Halo スレッドと Halo 関数を用いた MHD シミュレーションの高効率並列化

深沢圭一郎[†1, †5]　森江善之[†2, †5]　曽我武史[†3, †5]　高見利也[†4, †5]　南里豪志[†2, †5]

概要：プラズマを流体的に解く MHD（電磁流体）シミュレーションは，ステンシル計算であり，計算を行う点の周辺データを利用して計算を進めるため，並列化に伴いいわゆる Halo（袖）通信が発生する．この Halo 通信に伴い，通信データのパック/アンパックやプロセス間での同期も必要となり，並列数が大きくなるにつれて，並列化効率の劣化が無視できなくなってきている．そこで，我々は「計算」と「通信が必要な計算と通信」を分け，「通信が必要な計算と通信」を担当する Halo スレッドを MHD シミュレーションに導入した．この結果，ある条件下では Halo スレッド導入により並列計算性能効率の向上を確認できた．さらに，この Halo スレッド上の「通信が必要な計算と通信」は通信が終わらなければ，計算を行えない処理であり，非効率であったため，Halo 通信自体を効率的に行え，通信と計算をパイプライン的に行うことを可能とする Halo 関数を開発し，MHD シミュレーションに導入した．これを利用した性能評価では，Halo スレッドでの計算時間が減り，通信と計算のパイプライン処理の効果が確認された．

キーワード：並列計算，高性能計算，MHD シミュレーション，宇宙プラズマ

High efficiency parallel computation of MHD simulation with Halo thread and Halo functions

KEIICHIRO FUKAZAWA[†1, †5]　YOSHIYUKI MORIE[†2, †5]　TAKESHI SOGA[†3, †5]
TOSHIYA TAKAMI[†4, †5]　TAKESHI NANRI[†2, †5]

Abstract: Magnetohydrodynamic (MHD) simulation is stencil code and often used to study the macro scale plasma. The stencil computation requires the neighboring data to proceed the calculation. Thus, the Halo communication is needed in parallel computation. It is important for the parallel scalability of stencil computation to decrease the Halo communication time. 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. In addition, we develop the Halo communication functions which perform the Halo communication and related calculation effectively and introduce the functions to the MHD simulation code. As the results, we have obtained good performances and confirmed the decrease of elapse time in the Halo thread.

Keywords: Parallel computing, High performance computing, MHD Simulation, Space plasma

1. はじめに

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いる．*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 における通信進捗チェック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

†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 大分大学工学部知能情報システム工学科
Department of Computer Science and Intelligent Systems, Oita University
†5 CREST, JST
JST, CREST

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．また，Halo スレッド上で効率的に通信と計算を行うために，Halo 関数を作成し，その性能を評価する．

2. シミュレーションモデル

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．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 方程式があるが, 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，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 6 点差分法を使用する．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，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. Halo スレッドの導入

