スカラー並列機を用いた高効率ＭＨＤコードの開発

荻野竜樹、中尾真季（名古屋大学太陽地球環境研究所）

近年、大規模数値計算に使用できるスーパーコンピュータが、ベクトル並列機からスカラー並列機へと急速に移りつつある。国内でも日立と富士通がクラスター型のスカラー並列機の市場化に移行し、ベクトル並列機の開発を継続しているのはＮＥＣのみとなった。こうした状況下、スカラー並列機で効率良く動くプログラムの開発が急務となってきたが、欧米でもその高効率の並列計算プログラムの開発が最近１０年間の懸案問題で未だに成功したとは言えない状況である。私達は、そのスカラー並列機で高速計算できる一つの有効な方法を見出したので、その方法の要点と富士通の PRIMEPOWER HPC2500 などのスカラー並列機を含むスーパーコンピュータで実施した現在までのテスト計算の結果を示す。

１．まえがき

コンピュータシミュレーションを実行するスーパーコンピュータをめぐる環境が急速に変わりつつある。それは、ベクトル並列機からスカラー並列機への移行である。欧米では、約１０年前からスカラー並列機に代わり、ベクトル並列機はコスト高とみなされたために一時は市場から駆逐された。しかし、2003-2004 年にベクトル並列機である日本の地球シミュレータが世界最高速のスーパーコンピュータの栄誉を授かるにいたって、再評価の気運が起こり、米国政府の援助を受けてクレイ社が再びベクトル並列機(CRAY X1E)の開発生産に着手した。一方国内では、先ず、日立がベクトル並列機からスカラー並列機（Hitachi SR8000, SR11000, 疑似ベクトル機ともいわれる）へ移行して富士通もそれに続き（PRIMEPOWER HPC2500）、2004 年度にベクトル並列機の開発を継続しているのはＮＥＣのみ（NEC SX6, SX7, SX8）となった。

なぜ、ベクトル並列機からスカラー並列機への移行が急激に起こったのか、それはコストパフォーマンスが良いに違いないと多くの人が信じたためである。それでは、スカラー並列機がベクトル並列機より本当にコストパフォーマンスが良かったのか。これが利用者にとって最大の疑問点であったが、実際にスーパーコンピュータの高速計算を必要としている実用プログラムで実証される前に、スカラー並列機への移行が急速に進行したのが現実であった。大規模計算のプログラム実行で、ベクトル並列機では約４０％の絶対効率が出ていたものが、スカラー並列機では１０年以上の努力が世界中でなされてきたにもかかわらず、３－７％の絶対効率しか到達できないと言われてきた。この低い絶対効率の壁は超えられないものなのか。私達は、スカラー並列機で高速計算できる方法をいろいろ検討してきた結果、可能性のある一つの有効な解決方法を見出した。その方法と種々のスーパーコンピュータに対して実施した現在までのテスト結果を示すと共に、2005 年に更新された名古屋大学情報連携基盤センターのスカラー並列型のスーパーコンピュータ Fujitsu PRIMEPOWER HPC2500 に対する予測と期待、及び３次元ＭＨＤコードを用いたテスト計算結果について述べる。

２．ＭＨＤシミュレーションコード

私達は、３次元ＭＨＤ（電磁流体力学的）シミュレーションコードを用いて、太陽風と地球磁気圏相互作用のグローバルな計算機シミュレーションを実施してきた（[1]-[7]）。ダイポール磁場の軸が傾い

ている時（傾き角度３０°で北半球が夏）の地球磁気圏の磁場構造の例を、ＩＭＦ（惑星間磁場）が南向きと北向きの場合について図１に示す。更に、ＩＭＦが南北朝夕面で回転している時で、北向きから南向きに変化した後に起こるプラズモイドの尾方向への放出時の VRML（Virtual Reality Modeling Language）による３次元可視化（[5],[7], [9]）の例を図２に示す。この様なグローバルなモデルでは、外側境界は影響を減らすためになるべく遠くに置き、かつ空間分解能を上げる必要から、計算方法の改良が行われる一方、最大級のスーパーコンピュータの利用、それも効率的な利用が必須である。ＭＨＤモデルでは、Modified Leap-Frog 法を用いて、ＭＨＤ方程式とマックスウェル方程式を初期値境界値問題として解いている。（[8],[9]）

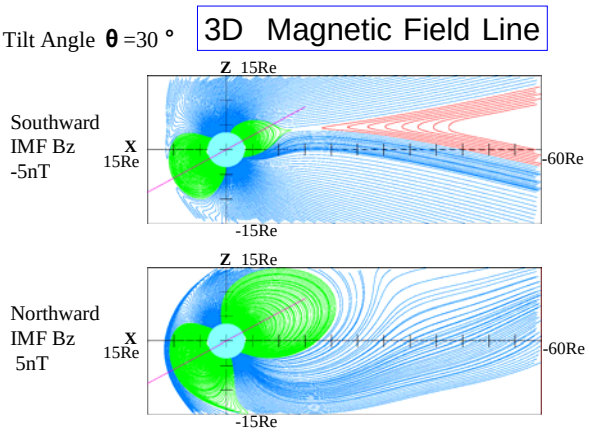

図1 惑星間磁場(IMF)が南向きと北向きの場合で、傾いたダイポール磁場に対する地球磁気圏の磁場構造

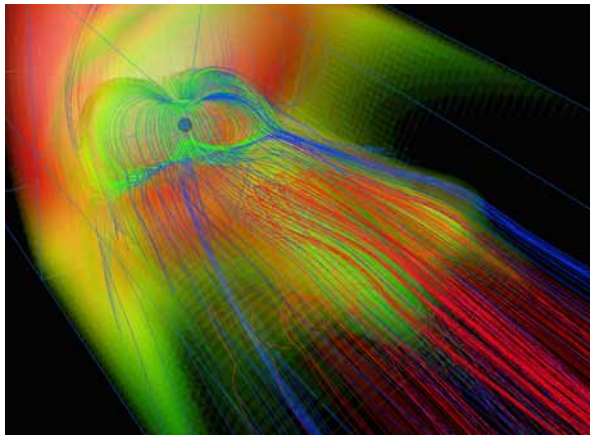

図2 IMFが北向きから南向きに変化した後に起こる尾部磁気リコネクションの発生とプラズモイドの尾方向への放出時の地球磁気圏構造

＜基礎方程式＞

MHD モデルの基礎となる規格された MHD 方程式と Maxwell 方程式

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot (\mathbf{v}\rho) + D\nabla^2 \rho \qquad (1)$$

$$\frac{\partial \mathbf{v}}{\partial t} = -(\mathbf{v}\cdot\nabla)\mathbf{v} - \frac{1}{\rho}\nabla p + \frac{1}{\rho}\mathbf{J}\times\mathbf{B} + \mathbf{g} + \frac{1}{\rho}\Phi \qquad (2)$$

$$\frac{\partial p}{\partial t} = -(\mathbf{v}\cdot\nabla)p - \gamma\, p\nabla\cdot\mathbf{v} + D_p\nabla^2 p \qquad (3)$$

$$\frac{\partial \mathrm{B}}{\partial t} = \nabla\times(\mathbf{v}\times\mathbf{B}) + \eta\nabla^2\mathbf{B} \qquad (4)$$

$$\mathbf{J} = \nabla\times(\mathbf{B}-\mathbf{B}_d) \qquad (5)$$

大規模シミュレーションに必要な並列計算を可能にする MPI (Massage Passing Interface)（[10]-[16]）を用いた並列化 Fortran ＭＨＤコードの構造を図３に示す（[8],[9]）。Modified Leap-Frog 法は、Two step Lax-Wendroff 法と同じように２ステップの計算になる。MPI を用いたＭＨＤコードにおける並列計算の特徴は、それぞれのステップの計算の直前に、分散メモリの並列計算で必要な通信をまとめてい

