Vol.2015-HPC-151 No.23
2015/10/1

エクサスケールコンピューティングに向けたHaloスレッドの電磁流体シミュレーションに対する効果

深沢圭一郎[†1†4]　森江善之[†2†4]　曽我武史[†3†4]　高見利也[†2†4]　南里豪志[†2†4]

電磁流体（MHD）シミュレーションは，マクロスケールのプラズマシミュレーションに用いられており，ステンシル計算である．我々が惑星磁気圏研究に利用しているMHDシミュレーションは現在weakスケーリングであり，スケーラビリティも高いが，エクサスケール時の性能見積りでは，スケーラビリティが20%以上も劣化することが予想される．そこで，本研究では並列ステンシル計算において，通信が必要となる袖領域（Halo領域）での処理を専門に担当するスレッド（Haloスレッド）を導入し，計算性能，通信性能においてどのような効果があるかを調べた．1スレッドを通信スレッドのように計算スレッドから割り当てるために，計算性能が悪くなると想定されるが，計算サイズやスレッド数によって，Haloスレッドを導入した場合の全体計算性能が良くなることが分かった．この場合のエクサスケールにおけるスケーラビリティ見積においても3%程度の劣化で収まることが想定される．

Effects of Halo Thread to the Magnetohydrodynamic Simulation toward Exascale Computing

KEIICHIRO FUKAZAWA[†1†4]　YOSHIYUKI MORIE[†2†4]
TAKESHI SOGA[†3†4]　TOSHIYA TAKAMI[†2†4]　TAKESHI NANRI[†2†4]

Magnetohydrodynamic (MHD) simulation is stencil code and often used to study the macro scale plasma. We presently perform the weak scaling of MHD simulation and have obtained good scalability, however in the exascale era the scalability will decrease by over 20% from our estimation. In this study we introduce the Halo thread which covers the communication and calculation in the halo region to MHD simulation and examine the effects of Halo thread. It seems that the calculation performance will be worse due to the decrease of calculation thread using the Halo thread, but we obtain the good performance depending on the number of thread and size of grid in the MHD simulation.

1. Introduction

エクサスケールの計算機システムは世界中で計画されているが，2020年頃に計算機システムが完成すると考えられている．このシステムは300万ノード以上の構成が想像されている．現在，惑星磁気圏を解く電磁流体（MHD）シミュレーションコードでは最大で京コンピュータの3万ノード程度で性能評価を行っており，weak scalingでスケーラビリティが約10%劣化している．惑星磁気圏シミュレーションはその計算規模から，エクサスケールにおいてもweak scalingが続く問題であるが，このままエクサスケールに行くと，weak scalingでさえもスケーラビリティが落ちてしまい，並列化の効果が見えなくなることが想像される．これは基本的にMHDコード内で利用されるブロッキング通信がノード数に比例して時間がかかることが原因とされる．これに対してハードウェア側から，通信専用コアの導入や，ノード間通信性能向上といった開発が進められており，ミドルウェアの部分では新しい通信ライブラリやMPI自体の性能向上も議論されている．一方，アプリケーション側からは計算と通信をオーバーラップさせる手法が開発されており，通信にかかる時間をできる限り隠蔽しようとする努力がされている．例えば，*Sur*らはRDMAベースのオーバーラップ手法を提案しており[1]，核融合分野では特定のネットワークハードウェア上だが，PGASを用いた計算と通信のオーバーラップを*Preissl*らが提案している[2]. 最近では*Idomura*らがMPI_isend/irecvにおける通信progressを効率的に実行でき，通信終了後に通信スレッドも計算スレッドに参加する手法を提案している[3]．これらはそれぞれの実験の中では良い成果を出している．しかしながら，どの手法においても通信の終わりを知るために，どこかで同期を取る必要がある．

MHDシミュレーションは流体シミュレーションの1種であり，stencil計算である．stencil計算では並列化に伴いHalo領域と呼ばれる各プロセスにある袖領域をプロセス間で通信する必要が出てくる．そこで本研究では，Halo通信とその通信結果が必要な計算を専用スレッド（Haloスレッド）にまかせる手法をMHDシミュレーションコードに導入し，その性能を評価する．通信と依存関係のある計算をすべてHaloスレッドに担当させることで，計算スレッドに通信に伴う同期を行わせる必要がない．今後の計算機シ

