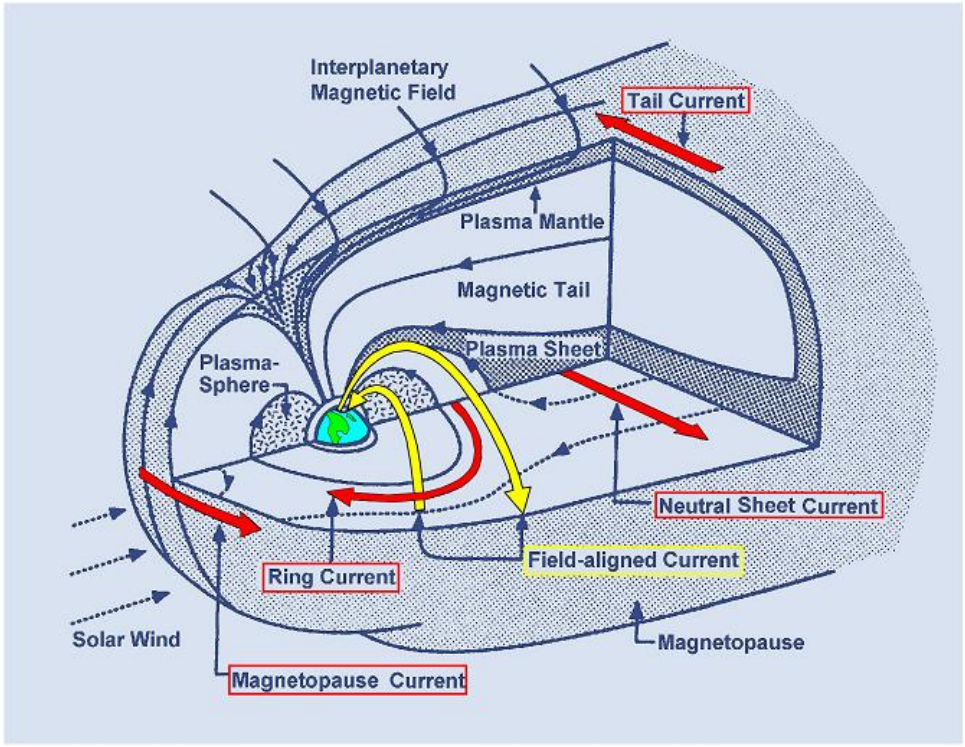


Fig.1. A sketch of the magnetosphere (modified from Kivelson and Russel [1995])

The phenomena in this region is called as “space weather”

Introduction

Forecast of space weather

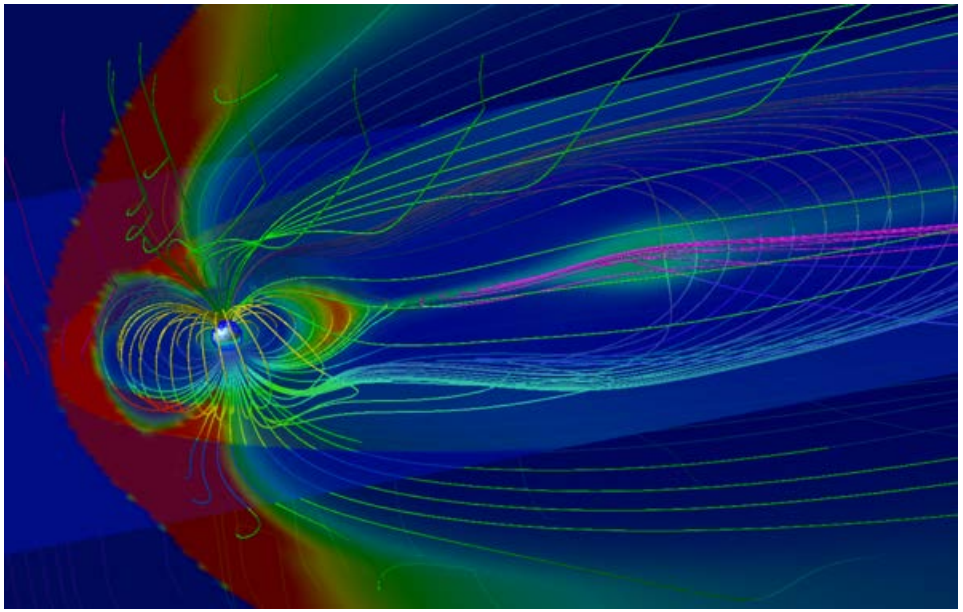


Fig. 2. Simulation result of Terrestrial magnetosphere

To forecast the space weather, we perform the simulation of magnetosphere with the global MagnetoHydroDynamic (MHD) simulation code.