ることである。この一括通信を有効に利用する方法は、効率的な並列プログラムを作成するためには極めて重要な点である。

３．領域分割法による並列計算

分散メモリ型の並列計算機を用いた並列計算では、３次元配列に対して領域分割を用いるのが通常である（[8],[9]）。３次元モデルの場合、領域分割の次元を１次元、２次元、３次元に選ぶことができる。その場合の計算時間と通信時間は大まかに次の様に見積もることができる。

	計算時間	通信時間
１次元領域分割	$Ts=k_1 N^3/P$	$Tc=k_2 N^2 (P-1)$
２次元領域分割	$Ts=k_1 N^3/P$	$Tc=2k_2 N^2 (P^{1/2}-1)$
３次元領域分割	$Ts=k_1 N^3/P$	$Tc=3k_2 N^2 (P^{1/3}-1)$

ここに、k_1 と k_2 は一定の係数、N は３次元配列における１方向の変数量、P は並列 CPU の数である。計算時間と通信時間の和が並列計算に要する時間になるが、その領域分割による並列計算効率を図４に示す。計算時間 Ts は CPU 数 P に反比例して短くなるが、通信時間 Tc は P の増加に伴って長くなる。しかし、その通信時間の長くなる様子は、１、２、３次元領域分割によって大きく異なる。即ち、３次元領域分割が最も通信時間を短くできることと、１次元と２次元領域分割の間でもその差は大きいことが理解できる。ただし、この比較では通信時間を決める係数 k_2 が同じであると仮定したが、その条件も通信部分のプログラムの工夫により実現できる。こうして、スカラー並列機では３次元領域分割が、一方ベクトル並列機では、１つの次元方向はベクトル化に利用する必要があるために２次元領域分割が最も効率的であろうと予想できる。

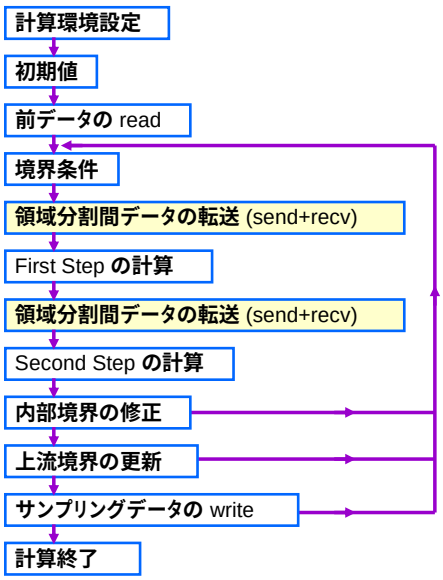

図3 MPI を用いた MHD コードの構造

３次元グローバル MHD コードにおいて３次元領域分割の具体的導入方法の要点を次に示す。MPI（[10], [14], [15]）を用いて３次元空間(x,y,z)で領域分割を行う様な以下の例の場合、x,y,z 方向にそれぞれ２分割(npex=2,npey=2,npez=2)で、全体で npe=npex*npey*npez=8 分割となり、8 個の CPU を使用する。

それぞれのCPUに入る配列変数は分割されて、f(nb,0:nxx+1,0:nyy+1,0:nzz+1)の配列が割り当てられ、nb=8はＭＨＤ方程式の成分の数である。即ち、配列変数は元々は領域分割された隣のCPUに存在する、3次元方向にそれぞれ両側のデータ１面が加わる。itable(-1:npex,-1:npey,-1:npez)はwild cardを含んだCPU間のデータ転送のための参照表である。また、ftemp1x, ftemp2xなどは、一括データ転送のために用意した２次元境界面のバッファ配列である。この様に、参照表とバッファ配列を準備することにより、３次元領域分割のために生じる分散型メモリ間のデータ転送を簡単にかつ効率的に実行できる。

<<３次元領域分割の方法>>

```
CC MPI START
       parameter (npex=2,npey=2,npez=2)
       parameter (npe=npex*npey*npez,npexy=npex*npey)
       integer itable(-1:npex,-1:npey,-1:npez)
c
       parameter(nzz=(nz2-1)/npez+1)
       parameter(nyy=(ny2-1)/npey+1)
       parameter(nxx=(nx2-1)/npex+1)
       parameter(nxx3=nxx+2,nyy3=nyy+2,nzz3=nzz+2)
c
       dimension f(nb,0:nxx+1,0:nyy+1,0:nzz+1)
       dimension ftemp1x(nb,nyy3,nzz3),ftemp2x(nb,nyy3,nzz3)
       dimension ftemp1y(nb,nxx3,nzz3),ftemp2y(nb,nxx3,nzz3)
       dimension ftemp1z(nb,nxx3,nyy3),ftemp2z(nb,nxx3,nyy3)
c
```

分散型メモリ間のデータ転送のプログラムは次に示すように極めて簡単になる。先ず、wild cardを変換表itableで使用することにより、プログラム中のx,y,z方向の端での特殊性がなくなる。これによってif文などの条件文を使う必要性がなくなる。次に、ftemp1x=f (:,is,:,:)でf (nb,0:nxx+1,0:nyy+1,0:nzz+1)のx方向の左端のyz面のデータをバッファ配列ftemp1xに一時保存して、mpi_sendrecvでileftxのCPUのftemp1xからirightxのCPUのftemp2xに一括転送する。続いて、f (:,ie+1,:,:)=ftemp2xを用いてf (nb,0:nxx+1,0:nyy+1,0:nzz+1)のx方向の右端に追加されたyz面のデータに移すことにより、データ転送を完結する。この様にして、３次元領域分割された各方向の両面のデータを分散型メモリを持つCPU間で容易に一括転送できる。プログラムで明らかなように、この方法ではifなどの条件文とdo loopが一切不要である。１次元領域分割との比較で問題となる３次元領域分割によるオーバーヘッドが、ほとんど生じないであろうことはそのことからも推測できる。結果として、図４に示すように通信時間Tcの比例係数を１、２、３次元領域分割でほとんど同じにすることができる。

<<領域分割されたCPU間のデータ転送>>

```
CC MPI START
       irightx = itable(irankx+1,iranky,irankz)
       ileftx  = itable(irankx-1,iranky,irankz)
       irighty = itable(irankx,iranky+1,irankz)
       ilefty  = itable(irankx,iranky-1,irankz)
       irightz = itable(irankx,iranky,irankz+1)
```

```
      ileftz  = itable(irankx,iranky,irankz-1)
c
      ftemp1x=f(:,is,:,:)
      ftemp1y=f(:,:,js,:)
      ftemp1z=f(:,:,:,ks)
      call mpi_sendrecv(ftemp1x,nwyz,mpi_real,ileftx,200,
     &                  ftemp2x,nwyz,mpi_real,irightx,200,
     &                  mpi_comm_world,istatus,ier)
      call mpi_sendrecv(ftemp1y,nwzx,mpi_real,ilefty,210,
     &                  ftemp2y,nwzx,mpi_real,irighty,210,
     &                  mpi_comm_world,istatus,ier)
      call mpi_sendrecv(ftemp1z,nwxy,mpi_real,ileftz,220,
     &                  ftemp2z,nwxy,mpi_real,irightz,220,
     &                  mpi_comm_world,istatus,ier)
c
      f(:,ie+1,:,:)=ftemp2x
      f(:,:,je+1,:)=ftemp2y
      f(:,:,:,ke+1)=ftemp2z
CC MPI END
```