†1 京都大学学術情報メディアセンター
Academic Center for Computing and Media Studies, Kyoto University
†2 九州大学情報基盤研究開発センター
Research Institute for Information Technology, Kyushu University
†3 九州先端科学技術研究所
Institute of Systems, Information Technologies and Nanotechnologies
†4 CREST, JST
JST, CREST

ステムでは，通信コアを持つシステムが多くなる可能性もあり，その通信コアに専用スレッドを割り当てることが可能であれば，計算スレッドが減ることに対するペナルティも無く，実行できることが想定される．

2. Simulation Model

宇宙空間は真空と思われているが，その 99 %はプラズマで満たされている．プラズマとは電離した気体のことであり，帯電している電子とイオンが分かれて存在する状態である．宇宙空間，特に我々の暮らす太陽系においては太陽から太陽風と呼ばれるプラズマの風が常時吹き出しており，太陽系全体にそのプラズマが充満している．このようなプラズマの振る舞いを記述する方程式として Vlasov-Maxwell 方程式がある．これは，無衝突 Boltzmann 方程式と Maxwell 方程式から成る．Vlasov（無衝突 Boltzmann）方程式は以下の形をとる．

$$\frac{\partial f_s}{\partial t}+\vec{v}\frac{\partial f_s}{\partial \vec{r}}+\frac{q_s}{m_s}(\vec{E}+\vec{v}\times\vec{B})\frac{\partial f_s}{\partial \vec{v}}=0 \qquad (1)$$

ここで $\vec{E}$，$\vec{B}$，$\vec{r}$ と $\vec{v}$ はそれぞれ電場，磁場，距離，速度を表す．また，$f_s(\vec{r},\vec{v}_s,t)$ は位置－速度位相空間における分布関数であり，s はイオンや電子など種類を示す．q_s は電荷を m_s は質量を表す．

しかしながら，Vlasov 方程式は多くの成分からなる非線形方程式であり，計算機システムを用いても解くことが非常に難しい．そこで，Vlasov 方程式のモーメントをとることで求められる電磁流体力学（MHD）方程式が，グローバルなプラズマ構造を調べるときには使用されている．MHD 方程式は以下のようになる．

$$\begin{aligned}
&\frac{\partial \rho}{\partial t}=-\nabla\cdot(\mathbf{v}\rho)\\
&\frac{\partial \mathbf{v}}{\partial t}=-(\mathbf{v}\cdot\nabla)\mathbf{v}-\frac{1}{\rho}\nabla p+\frac{1}{\rho}\mathbf{J}\times\mathbf{B}\\
&\frac{\partial p}{\partial t}=-(\mathbf{v}\cdot\nabla)p-\gamma p\nabla\cdot\mathbf{v}\\
&\frac{\partial \mathbf{B}}{\partial t}=\nabla\times(\mathbf{v}\times\mathbf{B})
\end{aligned} \qquad (2)$$

上から，連続の式，運動方程式，圧力変化の式（エネルギーの式），最後が磁場の誘導方程式となる[4]．簡単に言えば，電磁場を考慮した流体力学方程式と呼べる．詳しい導出方法は参考文献を参照されたい[5].

MHD 方程式を解く数値計算法としては，*Ogino* らによって開発された Modified Leap Frog（MLF）法[6，7]という差分法を使用する．これは最初の 1 回を two step Lax-Wendroff 法で解き，続く （l － 1）回を Leap Frog 法で解き，その一連の手続きを繰り返す．図 1 に MLF 法の計算スキームを示す．l の値は数値的に安定の範囲で大きい方が望ましいので，2 次精度の中心空間差分を採用するとき，