From our estimation, at least 20 PFlops (effective performance not theoretical performance) is required to simulate the accurate global magnetosphere in the real time.

Motivation

Perform the MHD code for space weather effectively

There are various computer architectures used in the recent supercomputer systems, such as vector, X86, POWER and SPARC for CPUs, then GPU and MIC for accelerators/coprocessors.

It is hard for application developers to optimize their applications to these various computer architectures.

In general, the computing efficiency of user applications on a scalar-type computer tends to **be low (~5 %)** [*Oliker et al.*, 2004], although the computing efficiency of LINPACK sometimes exceeds 80 %.

Considering the execution efficiency, we **need more performance of Top 1 supercomputer now** (Sunway TaihuLight 125 PFlops in the theoretical performance) to forecast the space weather precisely.

Simulation Model | MHD equations

Vlasov equation (collisionless Boltzmann equation)

$$\frac{\partial f_s}{\partial t} + \vec{v}\frac{\partial f_s}{\partial \vec{r}} + \frac{q_s}{m_s}(\vec{E} + \vec{v} \times \vec{B})\frac{\partial f_s}{\partial \vec{v}} = 0$$

Maxwell equations

$$\begin{cases} \nabla \times \vec{B} = \mu_0 \vec{J} + \dfrac{1}{c^2}\dfrac{\partial \vec{E}}{\partial t} \\ \nabla \times \vec{E} = -\dfrac{\partial \vec{B}}{\partial t} \\ \nabla \cdot \vec{E} = \dfrac{\rho}{\varepsilon_0} \\ \nabla \cdot \vec{B} = 0 \end{cases}$$

MHD equations

$$\begin{cases} \dfrac{\partial \rho}{\partial t} = -\nabla \cdot (\vec{v}\rho) + D\nabla^2 \rho \\ \dfrac{\partial \vec{v}}{\partial t} = -(\vec{v} \cdot \nabla)\vec{v} - \dfrac{1}{\rho}\nabla P + \dfrac{1}{\rho}\vec{J} \times \vec{B} + g + \dfrac{\boldsymbol{\Phi}}{\rho} \\ \dfrac{\partial P}{\partial t} = -(\vec{v} \cdot \nabla)P - \gamma P \nabla \vec{v} + D_P \nabla^2 P \\ \dfrac{\partial \vec{B}}{\partial t} = \nabla \times (\vec{v} \times \vec{B}) + \eta \nabla^2 \vec{B} \end{cases}$$

$$* \ \vec{J} = \nabla \times (\vec{B} - \vec{B}_d)$$

Simulation Model | Numerical method

Numerical simulation code

- Our three-dimensional MHD code uses the “Modified Leap frog (MLF)” method [*Ogino et al.*, 1992].
- Using MLF method, partial difference equations are solved by the two-step Lax-Wendroff method for one time step and then by the Leap frog method for $(l-1)$ time steps and the procedure is repeated.
- MLF method is a kind of combination technique which balances numerical stability of the two step Lax-Wendroff method and dissipationlessness of the Leap frog method.