１，２，３次元領域分割の実際の３次元 MHD コードは後で示すホームページで見ることができる（[9],[17]）。

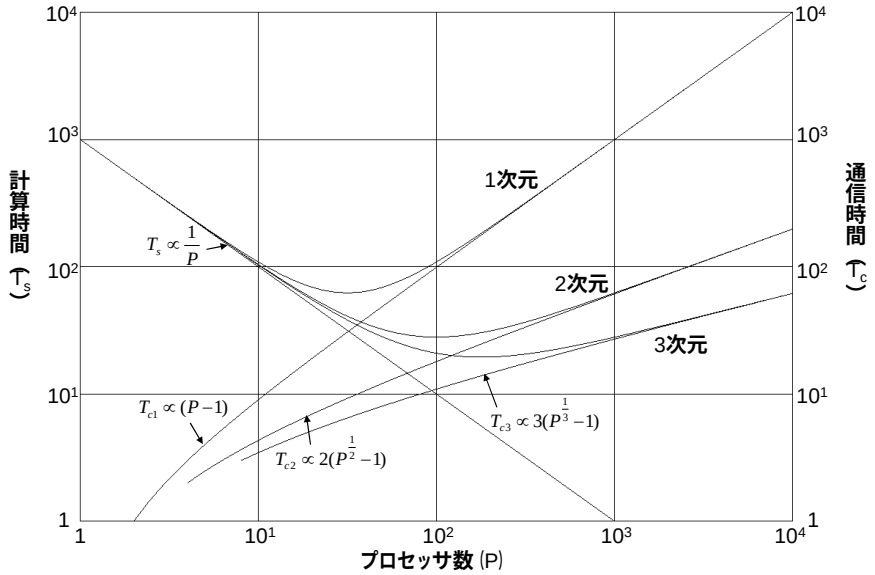

図4　1、2、3次元領域分割法による並列計算効率，並列計算時間は計算時間 (Ts)と通信時間 (Tc) の和となる。

４．領域分割法を用いたＭＨＤコードの計算効率の比較

情報連携基盤センターのスカラー並列型スーパーコンピュータ富士通 PRIMEPOWER HPC2500 で MPI を用いて書かれた３次元 MHD コードをコンパイル、実行した時のシェルを(A)と(B)に示す([10],[11],[13])。(A)は 128 CPU のプロセス並列のみの使用で、(B)は 128 CPU を用いて 32 プロセス並列と 4 スレッド並列（自動並列）（[12],[13]）を併用した場合である。また、コンパイルのオプションは最大の計算効率が得られた場合のものを示しているが、これらのオプションは今では特に指定する必要はない。今回

は使用しなかったが、スレッド間バリア機能を有効にする -Khardbarrier というコンパイルオプション（デフォルトでは OFF）を加えるのもスレッド並列での高速化には有効であるとのコメントを富士通からもらっている。

```
(A) Compile and execution of MPI Fortran program
    use 128 processors
    128 process parallel

mpifrt -Lt progmpi.f -o progmpi -Kfast_GP2=3,V9,largepage=2 -Z mpilist
qsub mpiex_0128th01.sh

hpc% more mpiex_0128th01.sh
#  @$-q p128 -lP 128 -eo -o progmpi128.out
#  @$-lM 8.0gb -lT 600:00:00
setenv  VPP_MBX_SIZE  1128000000
cd ./mearthd4/
mpiexec -n 128 -mode limited ./progmpi128

(B) Compile and execution of MPI Fortran program
    use 128 processors
    32 process parallel
     4 thread parallel (sheared memory)

mpifrt -Lt progmpi.f -o progmpi -Kfast_GP2=3,V9,largepage=2 -Kparallel -Z mpilist
qsub mpiex_0128th04.sh

mpiex_0128th04.sh
#  @$-q p128 -lp 4 -lP 32 -eo -o progmpi01.out
#  @$-lM 8.0gb -lT 600:00:00
setenv  VPP_MBX_SIZE  1128000000
cd ./mearthd4/
mpiexec -n 32 -mode limited ./progmpi
```

先ず、富士通の PRIMEPOWER HPS2500 を用いて、プロセス並列のみとプロセス並列とスレッド並列を併用した場合の３次元 MHD コード（１次元領域分割手法）の計算効率を比較した結果を表１に示す。表では、使用した総 CPU 数、プロセス並列数、スレッド並列数に対して、１回の時間ステップに要する計算時間、計算速度（GFLOPS）及び１ CPU 当たりの計算速度（GFLOPS/CPU）を示した。MPI の FORTRAN プログラムなので、プロセス並列数は２以上に設定する必要がある。また、その制約からスレッド並列数も自動的に６４以下に選ぶことになる。128 CPU の場合、プロセス並列のみの場合が最も計算効率がよいが、プロセス並列とスレッド並列の併用の場合でもそれほど遜色のない計算効率がでている。ただし、スレッド並列数が 32, 64 と大きくなるに連れて計算効率が劣化する傾向がみられる。

表１　PRIMEPOWER HPC2500 でスレッド並列を用いた場合のＭＨＤコードの計算効率

総 CPU 数	プロセス並列数	スレッド並列数	計算時間 (sec)	計算速度 (GFLOPS)	計算速度/CPU (GFLOPS/CPU)
1-dimensional decomposition by f(nx2,ny2,nz2,nb)=f(522,262,262)					
4	4	-	12.572	5.98	1.494
8	8	-	6.931	10.84	1.355
16	16	-	3.283	22.88	1.430
32	32	-	1.977	38.00	1.187
64	64	-	1.108	67.81	1.060
128	128	-	0.626	120.17	0.939
128	64	2	0.692	108.77	0.850
128	32	4	0.697	107.80	0.842
128	16	8	0.637	118.07	0.922
128	8	16	0.662	113.57	0.887
128	4	32	0.752	100.07	0.782
128	2	64	0.978	76.95	0.601
256	128	2	0.496	151.45	0.592
256	64	4	0.439	171.23	0.669
256	32	8	0.429	174.94	0.683
256	16	16	0.460	163.43	0.638
256	8	32	0.577	130.45	0.510
512	128	4	0.424	177.63	0.347
512	64	8	1.452	51.80	0.101
512	32	16	0.297	253.64	0.495
512	16	32	0.316	238.37	0.466
3-dimensional decomposition by f((nb,nx2,ny2,nz2)=f(8,522,262,262)					
512	512	1	0.0747	1007.78	1.968
512	256	2	0.0947	794.75	1.552
512	128	4	0.486	154.84	0.302
512	64	8	0.628	119.77	0.234
3-dimensional decomposition by f((nb,nx2,ny2,nz2)=f(8,1024,1024,1024)					
512	512	1	2.487	916.94	1.791
1024	512	2	1.448	1575.09	1.538
3-dimensional decomposition by f((nb,nx2,ny2,nz2)=f(8,2046,2046,2046)					
512	512	1	19.694	929.13	1.815
1024	512	2	10.763	1700.12	1.660
1024	256	4	15.648	1169.36	1.142
1536	512	3	7.947	2302.60	1.499
1536	256	6	16.462	1111.54	0.724

一般に、大きな共有メモリが必要であるとか、プロセス並列を大きくは取れない制限がある場合はスレッド並列を使用しなければならない（[12],[13]）。例えば、表１のｚ方向の１次元領域分割の３次元ＭＨＤコードの場合、外部境界条件設定の制限から、ｚ方向の配列変数（nz2=nz+2=262）の半分以下にプロセス並列数を取る必要がある。従って、CPU 数が 256 以上になる場合、必然的にスレッド並列を使うことになる。このように、プロセス並列とスレッド並列を併用する必要が生じた場合、３次元ＭＨＤコードではスレッド並列数を４～１６程度に取れば効率的な計算ができることが分かる。もちろん、どのくらいのスレッド数を用いれば高効率が得られるかは、プログラムに強く依存すると思われるが、先ずは、１６以下のあまり多くないスレッド数を試すことから始めればよさそうである。表１は、最大の並列計算速度が得られたデータのみを示しているが、スレッド数を多くする場合計算速度のばらつきがかなり現れてくる。これは、データ通信との兼ね合いとも考えられるが、実際の計算ではもう少し計算速度が遅くなることも起こりそうである。また、256CPU と 512CPU で６４スレッドの場合のデータがないが、これはワーク領域不足で実行できなかったためである。