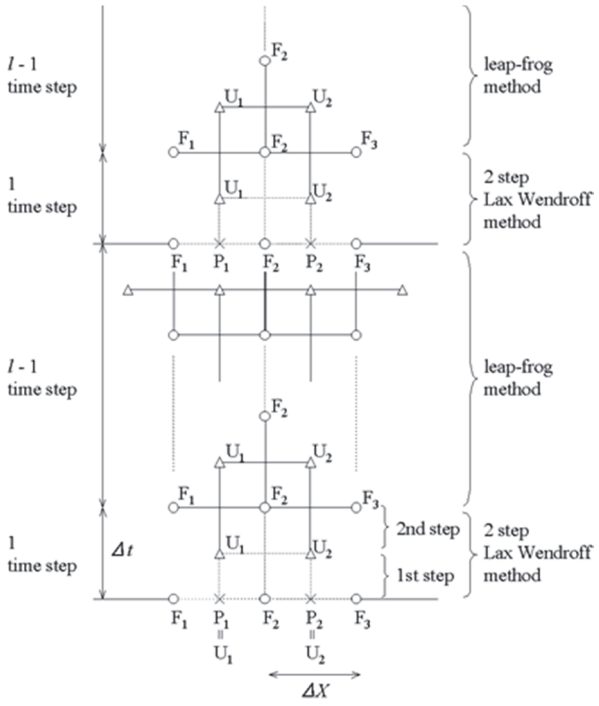

図 1　Modified Leap Frog 法の計算スキーム
Figure 1　Scheme of Modified Leap Frog method

数値精度の線形計算と予備的シミュレーションから $l=8$ に選んでいる．

3. Parallel Implementation

我々のMHDシミュレーションは直交格子を利用しており，いわゆるstencil計算である．計算手法は前述の通り，差分法で解いており，並列化には領域分割を利用し，通信は基本的にはHalo通信のみである[8]．これは計算時に周辺領域の値を利用して，数値を更新しているが，領域分割にともなって周辺領域が存在しなくなるために，周辺プロセスから，必要な値を通信で持ってくることが必要となる．そのために，いわゆるHalo通信と呼ばれる作業が必要となる．この通信には我々の計算がweak scalingということや，通信の安定性を求めて，同期通信（eg. MPI_sendrecv）を用いている．しかしながら，京コンピュータにおいて3万ノードまで利用すると，前述の通りスケーラビリティが下がることが分かっている．この問題を解決するために，今までの研究では，通信専用スレッドを立て，非同期通信を行い，袖領域の通信を行うことが多い[1，2]．この場合，計算スレッドが減少し，全体の計算性能が下がるため，通信時間が全実行時間の半分を占めるような場合を除き，一般には全体の計算コストは上がってしまう．そのため，*Idomura*らは通信終了後にスレッドのダイナミックスケジューリングにより計算に通信スレッドを参加させる工夫がされている[3]．本研究では，それらの手法とは異なり，通信終了後に通信結果を必要とする計算までを通信を行ったスレッド自身に行わせる．これにより，計算スレッドで行われている計算

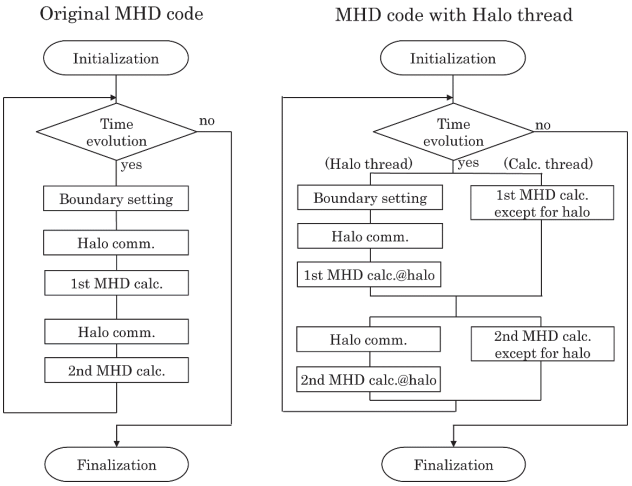

図 2　MHD フローチャート
Figure 2　Flowchart of MHD code

と通信スレッドで行われる通信と計算を完全にオーバーラップさせることができる．さらにこの手法であれば，計算終了後の1回だけ同期を取れば良い. この手法は更新される配列，更新に利用する配列が異なっている（図1の数値計算手法を参照）ために，安全に実行できる．

これまでのMHDシミュレーションのフローチャートとHaloスレッドを導入する場合のフローチャートは図2の通りである．MLF法では図1にあるように2段計算（1st+2nd MHD calc.）で1タイムステップを進める．差分計算を行うためにHalo領域のデータが必要となるため，これまでは計算開始前にHalo通信を行っていた．Haloスレッドを導入した場合は, 計算スレッドは通信をケアする必要が無いため，独立して計算ができる．このようにHaloスレッドと計算スレッドを分けると，計算担当スレッドが減る分，計算性能が下がり，通信時間が隠蔽できても全体的には計算性能が下がることが多い.