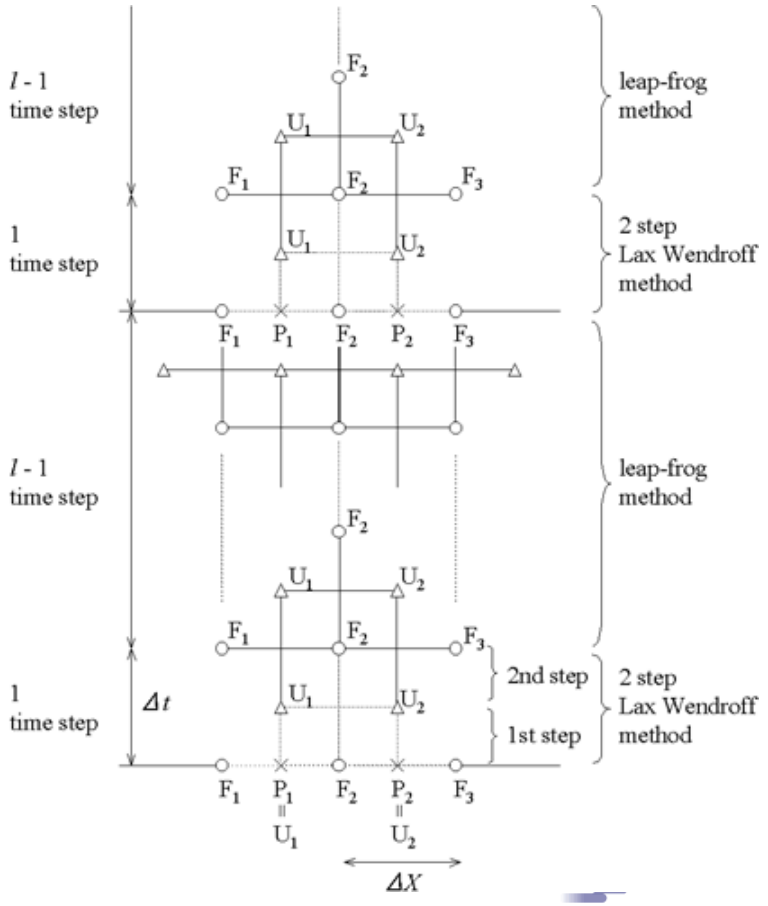


Fig. 3. Diagram of Modified Leap frog method

Simulation Model | Numerical method

Implementation

- The MLF method is a kind of central difference method using 8 grid points to update a value as Fig. 4.
- This method uses the staggered grid (half mesh) points to develop.
- The implementation of this development is like right code.

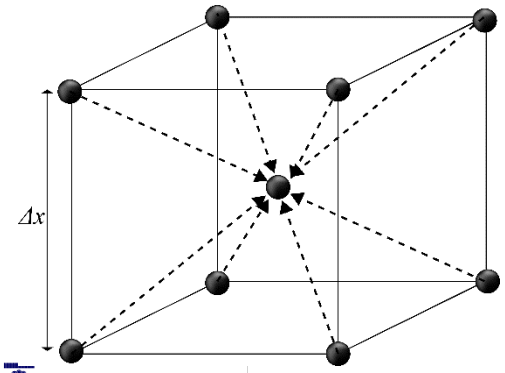


Fig. 4. Coordinate of MLF numerical method to update one value.

```
do k=1, nz
  do j=1, ny
    do i=1, nx
      u(i,j,k,8)=u(i,j,k,8)+dx*(                    &
               f(i+1,j+1,k+1,4)*f(i+1,j+1,k+1,6) &
             - f(i+1,j+1,k+1,2)*f(i+1,j+1,k+1,8) &
             + f(i+1,j,k+1,4)*f(i+1,j,k+1,6)     &
             - f(i+1,j,k+1,2)*f(i+1,j,k+1,8)     &
             + f(i+1,j+1,k,4)*f(i+1,j+1,k,6)     &
             - f(i+1,j+1,k,2)*f(i+1,j+1,k,8)     &
             + f(i+1,j,k,4)*f(i+1,j,k,6)         &
             - f(i+1,j,k,2)*f(i+1,j,k,8)         &
             - f(i,j+1,k+1,4)*f(i,j+1,k+1,6)     &
             + f(i,j+1,k+1,2)*f(i,j+1,k+1,8)     &
             - f(i,j,k+1,4)*f(i,j,k+1,6)         &
             + f(i,j,k+1,2)*f(i,j,k+1,8)         &
             - f(i,j+1,k,4)*f(i,j+1,k,6)         &
             + f(i,j+1,k,2)*f(i,j+1,k,8)         &
             - f(i,j,k,4)*f(i,j,k,6)             &
             + f(i,j,k,2)*f(i,j,k,8)  )
    end do
  end do
end do
```

Simulation Model | Parallel method

Domain decomposition

- To decompose the simulation, there are three way according to the simulation dimensions.