スカラー並列機 PRIMEPOWER HPC2500 ではノード内 CPU は共有メモリとして扱えるのでそれをスレッド並列などの機能を用いて自動並列化できる。１次元領域分割の３次元ＭＨＤコードを用いた場合、表１

から CPU 数を 128 に固定してスレッド並列による自動並列化の効率をグラフに示したのが図５である。この場合、CPU 数とプロセス並列数とスレッド並列数の間で（プロセス並列数）ｘ（スレッド並列数）＝（CPU 数）の関係が必要で、スレッド並列数はノード内 CPU 数に制限される。図５では、スレッド並列数を増やした場合の計算速度（Gflops）と 1CPU 当たりの計算速度（Gflops/cpu）を示しているが、CPU 数を 128 に固定しているので両者のグラフに違いは現れない。スレッド並列数が８，１６の場合計算効率が上がっているが、スレッド並列数を３２、６４と更に増やすに連れて計算効率は劣化する。PRIMEPOWER HPC2500 で CPU 数を増やした場合の計算速度(Gflops)と 1CPU 当たりの計算速度(Gflops/cpu)の変化を図６に示す。ここで示した計算速度は一定の CPU 数に対してスレッド並列使用も含めて最大の計算速度が得られた場合の結果を示す。PRIMEPOWER HPC2500 は最大 512 のプロセス並列数がとれるが、最大の計算速度が得られたのは 512CPU まではスレッド並列を使わずプロセス並列のみのいわゆるフラット MPI 使用の場合で、かつ３次元領域分割の３次元ＭＨＤコードを用いた場合であった。CPU 数が 1024 の場合はスレッド並列数は 2、CPU 数が 1536 の場合はスレッド並列数は 3 の場合でプロセス並列数を最大の 512 とした場合が最速であった。３次元領域分割の３次元ＭＨＤコードを用いた場合 1536CPU までスケーラビィティィがかなりよく伸びていることが分かる。

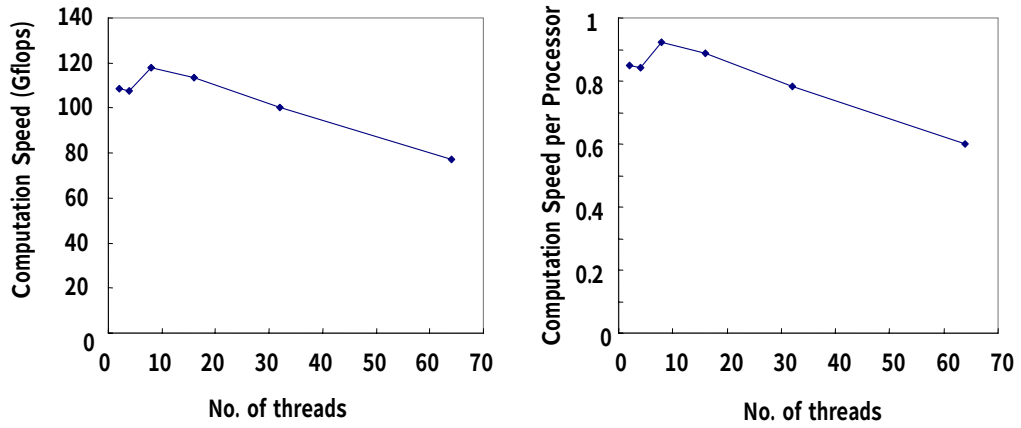

図５　PRIMEPOWER HPC2500 で CPU 数を 128 に固定してスレッド並列数を増やした場合の計算速度（Gflops）と 1CPU 当たりの計算速度（Gflops/cpu）。

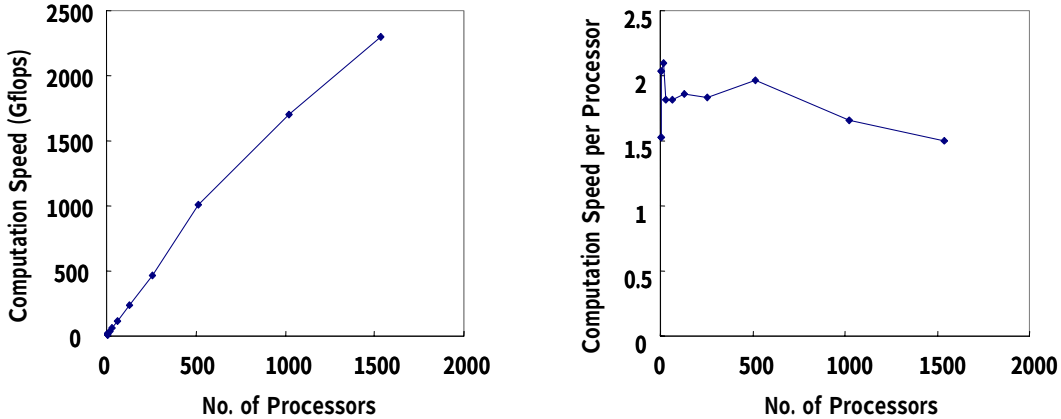

図６　PRIMEPOWER HPC2500 で CPU 数を増やした場合の計算速度（Gflops）と 1CPU 当たりの計算速度（Gflops/cpu）。

３次元領域分割の３次元ＭＨＤコードで配列の大きさが f(8,522,262,262)の場合、512CPU に固定してプロセス並列とスレッド並列数を変えた場合は面白いデータが得られた。最大計算速度は 512 プロセス並列のみの場合で、1007 GFLOPS を得た。これに対して、スレッド並列数を増やしていくと、2 スレッド並列で少し計算速度が低下し、4, 8 スレッド並列で更に大きく計算速度が低下した。特に、512 プロセス並列のみと 512CPU 使用で 64 プロセス並列と 8 スレッド並列の場合の計算速度の差は重要である。後者では x,y,z 方向とも同じ４分割なので、後者も前者とほとんど同程度の計算速度がでると期待していたが実際は違っていた。この結果は、３次元領域分割法を使って通信量を最小にしようとする方法と 8 スレッド並列内の自動並列化の方法がうまく整合しないことを意味していると考えられる。即ち、8 スレッド並列内の配列変数 f(nb,nx2,ny2,nz2)の順序付けがどの様になされているのかが、問題となりそうである。更に大きな配列の場合、f(8,1024,1024,1024)と f(8,2046,2046,2046)でも似た傾向がみられる。これらは、自動並列の最適化とユーザー並列の最適化が必ずしも両立しないことを示唆しているようにも思われる。特に、f(8,2046,2046,2046)の配列に対して、1024CPU と 1536CPU 使用でそれぞれ 4 スレッド並列と 6 スレッド並列を指定した場合、並列計算効率は上がらずにむしろ低下している。このスレッド並列数を増やした場合の計算効率劣化に対して、スレッド間バリア機能を有効にする -Khardbarrier のコンパイルオプションが有効であろうとのコメントを富士通からもらっているが確認はまだである。