3.1 Halo の実装

Stencil計算であるMHDシミュレーションでは，並列化に領域分割を利用し，通信は基本的にはHalo通信のみである

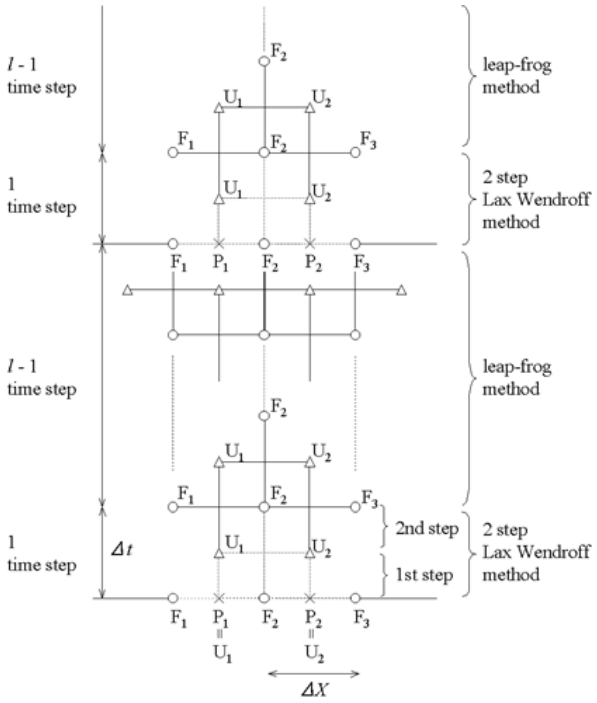

図 1　Modified Leap Frog 法の計算スキーム
Figure 1　Scheme of Modified Leap Frog method

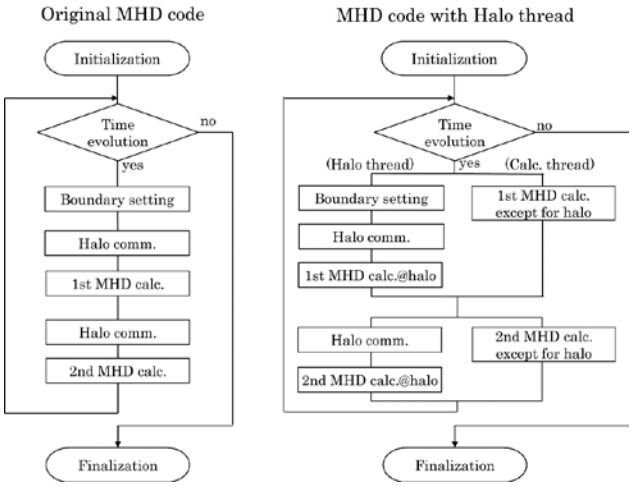

図 2　MHD フローチャート
Figure 2　Flowchart of MHD code

[8]．差分法で隣接領域の値を利用して，数値を更新していく際に, 領域分割により隣接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．この通信が，Halo通信と呼ばれる．現状でこのHalo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，現状の並列計算実装では，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，既存の研究では，通信専用

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

図 3　Halo スレッドの実装例
Figure 8　Implementation of Halo thread.

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（Halo領域のデータを利用する計算）までを通信を行うスレッドに行わせる．このスレッドをHaloスレッドと呼ぶ．これにより，計算スレッドで行われている計算と通信スレッドで行われる通信（通信データのパック/アンパック含む）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．

表 1　FX10，XC30 のシステム構成
Table 1　System of FX10 and XC30

システム名	FX10	XC30
CPU	SPARC64 IXfx (1.848GHz, 16cores)	Xeon E5-2695v3 (2.3 GHz, 14cores) ×2/node
DRAM	DDR3-1333 32GB	DDR4-2133 64GB
ノード数	768	416
Interconnect	Tofu Interconect (双方向 5GB/s)	Aries (片方向 15.7GB/s)
OS	XTCOS	Cray Compute Node Linux
Compiler	Fujitsu Fortran Compiler ver. 1.2.1-09	Cray Compiler ver. 8.3.9
Compiler option	-Kfast,openmp	-O3 -h omp
MPI	Fujitsu MPI ver. 1.2.1-09	Cray MPT (MPI) ver. 7.1.3

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.

3.2 Halo スレッドの性能測定

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）と京都大学学術情報メディアセンターのCRAY XC30（以下, XC30）の2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.

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XC30で計測を行った（図5）．XC30は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．

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 6）．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 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．