実際の MHD シミュレーションコードへの実装は図 3 のようになる（Fortran+OpenMP 利用）．Halo スレッド（スレッド番号は 0）はまず通信を行い，その後 Halo 領域を計算している．その一方で，他のスレッドは計算だけを行っている．実装自体は現状でも非常にシンプルであり，また Halo スレッドでの通信は同期通信でも構わず，非同期通信におけるプログレスチェック[3]も必要無い.

4. Performance Measurements

本研究では, スレッド数と計算サイズを変更すると, Haloスレッドを導入した場合としない場合で性能がどのように変化するか調べる．利用する計算機システムは，九州大学情報基盤研究開発センターのFujitsu PRIMEHPC FX10（以下，FX10），Hitachi HA8000-tc/HT210（以下，HA8000）と京都大学学術情報メディアセンターのCRAY XC30（以下，XC30）の3種類である．各計算機システムの情報は表1に掲載している．それぞれコンパイラ，インターコネクトなどが異なるシステムである.

```
  call init_mhd(f) ! Initilization
!
!----Time evoluation---!
  do time = 1, 1000
!
!----Thread setting----!
!$OMP PARALLEL PRIVATE(myid,mylid,ks,ke,ii)
    myid = omp_get_thread_num()
    nthreads = omp_get_num_threads() - 1
    mylid = myid - 1
    kmod = mod(nzz-2, nthreads)
    kdiv = floor(real((nzz-2)/nthreads))
!
    if (kmod > mylid) then
      ks = mylid * (kdiv + 1) + 1
      ke = ks + kdiv
    else if (kmod == mylid) then
      ks = mylid * (kdiv + 1) + 1
      ke = ks + kdiv - 1
    else
      ks = mylid * kdiv + kmod + 1
      ke = ks + kdiv - 1
    end if
!
!----Halo thread----!
    if(myid == 0) then
      call boundary(f)   ! boundary setting
      call halo3d(f)     ! Halo communication
!
      do k = zs, ze
        call mhd_calc(f) ! MHD calc. at Halo
      end do
!
!----Calc thread----!
    else
      do k = ks+1, ke-1
        call mhd_calc(f) ! MHD calc.
      end do
    end if
!
!$OMP END PARALLEL
.
.
.
  end do
```

図 3　Halo スレッドの実装例
Figure 3　Implementation of Halo thread

4.1 計算サイズとスレッド数を変化させた場合

計測に利用したプロセスは16MPI並列（2×2×4の3次元領域分割）で，計算サイズは各プロセスに100^3，200^3，300^3，400^3の3次元グリッドを割り当てた. スレッド数は各計算機システムにより異なっている.

図4にFX10におけるHaloスレッドを導入した場合と導入しない場合の測定結果を載せる. FX10はノードあたりのメモリが小さいために, 400^3の4スレッドは実行できなかった. 計算サイズの小さい100^3では8スレッド時だけHaloスレッドを導入した場合の性能が高く，それ以外の計算サイズでは16スレッドを利用した場合にHaloスレッド導入効果が見