１、２、３次元領域分割法を用いた場合の３次元ＭＨＤコードの計算効率の比較を表２に示す。数値は、１個当たりの CPU に対する計算速度(GF/PE)を示した。VPP5000、NEC SX6、NEC ES (Earth Simulator) はベクトル並列機であり、PRIMEPOWER HPC2500 と Hitachi SR8000, SR11000 はクラスター型のスカラー並列機である。３次元領域分割法では２種類のコードがあり、f (nx2,ny2,nz2,nb)は配列の順序が x,y,z 方向が先でベクトル成分（８個）が後であり、f (nb,nx2,ny2,nz2)は計算に関係する変数のメモリを近くにするためにベクトル成分（８個）を配列の最初に移動している。この配列順序の入れ替えは、スカラー並列機ではキャッシュのヒット率が高くなり、計算効率の改善が期待されるが、PRIMEPOWER HPC2500 ではその効果が顕著に現れている。また、512 CPU を用いたベクトル並列機の NEC ES の結果をみると x 方向でベクトル化を活かした２次元領域分割が最も効率的な結果を得ている。
スカラー並列機で高効率を実現するためには、キャッシュのヒット率を上げることと３次元領域分割を行って通信量を減らすことが重要であると提案してきたが、表２の PRIMEPOWER HPC2500 の 512 cpu までの結果からそのことの重要性を明瞭にみることができる。PRIMEPOWER HPC2500 での２、３次元領域分割の計算結果は 512cpu まですべてプロセス並列のみを利用したものである。１次元領域分割の場合、HPC2500 で非常に悪い結果が出ているが、これは３次元ＭＨＤコードで境界を含めたグリッド点数 (nx,ny,nz)=(510,254,254)を 2 のべき乗（512=510+2, 256=254+2）に取ったためで、グリッド点数を少し変える(nx,ny,nz)=(520,260,260) と括弧で示す数値の様に効率が大きく改善された。即ち、領域分割方向の配列の大きさを２のべき乗にとり、CPU 数も２のべき乗にとった場合、メモリアクセスの競合が生じて計算速度が極端に劣化する現象が発生したと考えられる（[12],[13]）。従って、これまでのほとんどのＭＨＤコードは、最初に高効率が得られない場合でも CPU 数や配列の大きさを少し変えるなどの変更さえすれば大きな問題無く高速に計算できることが分かった。

表2 3次元MHDコードによる計算効率の比較

Computer Processing Capability by 3D MHD Code for (nx,ny,nz)=(510,254,254)

	CPU Number	VPP5000 (GF/PE)	PRIMEPOWER HPC2500 (1.3GHz)	PRIMEPOWER HPC2500 (2.08GHz)	NEC SX6 (GF/PE)	NEC ES (GF/PE)	Hitachi SR8000 (GF/PE)	Hitachi SR11000/j1 (GF/PE)
1D Domain	2	7.08	-----	-----	6.36	6.66		
Decomposition by	4	7.02	0.031	0.039(1.494)	5.83	6.60		0.182
f(nx2,ny2,nz2,nb)	8	6.45	0.030	0.037(1.355)	5.51	6.50	0.016	
	16	6.18	0.028	0.046(1.430)	5.44	6.49		
	32	(7.49)		0.042(1.187)		6.39		
	64	(6.90)		0.040(1.060)		6.37		
	128			0.039(0.907)		(2.11)		
	256			0.016(0.683, 2 thread)				
	512			0.003(0.347, 4 thread)				
2D Domain	4	7.51	0.199	1.529	6.34	6.63		0.775
Decomposition by	8	6.88	0.191	1.451	6.28	6.47		
f(nx2,ny2,nz2,nb)	16	6.49	0.200	1.575	6.23	6.45		
	32			1.395		6.47		
	64			1.421		6.32		
	128			1.409		6.27		
	256			1.396		6.05		
	512			0.868		5.62		
f(2048,1024,1024,8)	1024	(MPI)				7.18(7.36 TF)		
f(1024,1024,1024,8)	512	(HPF/JA)				6.47(3.31 TF)		
3D Domain	8	7.14	0.207	1.558	6.24	6.38	0.253	0.869
Decomposition by	16	6.77	0.202	1.593	6.34	6.33		
f(nx2,ny2,nz2,nb)	32			1.527		6.25		
	64			1.534		5.61		
	128			1.518		5.57		
	256			1.513		5.38		
	512			0.923		3.94		
3D Domain	8	2.91	1.438	2.038	1.13	4.11	0.268	2.221
Decomposition by	16	2.63	1.416	2.099	4.53	4.06		
f(nb,nx2,ny2,nz2)	32			1.820		4.11		
	64			1.813		4.17		
	128			1.857		4.12		
	256			1.831		4.12		
	512			1.968		3.70		
f(8,1024,1024,1024)	512	(MPI)		1.791(0.917 TF)				
f(8,1024,1024,1024)	1024	(MPI, 2 thread)		1.538(1.575 TF)				
f(8,2048,2048,2048)	512	(MPI)		1.815(0.929 TF)				
f(8,2048,2048,2048)	1024	(MPI, 2 thread)		1.660(1.700 TF)				
f(8,2048,2048,2048)	1024	(MPI, 4 thread)		1.142(1.169 TF)				
f(8,2048,2048,2048)	1536	(MPI, 3 thread)		1.499(2.303 TF)				
f(8,2048,2048,2048)	1536	(MPI, 6 thread)		0.724(1.112 TF)				

従来の配列f(nx2,ny2,nz2,nb)を用いた１次元（1D）、２次元（2D）、３次元領域分割の３次元ＭＨＤコード(3Da)及びf(nb,nx2,ny2,nz2)と配列の順序を換えた３次元領域分割の３次元ＭＨＤコード(3Db)を用いて、CPU数を増やした場合どのような計算速度変化の傾向が見られるかを、スカラー並列機PRIMEPOWER HPC2500に対して図７に、ベクトル並列機ES（Earth Simulator）対して図８に示す。ただし、配列の大きさは１次元領域分割を除いて（nx,ny,nz）＝（510,254,254）を用い、１次元領域分割の場合のみ表２で説明したように境界を含めた配列の大きさが２のべき乗になることからの極端な劣化を避けるために（nx,ny,nz）＝（520,260,260）を用いた。その配列の大きさの違いは換算しているので図７でも４種類のＭＨＤコード間の計算速度（Gflops）と1CPU当たりの計算速度（Gflops/cpu）

を直接に比較することができる。まず、図７のスカラー並列機 PRIMEPOWER HPC2500 の場合、1D より 2D が計算速度が速くなっていることから１次元から２次元領域分割に換えたことよって通信量を減らした効果をみることができる。しかし、2D と 3Da の間では顕著な計算速度の変化はみられない。これはキャッシュのヒット率が同じように良くないためであると考えられる。ここで、キャッシュのヒット率を改善するために配列の順序を換えた３次元領域分割（3Db）では計算速度が CPU 数 512 まで大きく改善されて、スケーラビリティが非常に良くなっているのがみられる。このようにスカラー並列機ではキャッシュのヒット率を向上するために配列の順序を工夫して、かつ３次元領域分割法を利用することで非常に良い並列計算効率を得られることが 512CPU まで確認された。