	Calculation time (T_S)	Communication time (T_C)
1D	$T_{S1} = k_1 n^3 / p$	$T_{C1} = k_2 n^2 (p-1)$
2D	$T_{S2} = k_1 n^3 / p$	$T_{C2} = 2k_2 n^2 (p^{\frac{1}{2}}-1)$
3D	$T_{S3} = k_1 n^3 / p$	$T_{C3} = 3k_2 n^2 (p^{\frac{1}{3}}-1)$

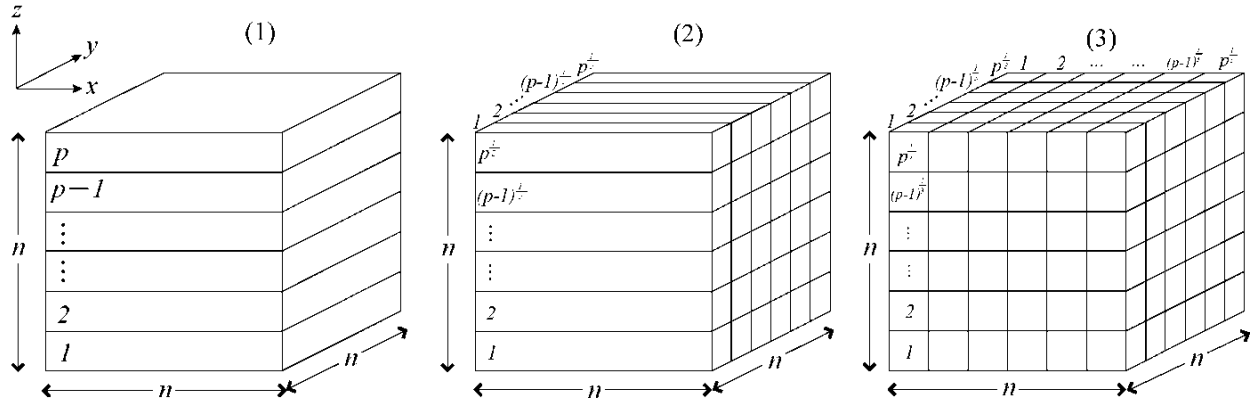


Fig. 5. Schematics of three kinds of the domain decomposition methods. (1) One-dimensional domain decomposition in z-direction. (2) Two-dimensional in y- and z-directions. (3) Three-dimensional domain decomposition [*Fukazawa et al*., 2010].