表 1　FX10，HA8000，XC30 のシステム構成

Table 1　System of FX10, HA8000 and XC30

システム名	FX10	HA8000	XC30
CPU	SPARC64 IXfx (1.848GHz, 16cores)	Xeon E5-2697v2 (2.7 GHz, 12cores)×2/node	Xeon E5-2695v3 (2.3 GHz, 14cores) ×2/node
DRAM	DDR3-1333 32GB	DDR3-1600 256GB	DDR4-2133 64GB
ノード数	768	965	416
Interconnect	Tofu Interconect (双方向 5GB/s)	InniBand FDR (片方向 6.78GB/s)	Aries (片方向 15.7GB/s)
OS	XTCOS	Red Hat Linux Enterprise 6	Cray Compute Node Linux
Compiler	Fujitsu Fortran Compiler ver. 1.2.1-09	Intel Fortran Compiler ver. 15.0.3	Cray Compiler ver. 8.3.9
Compiler option	-Kfast,openmp	-fast -openmp	-O3 -h omp
MPI	Fujitsu MPI ver. 1.2.1-09	Intel MPI ver. 5.0	Cray MPT (MPI) ver. 7.1.3

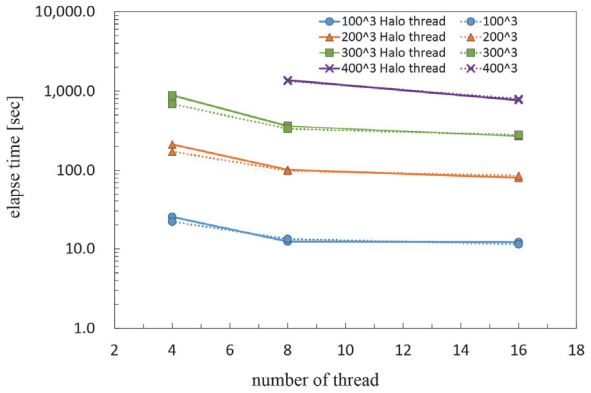

図 4　FX10 における計算サイズとスレッド数変化時の Halo スレッドの効果

Figure 4　Effects of Halo thread with variation of calculation size and number of thread on FX10

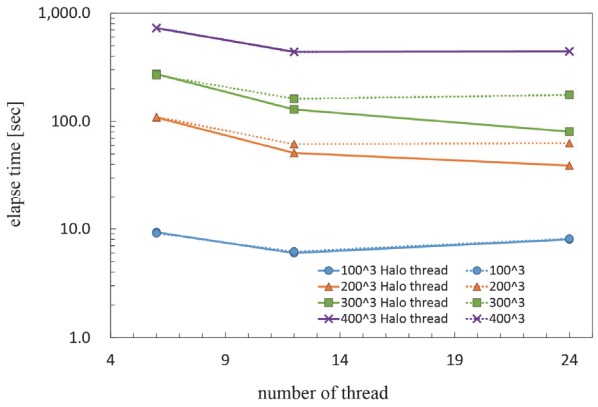

図 5　HA8000 における計算サイズとスレッド数変化時の Halo スレッドの効果

Figure 5　Effects of Halo thread with variation of calculation size and number of thread on HA8000

えている．計算性能自体に注目すると，16スレッドは8スレッドに比べて，性能がスケールしておらず，スレッドあたりの計算性能が悪いため，Haloスレッドを導入することで，Halo通信・計算部分が隠蔽される分，Haloスレッド導入効果が見えると考えられる．100^3で8スレッドは見積通りに，計算スレッドが担当する計算量が少ないため，スレッドの計算負荷が低く，通信負荷が高いため，Haloスレッドの効果が見えている．

次にHA8000での計測結果を図5に載せる．まず，HA8000はノードあたり2ソケットシステムであるため，2CPU分の

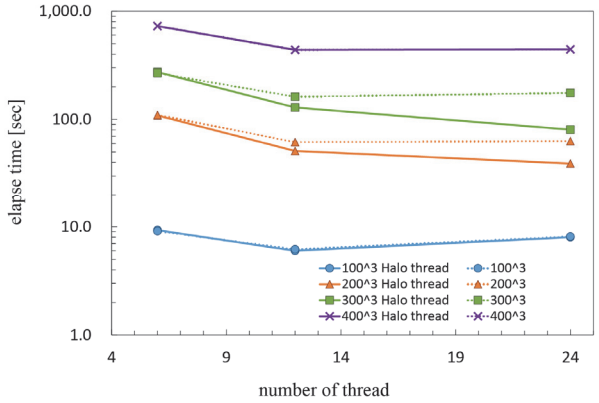

図 6　HA8000 における計算サイズとスレッド数変化時の Halo スレッドの効果

Figure 6　Effects of Halo thread with variation of calculation size and number of thread on XC30