一方、図８のベクトル並列機 Earth Simulator では予想通り２次元領域分割（2D）がスケーラビリティの良さは 512CPU まで最もよく保たれている。これは、最内部の do loop はベクトル化に用いるので、そのベクトル長は長い方がベクトル化効率が良くなるためと、同時に２次元領域分割を用いて通信量を減らしたために最高の計算速度が得られたと考えられる。これは、表２に示すように配列を大きくして CPU 数を 1024 と増やした場合より顕著に見られる。また、ベクトル長よるベクトル化効率の違いは 3Da と 3Db の差、即ち 3Da が 3Db よりも計算速度が速い結果になって現れている。CPU 数が 512 に大きくなると 3Da の計算速度は遅くなって 3Db の計算速度近づいているが、最内部の do loop 長が 512/8=64 と短くなったためと考えられる。Earth Simulator の場合、むしろ 3Da と 3Db の差が小さい、さらに 2D と 3Db の差も大きくはないことに驚くべきであると思われる。一般的にはもっと差が出ると予想していたが、Earth Simulator では CPU 間のデータ転送が高速で非常に優れているために３次元領域分割（3Da と 3Db 共）でも並列計算速度があまり落ちないと理解できる。また、図８の 256CPU 使用の１次元領域分割で計算速度が劣化しているが、領域分割方向の変数が境界を含めて nz+2=256 となり並列化ノード数はその半分以下に設定すべき必要性から、その場合のみ自動並列を利用したためであると考えられる。この点は更なる調査が必要である。これらの結果からベクトル並列機では２次元領域分割法を利用することで非常に良いベクトル並列計算効率を得られることが確認された。

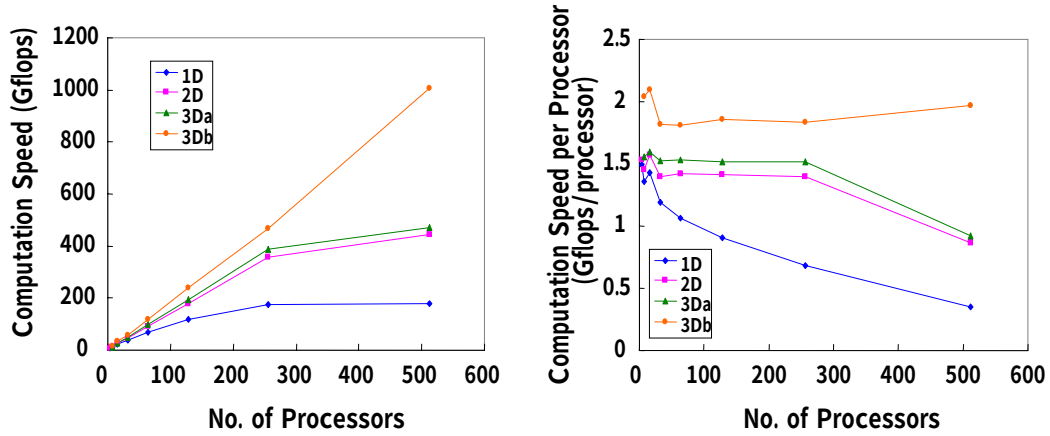

図７　スカラー並列機 PRIMEPOWER HPC2500 で CPU 数を増やした場合の４種類のＭＨＤコード（1D：１次元領域分割、2D：２次元領域分割、3Da：３次元領域分割で f(nx2,ny2,nz2,nb)、3Db：３次元領域分割と f(nb,nx2,ny2,nz2)）に対する計算速度（Gflops）と 1CPU 当たりの計算速度（Gflops/cpu）。

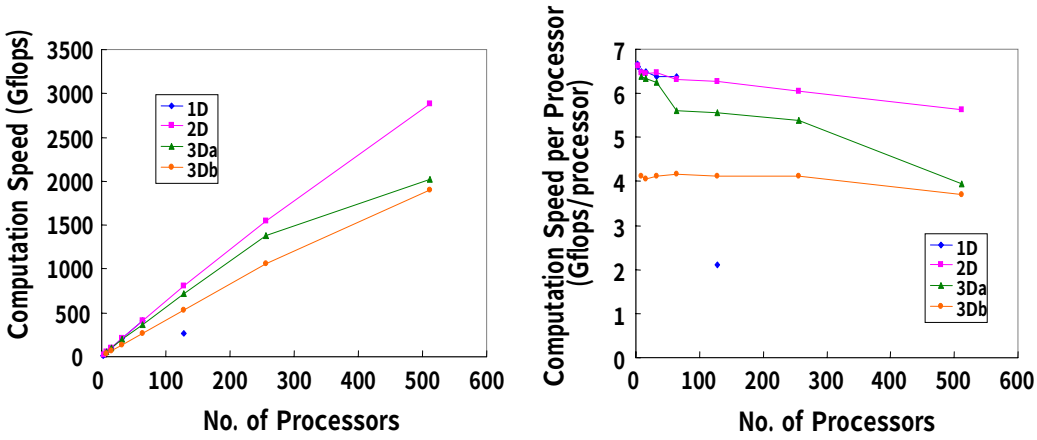

図８　ベクトル並列機（Earth Simulator）で CPU 数を増やした場合の４種類のMHDコード（1D：１次元領域分割、2D：２次元領域分割、3Da：３次元領域分割で f(nx2,ny2,nz2,nb)、3Db：３次元領域分割と f(nb,nx2,ny2,nz2)）に対する計算速度（Gflops）と 1CPU 当たりの計算速度（Gflops/cpu）。

残された問題点として、１次元領域分割では 128 並列までがプログラム上可能で、256cpu を使う場合 128 プロセス並列と 2 スレッド並列の組み合わせ、512cpu を使う場合 128 プロセス並列と 4 スレッド並列の組み合わせを使う必要があった。これらの場合(nx,ny,nz)=(520,260,260)のグリッドの場合でも、データのハード転送機能をオフ（-ldt=0）にした場合は計算速度はかなり低下して、更に 512cpu では極端に低下する結果が得られた。しかし、そのデータトランスファーユニットをオン（-ldt=1）にすれば、その計算効率の低下傾向は大きく改善された。また、名古屋大学の PRIMEPOWER HPC2500 では最大 512 プロセス並列までに制限されているので、1024cpu 使用では 512 プロセス並列と 2 スレッド並列の組み合わせ、1536cpu 使用では 512 プロセス並列と 3 スレッド並列の組み合わせを用いることになる。しかし、これらの多くの cpu を使用してのプロセス並列とスレッド並列の組み合わせでは、データのハード転送機能オフ（-ldt=0）のテストでは並列化の効率は良くならないばかりか、逆に非常に悪くなっていた。しかし、その機能をオン（-ldt=1）にすれば、表１、２の３次元領域分割法を用いた場合の f(8,1024,1024,1024)と f(8,2046,2046,2046)の結果から分かるように、計算効率は著しく改善された。この様に、ハードデータ転送機能を使い、かつ３次元領域分割法を用いればスケーラビリティティが 1536cpu 使用まで非常によく伸びることが確認できた。もう一点、気が付くことは、多数の CPU を使用する場合、プロセス並列数を減らしてスレッド並列数を増やす方法は計算効率の劣化を招くことである。この傾向は表１の３次元領域分割法を用いた場合の f(8,522,262,262), f(8,1024,1024,1024), f(8,2046,2046,2046)の結果に顕著に現れている。これら現在までの結果は、それぞれ最適のユーザー並列と自動並列を組み合わせる方法より、ユーザー並列のみを利用する方法（例えばフラット MPI）の方が、高効率の並列計算を実現できるという考え方を指示している様にも思われる。VPP5000 の括弧内の数値は、32-64PE でこれまでどのくらいの計算速度の最高値が出ていたかを比較のために示したものである。

実際、VPP5000 の 16cpu で通常動かしていた３次元MHDコードは 110-120 GFLOPS の速度が出ていたが、同じコードを PRIMEPOWER HPC2500 で動かすと 64cpu で 80-90 GFLOPS、128 cpu で 90-140 GFLOPS が出

ていて、これまでの計算はそのまま継続できる。勿論、VPP5000 で高効率を実現していた３次元ＭＨＤコードは cpu 数が更に増えると計算速度が飽和する傾向が見られる。プロセス並列とスレッド並列を併用して、計算速度の変化を調べると、スレッド並列を併用する場合６ー７％計算速度が向上する場合もあるが、前述したようにデータトランスファーユニットを利用しない場合はスレッド並列併用で計算効率が向上するのはcpuがあまり多くない場合(128～256cpu以下)に限られる傾向が認められる。しかし、データのハード転送機能をオンにしておけば、その cpu 数の上限はかなり上昇する。こうして、プロセス並列とスレッド並列の併用でもかなり高い並列計算効率の達成を期待することができる。