Simulation Model | Parallel method

Estimation of communication time

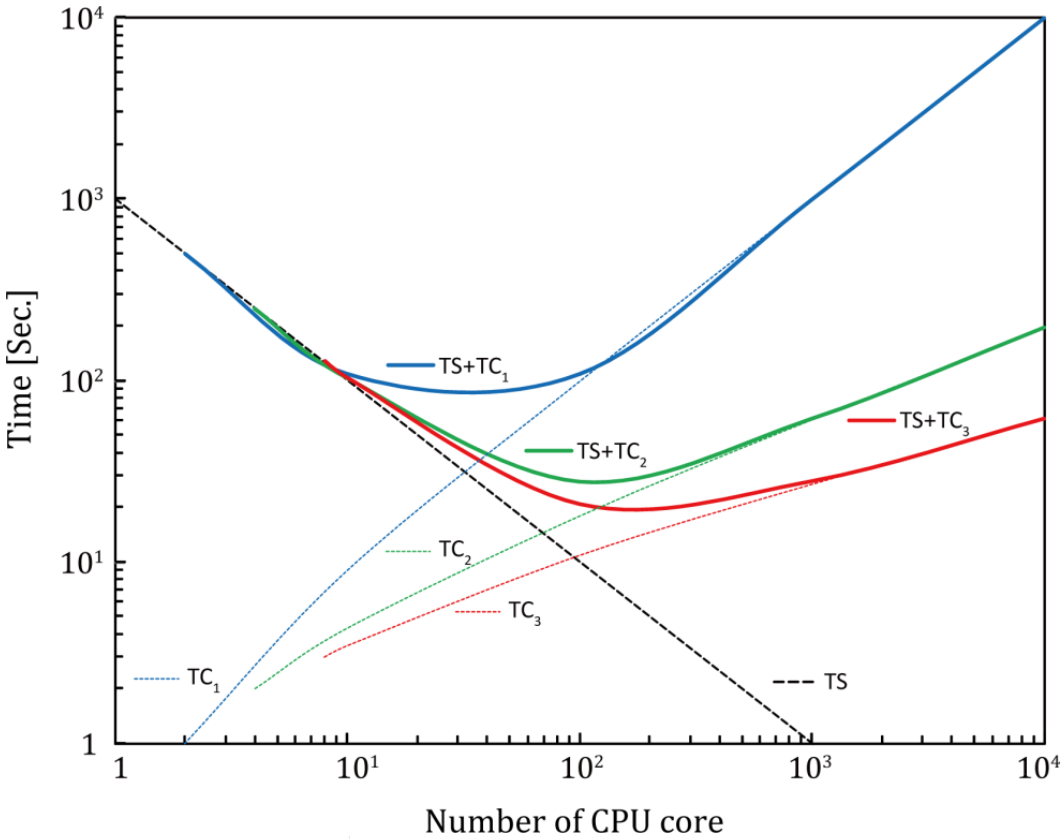


Fig. 6. Computation time (T_S), communication time (T_C), and total parallel computation time (T_S + T_C) as a function of the number of processor core. Here k_1 and k_2 are set to be 1 and 0.01, respectively, to simplify this figure [*Fukazawa et al.*, 2010].

The communication time (T_C) of the 3D domain decomposition is the shortest.

Thus the total time (T_S+T_C) is shortest in 3D domain decomposition.

*the coefficients k_1 and k_2 assumed to be independent of the way of decomposition.

Simulation Model | Setting

Parallelization

- 1D, 2D and 3D domain decompositions (DD)
- Flat MPI and hybrid MPI
- To minimize the communication time, we use a buffer array which stores all the boundary data for inter-core and inter-node communications (pack/unpack operation) except for 1D decomposition.

Variation of array order

- Normal array type : $f(i, j, k, m)$ (SoA)
- To consider the cache hit efficiency, we change the array order in the 3D DD as $f(m, i, j, k)$ (AoS)

  * We use the Fortran.

System size

- Size of array is 64 MB/core for the computational domain and additionally 192 MB/core for workspaces for computing the MHD equations (the weak scaling is used in this study).

Evaluation System

Xeon Phi Knights Landing (KNL)

- Cray XC40 @ Kyoto Univ.

Fig. 7. Cray XC40

Table 1. Characters of XC40

		XC40
CPU	Architecture	68 cores Xeon Phi KNL
	Frequency	1.4 GHz (3.05 TFlops)
	Cache	L1: 32 KB/core L2: 34 MB/CPU（1MB/Tile）
Memory	Band width	102.3 + 921 GB/s /node
B/F		0.03 or 0.30
Node	Number of CPUs	1
	Memory size	96 GB + 16GB
System	Number of nodes	1,800 (122,400 cores)
	Rmax	5.48 PFlops
	Node comm.	Dragonfly, Aries (12.5GB/s)