3.3 Halo スレッドの効果がある条件

2 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

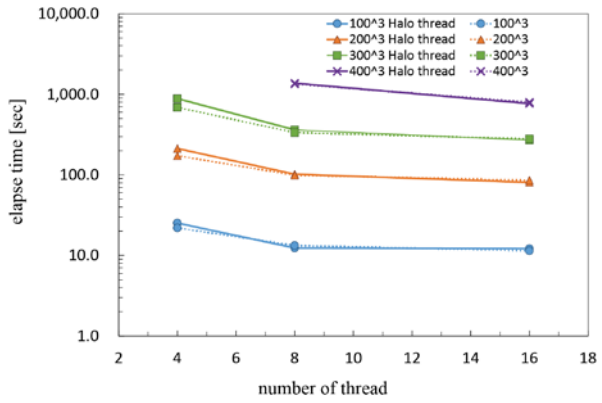

図 4　FX10 における計算サイズとスレッド数変化時の Halo スレッドの効果

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

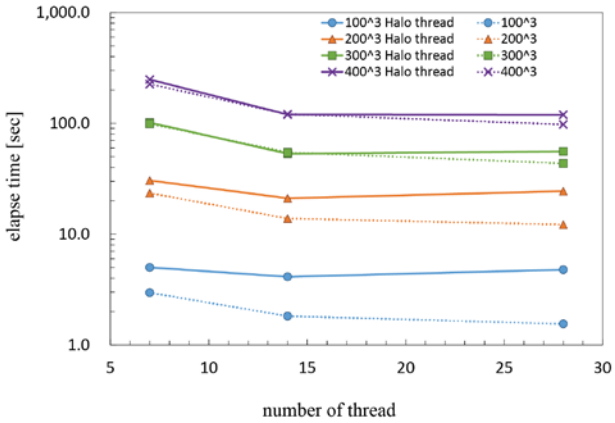

図 5　XC30 における計算サイズとスレッド数変化時の Halo スレッドの効果

Figure 5　Effects of Halo thread with variation of calculation size and number of thread on XC30

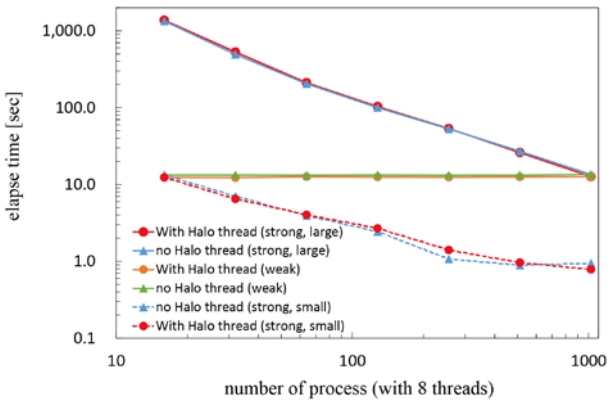

図 6　FX10 における Halo スレッドの weak/strong scaling での性能

Figure 6　Performances of Halo thread in weak and strong scaling on FX10

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%

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 領域外の計算にかかる時間が増加するが（t_{cala}），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（t_{com}）より短い時に Halo スレッド導入効果はある．図 7 にシミュレーション経過時間のモデルを示すが，ここにある $t_{cala} < t_{com}$ が Halo スレッド導入効果のある条件となる．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．または，自動チューニングを利用し，計算条件を設定することが有効であると考えられる．

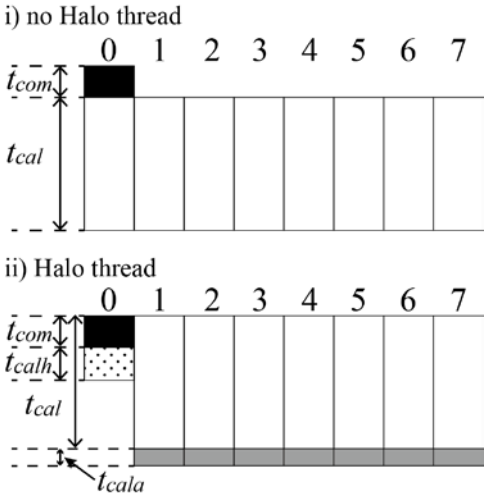

図 7　Halo スレッド有り/無しのスレッド経過時間モデル．t_{com} は Hal 通信時間，t_{cal} は Halo スレッド以外での MHD 計算時間．t_{calh} は Halo 領域の MHD 計算時間，t_{cala} は Halo スレッド導入に依る MHD 計算増加時間

Figure 7　Elapse time model of with/without Halo thread using 8 threads. t_{com} is the halo communication time, t_{cal} is MHD calculation time without the Halo thread. t_{calh} is MHD calculation time of halo region, and t_{cala} is an additional MHD calculation time due to decrease of calculation thread.