５．2005 年更新のスーパーコンピュータ PRIMEPOWER HPC2500 の概要と高速計算への予想と実測

平成１６年度に更新された名古屋大学情報連携基盤センターの計算機の概要を次に示す。従来のベクトル並列機 Fujitsu VPP5000/64 からスカラー並列機 Fujitsu PRIMEPOWER HPC2500 に２００５年３月に更新された（[10],[11]）。

＜スーパーコンピュータ＞

Fujitsu PRIMEPOWER HPC2500 23 ノード

（64 CPU／メモリ 512 GB × 22 ノード、128 CPU／メモリ 512 GB × 1 ノード）

総合性能 ：12.48 TFLOPS

総メモリ容量 ：11.5 TB

ディスク容量 ：50 TB

更新によって、どれだけ高性能になるかを計算速度、主メモリ、磁気ディスク容量およびネットワーク速度のスーパーコンピューティング４機能について、VPP5000/64 と PRIMEPOWER HPC2500 のカタログ性能比として図 9 に示す。ベクトル並列機 VPP5000/64 からスカラー並列機 PRIMEPOWER HPC2500 への更新によって、理論上の性能は大幅によくなるが、スカラー並列機の相対的な非効率性があるので性能向上を安易に期待するのは危険である。更新後の PRIMEPOWER HPC2500 でどこまで高速計算が期待できるかを従来のテストを基に予想すると次の様になった。その後、テスト計算を実施したので、その実測結果も表３に加えた。

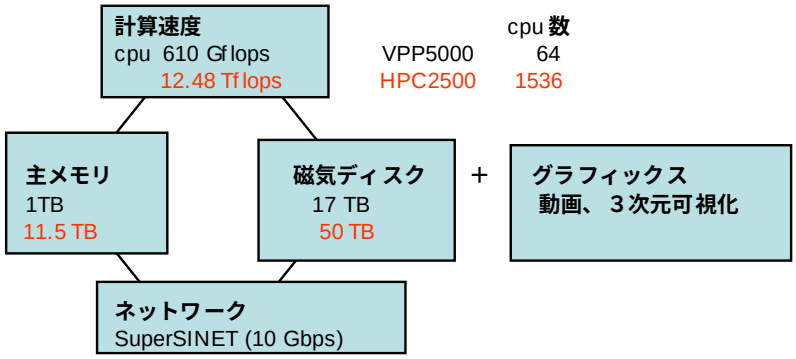

図 9 PRIMEPOWER HPC2500/1536 と VPP5000/64 の比較（名古屋大学情報連携基盤センター）とスーパーコンピューティングの4つの重要な機能

表３ PRIMEPOWER HPC2500 でどこまで高速計算が期待できるかとその実測値

```
------------------------------------------------------------------------
VPP5000/64
        理論性能：9.6 GF x 64  =              614 GF
        実測                                  410 GF (66.8%)

PRIMEPOWER HPC2500/1536
       理論性能：8.125 GF x 1536  =  12,480 GF
        期待値                         3,400 GF (27.2%)
                                     GF/cpu    cpu 数      計算速度    効率
        ３次元領域分割    1.416 x (8.125/5.2) x  1536 =    3,400 GF ( 27.2%)
        ２次元領域分割    0.202 x (8.125/5.2) x  1536 =      485 GF (  3.9%)
        １次元領域分割    0.028 x (8.125/5.2) x  1536 =       67 GF (  0.5%)

        実測
                                     GF/cpu    cpu 数      計算速度    効率
        ３次元領域分割                1.831  x   256 =      469 GF ( 22.5%)
                                      1.968  x   512 =     1008 GF ( 24.2%)
                                      1.660  x  1024 =     1700 GF ( 20.4%)
                                      1.499  x  1536 =     2303 GF ( 18.5%)
        ２次元領域分割                1.396  x   256 =      357 GF ( 17.1%)
        １次元領域分割                0.907  x   128 =      116 GF ( 11.1%)
------------------------------------------------------------------------
```

更新当時の危惧をまとめて書くと（[8],[9]）、特に、ベクトル並列機で高効率を実現していた１次元領域分割のＭＨＤコードでの劣化は悲惨であるかも知れないし、計算効率が VPP5000/64 の 10 分の１に落ちる可能性があるかも知れない。もし、この様な状況になるとすれば、従来の３次元 MHD コードは全く使いものにならないことになる。２、３次元領域分割をうまく採用してやっと同程度の計算効率が期待できそうである。配列変数の並び替えを行った３次元領域分割のＭＨＤコードを用いた場合、最も楽観的な予想で 3.4 TFLOPS が期待できる可能性がある。しかし、絶対効率でどれだけよくなるかは、実際に３次元ＭＨＤコードで実行することが肝要である。

上記の様に、スカラー並列機 PRIMEPOWER HPC2500 への更新に多くの懸念材料があったが、２００５年３月から実際に稼働開始したので、精力的にテストした結果が表１、２であり、実測としてまとめたものが表３である。これらの結果から、従来ベクトル並列機で高効率を発揮していた３次元ＭＨＤコードも、ほとんど変更無しでそれなりの高速化を実現できることが明らかになった。実際１次元領域分割法を用いた従来の３次元 MHD コードでも、128cpu で 116GF、256cpu で 175GF の計算速度を実現している。こうして、スペースプラズマの大規模 3 次元 MHD シミュレーションは、情報連携基盤センターのベクトル並列機 VPP5000 からスカラー並列機 PRIMEPOWER HPC2500 への移行に際しても問題なくスムーズに維持することができた。また、(nx2,ny2,nz2) =(2048,2048,2048) のグリッドを用いた３次元領域分割法の場合、512cpu で 929GF、1024cpu で 1700GF、さらに 1536cpu で 2303GF の最高速値を得ることができた。前述のスレッド間バリア機能を有効にするコンパイルオプションを使用すれば、さらなる高速化が期待できそうである。

６．まとめ

太陽地球環境研究所の計算機利用共同研究で使用している名古屋大学情報連携基盤センターのスーパーコンピュータも２００５年３月にベクトル並列機 Fujitsu VPP5000/64 からスカラー並列機 Fujitsu PRIMEPOWER HPC2500 へ更新された。ベクトル計算機は先ずベクトル化効率を上げることが重要だったの

で最内の do ループの変数（例えばｘ方向）をできるだけ長く取ってきた。こうして、１、２、３次元領域分割や MPI や HPF（High Performance Fortran）（[6],[8]）などいずれの方法でも、比較的簡単に高効率の並列プログラムを作成することに成功してきた。しかし、スカラー並列機では、キャッシュのヒット率が最も重要なので、最内の do ループの変数は空間一方向などの長い変数に取るのは不適切で、例えば計算に直接関与するＭＨＤ方程式の８個のベクトル成分を取るのが望ましい。こうして、特に CPU が増えた場合、ベクトル並列機で高効率を実現していた多くのプログラムは、スカラー並列機で全く高効率を上げることができない問題に直面するのではないかと懸念していた。しかし、従来ベクトル並列機で高効率を発揮していた３次元ＭＨＤコードも、PRIMEPOWER HPC2500 でほとんど変更無しでそれなりの高速化を実現できることが明確に分かってきた。更に、キャッシュのヒット率の向上と３次元領域分割を用いた３次元ＭＨＤコード（3D Domain Decomposition by f (nb,nx2,ny2,nz2)）では、MPI によるプロセス並列のみで最高速値(512cpu で 929GF)が得られていて、かつスケーラビリティも非常に良いことが確認できた。ただし、これらの高効率を得るためにはデータのハード転送機能を利用することが不可欠である。