Evaluation Results 1

Flat MPI

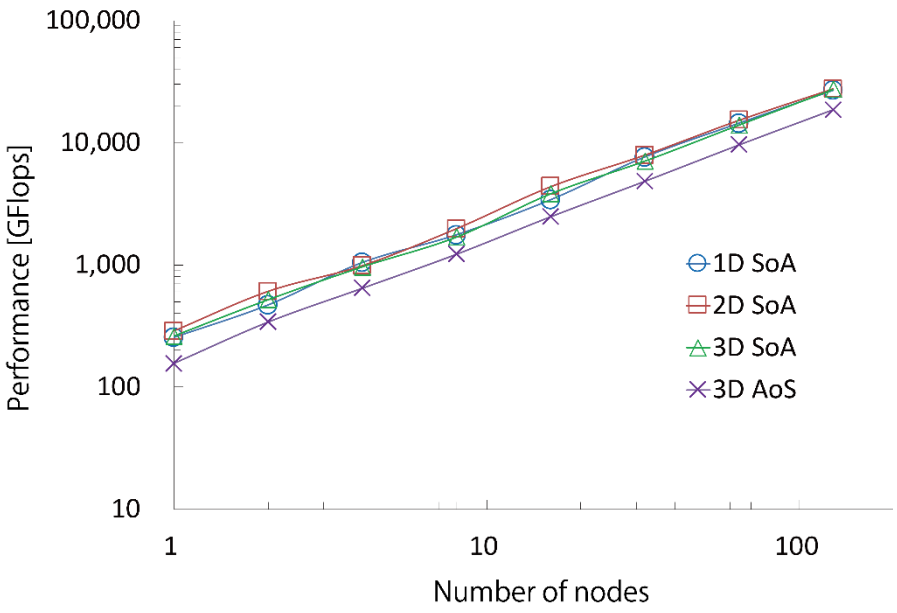


Fig. 8. Performance results of MHD simulation with Flat MPI on Cray XC40

The computing performance of AoS is not good compared to the SoA cases.

The SoA type be suitable for the Xeon Phi since the Xeon Phi has the high SIMD width due to the vector optimization.

The performances of SoA seems not so different.

The performances 1D, 2D, and 3D of SoA with 128 nodes are 26.6, 27.7 and 27.2 TFlops, respectively.

Evaluation Results 2

Hybrid MPI

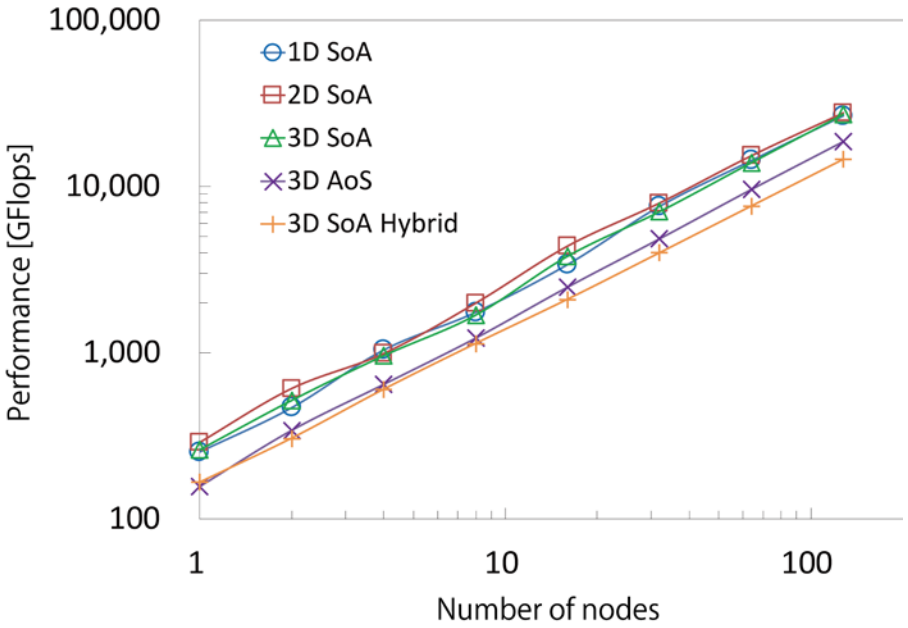


Fig. 9. Performance results of MHD simulation on Cray XC40

*The average communication time of hybrid MPI and flat MPI of 3D SoA are 3.5×10^{-2} sec and 1.5×10^{-2} sec.

The performance of hybrid parallel computation is clearly not good.

In this hybrid MPI, the MPI communications are performed on 8 cores per node.

The core performance of Xeon Phi KNL is not good as compared to the general Xeon core.

Then a load of communication per core becomes high in the hybrid MPI.

Evaluation Results 3

Hybrid MPI

To examine the effect of thread to the performance, the evaluations with several combinations of process and thread are performed.

The performance increases with the small number of thread.