4. Halo 関数

4.1 Halo 関数の構造

Halo スレッドで実行される Halo 通信は，固定された相手と固定量を通信する場合が多い（AMR などは除く）．また，複数次元の領域分割より並列化を行った stencil 計算では，通信の回数を削減するために，通信順序を固定して Halo 通信を行うことが多い[9]．そこで，Halo 通信に必要な各種パラメータを事前に登録し，Halo 領域にある面，線と点データを効率的に送受信できる下記の関数を作成した．またパック/アンパックに時間がかかることがわかっているため，ブロッキング化することで，効率化を行っている．

・halo_init：Halo 通信の初期設定を行う．本関数は，与えられたプロセス分割の次元数及び，次元毎のプロセス数に応じて，自動的に自プロセスの論理座標を割り当てる．また，Halo 通信の対象となる行列(配列)と，その次元数と次元毎の要素数及び，論理分割次元に配列のどの次元が相当するかを指示することにより，行列を halo 通信の対象として登録する．halo_init 関数は通信に参加する全プロセスが通信の対象であるとみなして論理座標割り当てを行う．

・halo_isend：袖領域の送信を行う．本関数は，指示したハンドルに登録されている自プロセスの配列内の指示した範囲から（パッキングも行う），指示した方向の隣接プロセスへの通信を開始する．本関数は通信完了を待つことなく完了する．通信の完了は，halo_wait または，halo_test 関数により知ることができる．

・halo_irecv：袖領域の受信を行う．本関数は，指示した方向の隣接プロセスから，指示したハンドルに登録されている自プロセスの配列に内の指示した範囲へ（アンパッキングも行う）の通信を開始する．本関数は通信完了を待つことなく完了する．通信の完了は，halo_wait または，halo_test 関数により知ることができる．通信完了前に受信領域を書き換えた場合には，データの内容は不定になる．

・halo_wait：本関数は，指示したリクエスタに対応する通信が完了するまで待つ．halo_isend 関数に対応した halo_wait 関数の完了後は，通信領域を書き換えても書き換え前のデータが受信側に送られていることが保障される．また，halo_irecv 関数に対応した halo_wait 関数の完了後に通信領域から読み出されるデータは，送信側から送られてきたデータであることを保障する．

・halo_finalize：本関数は，halo_init で登録された内容を解放して，halo 通信を終える．

この関数群はC言語と現在はMPIライブラリで実装されているが，ACP ライブラリ[10]で実装することも可能である．アプリケーション開発者は Halo 関数インターフェースを利用することで，様々な通信ライブラリを陰に利用することが可能である．更に，大規模数値計算に利用者の多い Fortran から利用できるように wrapper が用意されており，本研究では Fortran から Halo 関数を利用している（図 8 参照）．この Halo 関数群を利用すると，ランク配置などは halo_init で行うため，典型的な stencil 計算であれば，MPI を陽に呼ばずに並列計算を実装することができる．

4.2 Halo 関数の性能測定

ここでは，3.2 で Halo スレッド導入効果があった XC30 の結果に Halo 関数を導入し，その効果を調べる．計測に利用したプロセスは 8，16，32MPI 並列（3 次元領域分割）で，計算サイズは各プロセスに 100^3，200^3 の 3 次元グリッドを割り当てた．スレッド数は 7 スレッドを利用した．

```
  call halo_init(f) ! Halo Initialization
!
   do l = 1, 26
     call halo_irecv(f) ! Halo receive
     call halo_isend(f) ! Halo send
     call halo_wait ! for receive
     call halo_wait ! for send
!
   end do
!
  Call halo_finalize
```

図 8　Halo 関数の実装例

Figure 8　Implementation of Halo function.