富士通 PRIMEPOWER HPC2500 の現在のシステムではプロセス並列は 512 が上限なので、それ以上の cpu を使用する場合、プロセス並列とスレッド並列の併用が必要である（[10]-[13]）。その初期のデータのハード転送機能オフのテスト結果では 512cpu を越えてのプロセス並列とスレッド並列の組み合わせで高速化を実現することはできなかった。しかし、データのハード転送機能オンにすれば、その 512cpu を越えての並列計算効率は著しく改善された。こうして、計算のスケールを大きくした f(8,2046,2046,2046)の３次元領域分割を用いた３次元ＭＨＤコードでは、1024cpu を用いたプロセス並列（512）とスレッド並列（2）の併用で 1.7 TFLOPS、さらに 1536cpu で 2.3 TFLOPS の最高速を得ることができた。この様にプロセス並列とスレッド並列をうまく併用すれば、かなりの並列計算効率が実際に得られることが分かった。ただし、この様な多数の CPU を使用する場合、スレッド数を増やすと並列計算効率が逆に低下する傾向がある。これは、並列計算機で多数の CPU を使う場合、全てユーザー並列（いわゆる MPI でのフラット並列など）で実行するのが高効率を得られるか、ユーザー並列と自動並列の最適の組み合わせで実行するのが高効率を得られるかという、最も基本的な並列計算機利用の問いにも直接関係している。即ち、最適化されたノード内自動並列を利用し、利用者が最適のノード間並列を実現すれば、最大の並列化効率が得られるはずである、という未確認だが多くの人に信じられている神話の信憑性を問う必要がある。それに対する反論は、ユーザー並列と自動並列の最適と言われる組み合わせで、最大の並列化効率が得られたという実例がないというものである。これらの命題は今後も論争が続くと思われるが、今回の並列計算ではユーザー並列のみの方が高効率の並列計算を実現している。富士通はプロセス並列数の上限を 1024CPU 以上に上げる計画であると言っているので、これは利用者にとって大変嬉しい話である。

今後益々主流となってきそうなスカラー並列機での高効率な計算を実現する場合、キャッシュのヒット率を上げるために配列の順序を入れ替えること、３次元領域分割法を導入すること、およびその通信方法をまとめて一括方式で送受信することが一層必要となる。そのような条件を満たす新しい３次元ＭＨＤコードの開発導入により、大規模スカラー並列機の効率的利用の可能性に初めて道が開けてくると思われる。もちろん、高速なハード転送機能の必要性は言うまでもないことである。太陽風と地球磁気圏

相互作用を解く３次元グローバルＭＨＤコードで実現していた、ベクトル並列機の場合の様な絶対効率で４０ー７０％の高効率計算は無理としても、今回の富士通 PRIMEPOWER HPC2500 を用いたテスト計算からスカラー並列機で２０％を超える、あるいは２０％に近い計算効率が実現できることも確実に期待できそうである。日本のスーパーコンピュータメーカーの優れた技術力と更なる開発力に大いに期待するところである。

謝辞

本稿のコンピュータシミュレーションは名古屋大学情報連携基盤センターのスーパーコンピュータ、Fujitsu VPP5000/64 と PRIMEPOWER HPC2500、JAXA/ISAS の NEC SX6、地球シミュレータセンターの Earth Simulator(ES)、及び国立極地研究所の日立 SR8000 と SR11000/j1 を利用してなされたものです。また、MPI バージョンの３次元 MHD コードの開発とテストでは多くの助言を頂いた名古屋大学情報連携基盤センターの津田知子助手と富士通株式会社の方に、更に、日立のスーパーコンピュータ利用で協力して頂いた国立極地研究所の岡田雅樹助手に感謝いたします。

[参考文献]

[1] T. Ogino, A three-dimensional MHD simulation of the interaction of the solar wind with the earth's magnetosphere: The generation of field-aligned currents, J. Geophys. Res., 91, 6791-6806 (1986).

[2] T. Ogino, R.J. Walker and M. Ashour-Abdalla, A global magnetohydrodynamic simulation of the magnetosheath and magnetopause when the interplanetary magnetic field is northward, IEEE Transactions on Plasma Science, Vol.20, No.6, 817-828 (1992).

[3] T. Ogino, Two-Dimensional MHD Code, (in Computer Space Plasma Physics), Ed. by H. Matsumoto and Y. Omura, Terra Scientific Publishing Company, 161-215, 411-467 (1993).

[4] T. Ogino, R.J. Walker and M. Ashour-Abdalla, A global magnetohydrodynamic simulation of the response of the magnetosphere to a northward turning of the interplanetary magnetic field, J. Geophys. Res., Vol.99, No.A6, 11,027-11,042 (1994).

[5] 荻野竜樹、「太陽風と磁気圏相互作用の電磁流体力学的シミュレーション」, プラズマ・核融合学会誌, CD-ROM 特別企画（解説論文）, Vol.75, No.5, CD-ROM 20-30, 1999. http://gedas.stelab.nagoya-u.ac.jp/simulation/mhd3d01/mhd3d.html

[6] 荻野竜樹、「太陽風磁気圏相互作用の計算機シミュレーション」, 名古屋大学大型計算機センターニュース, Vol.28, No.4, 280-291, 1997.

[7] 荻野竜樹、「コンピュータシミュレーションと可視化」, 愛媛大学総合情報処理センター広報, Vol.6, 4-15, 1999. http://gedas.stelab.nagoya-u.ac.jp/simulation/simua/ehime985.html

[8] Ogino, T., Global MHD Simulation Code for the Earth's Magnetosphere Using HPF/JA,

Special Issues of Concurrency: Practice and Experience, 14, 631-646, 2002.
http://gedas.stelab.nagoya-u.ac.jp/simulation/hpfja/mhd00.html
[9] 荻野竜樹、「並列３次元 MHD コードと３次元可視化」,
天体とスペースプラズマのシミュレーションサマーセミナー講義テキスト、107-136、2003.
http://center.stelab.nagoya-u.ac.jp/kaken/mpi/mpi-j.html
[10] 津田知子、「MPI による並列化」、名古屋大学情報連携基盤センター　MPI 講習会資料、
2002 年 2 月
[11] 永井亨、津田知子、「新スーパーコンピュータシステムの概要と利用方法について」、
名古屋大学情報連携基盤センターニュース、Vol.4, No.1, 6-14、2005.
[12] 青木正樹「ベクトル（VPP5000）からスカラ SMP（PRIMEPOWER HPC2500）へ」、
名古屋大学情報連携基盤センターニュース、Vol.4, No.1, 20-27、2005.
[13] 津田知子、「新スーパーコンピュータ（HPC2500）利用のしおり」、
名古屋大学情報連携基盤センターニュース、Vol.4, No.2, 104-113、2005.
[14] 富士通株式会社、「MPI プログラミング　～Fortran, C～」第 1.3 版、2001 年 4 月、
第 1.4 版、2002 年 12 月.
[15] 富士通株式会社、「MPI 使用手引き書　V20 用」、UXP/V 初版　1999 年 9 月
[16] 青山幸也（日本アイ・ビーエム株式会社)、「並列プログラミング虎の巻　MPI 版」
虎の巻シリーズ２、2001 年１月.
[17] 本稿で使用した１、２、３領域分割の３次元ＭＨＤコード参照のホームページ、
Homepage of New Domain Decomposition 3D MPI MHD Programs,
http://center.stelab.nagoya-u.ac.jp/kaken/mpi/mpiex012.html