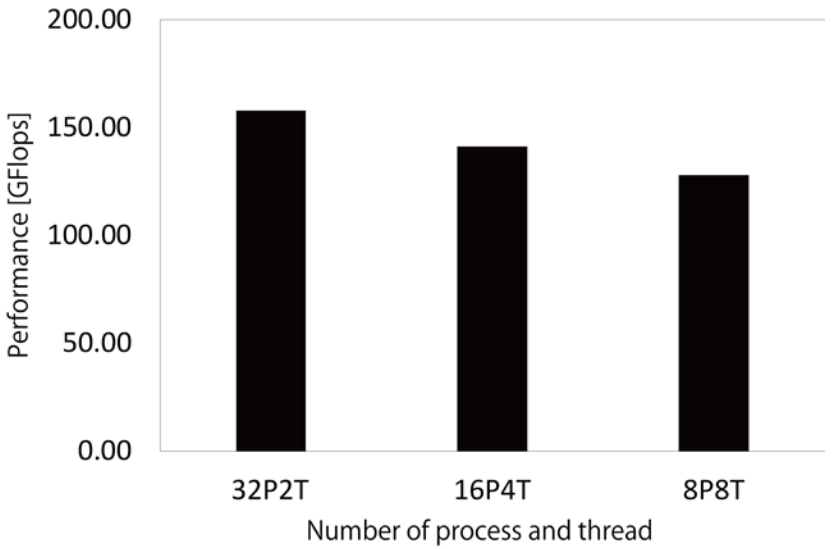


Fig. 10. Performance of Hybrid MPI with various combination of process and thread on one Xeon Phi KNL.

To avoid this performance degradation, it is thought the overlap of calculations and communications is effective.

Optimization 1

Need more serial performance of Xeon Phi

- From the evaluation results, the execution on Xeon Phi KNL seems to be low, thus the optimizations are added to our MHD simulation code.
- Considering the architecture of Xeon Phi KNL, it is important for the optimization to use the SIMD effectively and decrease the non-sequential memory access (increase the cache hit rate).
- So, the followings optimizations are performed to 3DD of SoA.

1. 64-byte alignment of array
2. Prefetch of Memory access
3. Arrangement of array length

Optimization 2

Serial tuning of Xeon Phi

- For the 64-bytes aligned load/store with Xeon Phi, we use the compile option “-align array64byte” and the following directive

```
real(kind=4) A(1024,100)
!DEC$ATTRIBUTES ALIGN: 64:: A
```

➡ This optimization of alignment increases the execution efficiency by 0.6 %.

- The prefetch can be controlled by the compile option “-qopt-prefetch=4”.

➡ This makes the 1.2 % efficiency increase.

*There are some compile options for control the prefetch however the other options are not effective to our code.

Optimization 3

Serial tuning of Xeon Phi

- The effect of array length to the calculation performance.
  Original array configuration **f(nx, ny, nz, m) = (100, 100, 100, 8)** / proc.

The configurations of array as **(200, 100, 50, 8), (200, 50, 100, 8)** and **(400, 50, 50, 8)** are evaluated to see the vector performance difference.
*The performance is good with the long length of nx.

(200, 50, 100, 8) performance ↘
(200, 100, 50, 8) and (400, 50, 50, 8) performance ↗
*0.6 % incensement of execution efficiency