24スレッドではほとんど性能がスケールしていない．Haloスレッドの効果は12と24スレッド利用時ではすべての計算サイズで現れている．24スレッド時はFX10の16スレッドと同様の理由で効果が見えていると考えられる．100^3と400^3ではあまりHaloスレッドの効果が見えないが，200^3と300^3では明らかに効果があり，スレッド数と計算サイズのバランスが良いことが想像される．

最後にXC30で計測を行った（図6）．XC30も2ソケットシステムのため，28スレッドでは計算性能がほぼ上がらない．この計測では14スレッドで300^3と400^3の計算サイズでHaloスレッドの効果が見えた．その他ではHaloスレッド導入した場合の計算性能が明らかに悪い場合が多く，計算システムのスレッドあたりの計算性能が高く，通信性能も高いと想像される．

4.2　Weak and Strong Scaling

次に多くのノード数を利用することができた FX10 において，Halo スレッドを導入した場合の weak scaling と strong scaling の性能計測を行った（図 7）．ここでは 8 スレッドを利用し，最小プロセスは 16，最大プロセスは 1024 プロセスである．プロセス分割前の計算サイズが 800×800×1600 と 200×200×400 の 2 種類の strong scaling とプロセスあたり 100×100×100 の weak scaling での測定結果を載せている．weak scaling の 100×100×100 は 4.1 において FX10 で Halo スレッド導入効果があったサイズであり，プロセス数を増

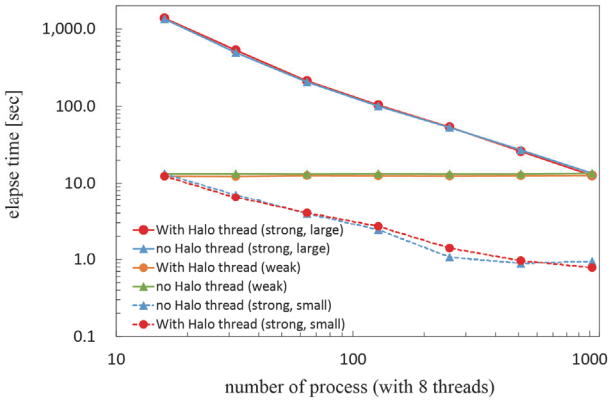

図 7　FX10 における Halo スレッドの weak/strong scaling での性能

Figure 7　Performances of Halo thread in weak and strong scalings on FX10

やしても性能劣化がないことが計測結果からよく分かる．サイズが大きい場合の strong scaling ではプロセス数が 512 を超えると，Halo スレッドを導入した場合の性能が高くなっている．また計算サイズが小さい場合は，プロセス数が少ない時には，プロセスあたりの計算量が計算サイズの大きい strong scaling の続きのような変化であるため，Halo スレッド導入効果が見えるが，プロセス数が大きくなってくると，キャッシュに計算サイズが載るために，計算性能が高くなり，Halo スレッド導入効果は無くなる．さらにプロセス数が増えると計算に対する通信の時間が大きくなるため，512，1024 プロセスで再び Halo スレッド導入効果が出ている．ここから分かるように計算スレッドの計算時間が Halo スレッドの通信・計算時間より短くならなければ，スケーラビリティは悪くならない．

5. Effective Condition of using Halo thread

3 種類の計算機システムにおいて，計算条件を変化させ Halo スレッドの効果を調べたが，効果があるときと無いときがあった．ここでは FX10 の測定結果を元にどのような条件下で Halo スレッド導入効果があるのかを議論する．表 2 に図 4 の測定条件において，計算スレッドや Halo スレッドでの経過時間計測した結果を載せる．ここでは計算サイズが 100^3 でスレッド数が 8 の時に Halo スレッドの効果があったが，Halo スレッドと計算スレッドの総経過時間（elapse at H thread と at C thread）を比べると両者が同じくらいの時間であり，スレッドでの経過時間のバランスが取れていることが分かる．Halo スレッドの効果が無い他の倍では，どちらかの elapse が遅いなどバランスが悪くスレッドの待ちが発生している．