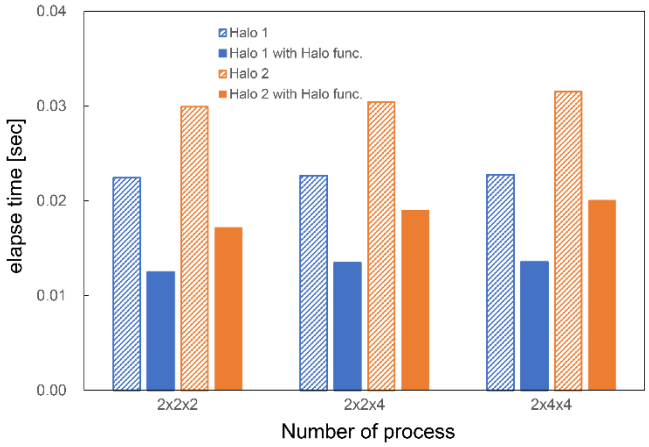

図 9　XC30 における 100^3 格子利用で Halo 関数を利用しない場合，Halo 関数を導入した場合の測定結果．

Figure 9　Performance of regular halo communication and Halo functions with 100^3 grid size on XC30.

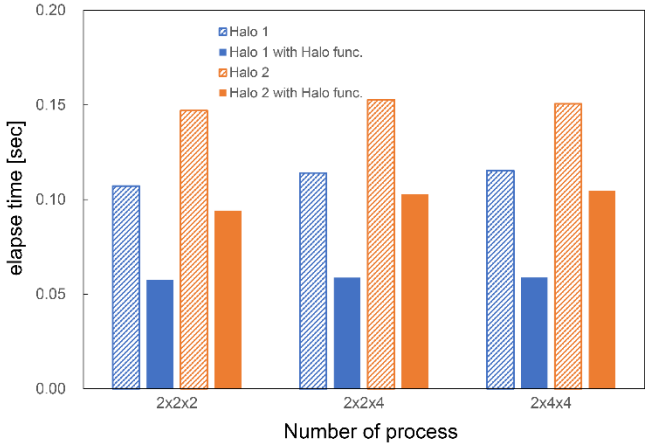

図 10　XC30 における 200^3 格子利用で Halo 関数を利用しない場合，Halo 関数を導入した場合の測定結果．

Figure 10　Performance of regular halo communication and Halo functions with 200^3 grid size on XC30.

図9，10にHalo関数を利用した場合，利用しない場合のMHDシミュレーション中におけるHalo通信経過時間計測結果を載せる．図2にあるように我々のMHDシミュレーションでは一つのタイムステップ中に2回Halo通信があり（計算が2段階あるため），それぞれのHalo通信を図9，10内ではHalo 1，Halo 2としている．本研究で利用している差分法の計算手法により，2段階目は1段階目の計算より配列を多く利用するために，Halo2の通信量がHalo1の2.5倍となっている．

まず，プロセス数が増加した場合に，Halo関数有り無しともに通信時間はそれほど変化が現れておらず，スケーラビリティは良いことがわかる．Halo関数の導入に伴う性能向上は，Halo 1の方が大きく（約2倍の性能向上），また，利用格子数が少ない（100^3）場合が大きい結果となった．Halo関数導入に伴う性能向上は，本計算ではストライドメモリアクセスになっているパック/アンパックのブロッキング化による最適化効果、また，パック/アンパックと通信のオーバーラップの効果が出ていることを示している．利用格子数による効果の違いも，Halo関数が通信自体ではなく，パック/アンパックや通信順序などの効率化を行うため，通信自体が長くかかる場合には性能向上が現れにくいと考えられる．

5. 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 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 スレッド上での効率的な Halo 通信を行うために Halo 関数を開発し，その効果を調べた．XC30 において Halo 関数の効果を調べたところ，通信量が少ない Halo 通信（Halo 1）で 2 倍程度の通信性能向上が確認できた．また，計算サイズが小さい場合に性能向上効果が出やすいことも確認できた．計算サイズが小さい場合は，通信時間が短くなるため，Halo 関数で効率化を行う部分の割合が多くなるために性能向上が現れやすいと考えられる．

本研究で開発した Halo 関数群や Fortran で利用するための wrapper は ACE Project で公開する予定である．Halo 関数は Halo スレッドと独立で利用でき，これらを利用した MPI を明示的に呼ばずに stencil で並列計算可能なフレームワークも準備する予定である．

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