Optimization 4

Optimization results

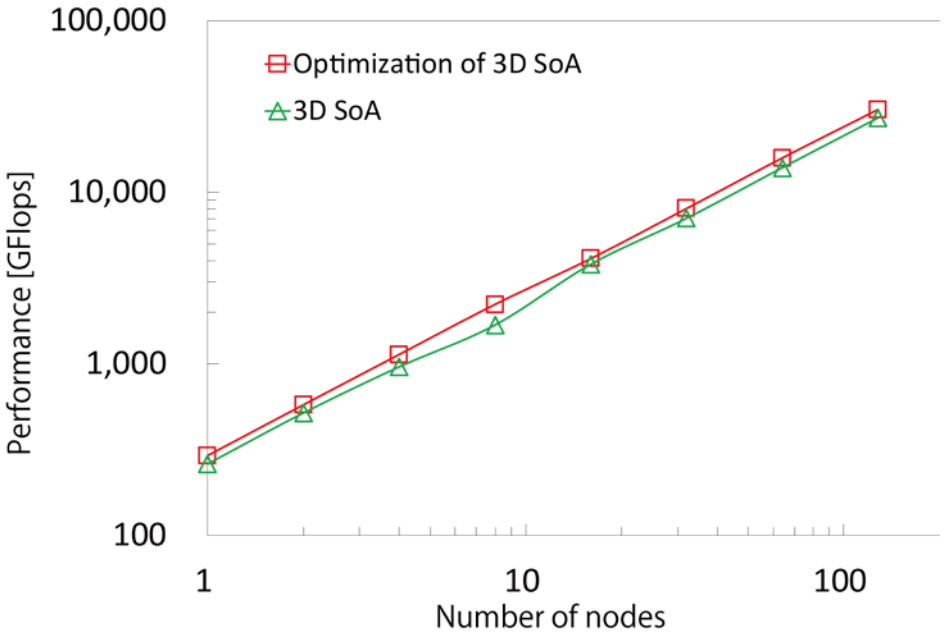


Fig. 11. Optimized performance results of MHD simulation code.

From these optimizations, we obtained the 2.4 % increases in execution efficiency in total.

The optimization code achieves 30 TFlops using 128 nodes and the non-optimization code is 27 TFlops.

Considering the simulation is performed for one year, this performance difference decrease the simulation time about one month.

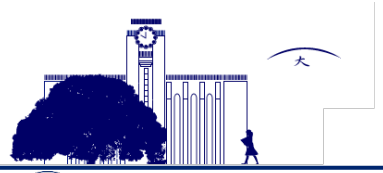
The results of this study are important for the numerical simulation area of fluid which requires the long time evolution.

Performance of MHD code

Performance comparison

Table 2. Performance evaluation of MHD simulation code on various computer systems.

	Core/CPU	Rmax [TFlops]	Rpeak [TFlops]	Rpeak /CPU [GFlops]	Efficiency [%]	Suitable domain decomposition	CPU architecture
SX-ACE	1024/256	65.50	29.20	114.0	45	3D SoA	Vector
K	262144/32768	4194.30	914.12	27.9	22	3D AoS	SPARC64 VIIIfx
FX100	16384/512	576.72	91.49	178.7	17	3D SoA	SPARC64 XIfx
CX400	23616/2952	510.11	104.23	35.3	20	3D SoA	Xeon (SB)
HA8000	23160/1930	500.26	83.42	43.2	17	2D SoA	Xeon (IB)
XC30	448/32	16.49	1.37	42.8	8	2D SoA	Xeon (HSW)
XC40	1088/16	48.86	4.32	273.3	9	3D SoA	KNL
Xeon Phi 5120	60/1	1.00	0.08	84.0	8	3D SoA	KNC
Tesla K20X	896/1	1.31	0.15	153.3	12	3D SoA	Kepler



Summary

Performance evaluation of MHD simulation code on Xeon Phi KNL

- ✓ To forecast the space weather accurately, it is necessary to perform the large magnetohydrodynamic (MHD) simulation of magnetosphere.
- ✓ In this study, we have evaluated the performance of our MHD simulation code with Xeon Phi KNL on Cray XC40 at Kyoto University.
- ✓ As the results of evaluation with flat MPI, the 2D and 3D domain decompositions with SoA are the effective calculation performances and 3D domain decomposition with AoS becomes lower performances
- ✓ Using the hybrid MPI, the performance becomes the worst due to a load of communication and related load/store operations.

Summary

Performance evaluation of MHD simulation code on Xeon Phi KNL

- ✓ The optimizations of alignment, prefetch and array configuration are added to our MHD simulation code and we have obtained 2.4 % increase of execution efficiency in total and 3 TFlops performance gain using 128 nodes of Cray XC40.
- ✓ Comparing the results of performance evaluation with other computer systems, Xeon Phi KNL achieves the triple performance of Xeon Phi KNC.
- ✓ To obtain more calculation performance, we need to overlap the calculation with communication and arrange the number of “add” and “multiple” in the implementation for “Fused multiple add”.