このバランスが良いというは，スレッドあたりの計算時間と通信時間の条件によって決まる．つまり，Halo スレッド導入により，計算スレッド数が減って Halo（袖）領域外の計算にかかる時間が増加するが，この増加分が，Halo スレッドを導入していない場合の通信時間より短い時に Halo スレッド導入効果はある．もちろん，最低限 Halo スレッドでの計算時間が他の計算スレッドの計算時間と同じくらいか，それより短いときも条件の一つである．表 2 には Halo スレッド導入により増加する計算量の割合（計算量増加率）と，Halo スレッド非導入時の通信時間の計算時間に対する割合（通信時間の割合）がそれぞれ計算してあり，この値を比べると，前述の条件である通信時間の割合が計算時間の増加分より小さい場合では，Halo スレッドの効果がある結果になっている．このように Halo スレッド導入効果がある条件がはっきりしているため，実際のシミュレー

表 2　FX10 における計算サイズとスレッド数変化時の Halo スレッドと計算スレッドの経過時間．表中で H thread は Halo thread，C thread は Calc thread を表す．

Table 2 Elapse time of Halo and calculation threads with the variation of calculation size and number of thread on FX10. H thread is Halo thread and C thread is Calc thread in the table.

With Halo thread											
Thread number	4	4	4	8	8	8	8	16	16	16	16
grid 数/process	100^3	200^3	300^3	100^3	200^3	300^3	400^3	100^3	200^3	300^3	400^3
1st H comm at H thread	0.013	0.047	0.100	0.010	0.039	0.082	0.133	0.009	0.037	0.075	0.125
1st MHD calc at H thread	0.066	0.256	0.580	0.068	0.249	0.577	1.264	0.079	0.381	0.725	1.257
1st elapse at H thread	0.079	0.303	0.680	0.079	0.288	0.659	1.397	0.089	0.418	0.800	1.382
1st elapse at C thread	0.189	1.595	5.453	0.091	0.770	2.625	9.992	0.072	0.604	1.868	5.678
2nd H comm at H thread	0.035	0.134	0.361	0.028	0.098	0.219	0.381	0.025	0.093	0.195	0.358
2nd MHD calc at H thread	0.064	0.268	0.645	0.065	0.259	0.608	1.502	0.072	0.382	0.833	1.476
2nd elapse at H thread	0.099	0.402	1.007	0.094	0.357	0.827	1.883	0.097	0.475	1.028	1.834
2nd elapse at C thread	0.206	1.822	6.370	0.097	0.854	2.930	11.485	0.072	0.641	1.993	6.352
sampling time	25.245	212.559	885.331	12.367	101.900	362.068	1373.371	12.165	80.808	270.063	766.988
計算量増加率	125%	129%	131%	108%	111%	112%	113%	100%	103%	105%	105%
No Halo thread results											
Sampling time	21.982	172.915	688.850	13.298	99.172	334.781	1340.867	11.429	85.705	279.833	797.145
通信時間の割合	8%	4%	2%	11%	6%	4%	2%	11%	6%	4%	3%

ションコードに導入する際には，この条件を満たすように計算設定をする必要がある．

6. まとめ

本研究では宇宙プラズマを扱う MHD シミュレーションという stencil 計算に対して, いわゆる袖領域である Halo 領域での処理を専門に担当する Halo スレッドを導入し, その効果を調べた．Halo スレッドが通信スレッドと異なるのは, Halo 領域の通信だけでなく，その領域での計算を担当することで計算スレッドとの同期が必要無いこと，また，計算も担当することで 1 スレッド分計算スレッドが減り計算性能が下がる影響を抑えていることである．このようにスレッド数と計算サイズが性能に与える影響が大きいと考えられるため，スレッド数を変化させた場合と，計算サイズを変化させた場合の性能測定を FX10，HA8000 と XC30 において行った．この結果，各スレッドが担当する計算量が少ない場合に Halo スレッドの導入効果が出やすいことが分かった．次に weak scaling と strong scaling を利用し性能を測定したところ，プロセス数が増加する毎にスレッドあたりの計算量が減少するため，Halo スレッドの効果が明らかに見え，またスケーラビリティも良かった．これらの結果において Halo スレッドと計算スレッドの経過時間や通信時間などを詳しく見ると，Halo スレッドと計算スレッドの経過時間が同程度の時に効果が出ていることが分かった．この条件は，Halo スレッド導入により計算スレッドの計算負荷増加割合と，Halo スレッドを導入しない場合の通信時間が同程度か, 通信時間が長い場合に Halo スレッドの効果が出ることを表している．

今回の結果から，Halo スレッド導入はスレッド数と計算サイズのバランスが重要ではあるが，基本的には weak scaling であればスケーラビリティは劣化せず，strong scaling においても通信時間が計算スレッドの計算時間より大きくならない条件であれば，スケーラビリティは劣化しないと考えられる．特に我々の MHD シミュレーションはエクサスケールにおいても weak scaling の計算が続くため，Halo スレッドの導入で高いスケーラビリティを維持できると想定される．さらに，Halo スレッドの処理自体に ACP[9]を用いて Halo 通信と Halo 領域計算をフロー型の処理にし，さらなる最適化を現在行っており，より高いスケーラビリティが期待される．また，このような従来と異なった通信モデルを導入する際に，Halo スレッドのように主計算部分と通信部分が分けられている方が，主計算部分の最適化に通信モデルが影響を与えず，導入メリットが出やすいと考えられる．

また，Halo スレッド導入効果が出ない場合に対しては，Halo スレッドの計算スレッドへの参加やその逆をダイナミックにスレッドスケジューリングする手法が考えられるが，Halo スレッドと計算スレッド間で同期が必要となるため，同期に対する特別な最適化が必要となる．

謝辞 本研究の計算結果は九州大学情報基盤研究開発センターと京都大学学術情報メディアセンターの計算機システムを利用して得られた．本研究は JST，CREST の研究領域「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」の研究課題「省メモリ技術と動的最適化技術によるスケーラブル通信 ライブラリの開発」の支援を受けている．

参考文献

1) Sur S, Jin HW, Chai L and Panda DK, RDMA read based rendezvous protocol for MPI over Infiniband: Design alternatives and benefits. In: ACM SIGPLAN symposium on principles and practice of parallel programming, (PPOPP 2006) (ed J Torrellas and S Chatterjee), New York, USA, 29-31 March 2006, pp. 32-39. New York: ACM Press.
2) Preissl R, Wichmann N, Long B, Shalf J, Ethier S and Koniges A, Multithreaded global address space communication techniques for gyrokinetic fusion applications on ultra-scale platforms. In: 2011 international conference for high performance computing, networking, storage and analysis (SC ‘11), Seattle, USA, 14-17 November 2011. New York: ACM Press.
3) Idomura, Y., Nakata, M., Yamada, S., Machida, M., Imamura, T., Watanabe, T., Nunami, M., Inoue, H., Tsutsumi, S., Miyoshi, I., Shida, N.: Communication-overlap techniques for improved strong scaling of gyrokinetic Eulerian code beyond 100k cores on the K-computer. Int. J. High Perform. Comput. Appl. 28, 73-86, 2013.
4) Chang, C. L. and Lee, R. C. T.: Symbolic Logic and Mechanical Theorem Proving, Academic Press, New York, 1973.
5) R. O. Dendy,『Plasma Dynamics』, Oxford University Press, 1990.
6) T. Ogino, R. J. Walker, M. Ashour-Abdalla, A global magnetohydrodynamic simulation of the magnetopause when the interplanetary magnetic field is northward, IEEE Trans. Plasma Sci.20, 817.828, 1992.
7) Fukazawa, K., T. Ogino, and R.J. Walker, "The Configuration and Dynamics of the Jovian Magnetosphere", J. Geophys. Res., 111, A10207, 2006.
8) Fukazawa, K., T. Umeda, Performance measurement of magnetohydrodynamic code for space plasma on the typical scalar type supercomputer systems with the large number of cores, International Journal of High Performance Computing, doi:10.1177/1094342011434813, 2012.
9) ACE Project
http://ace-project.kyushu-u.ac.jp/main/jp/index.html