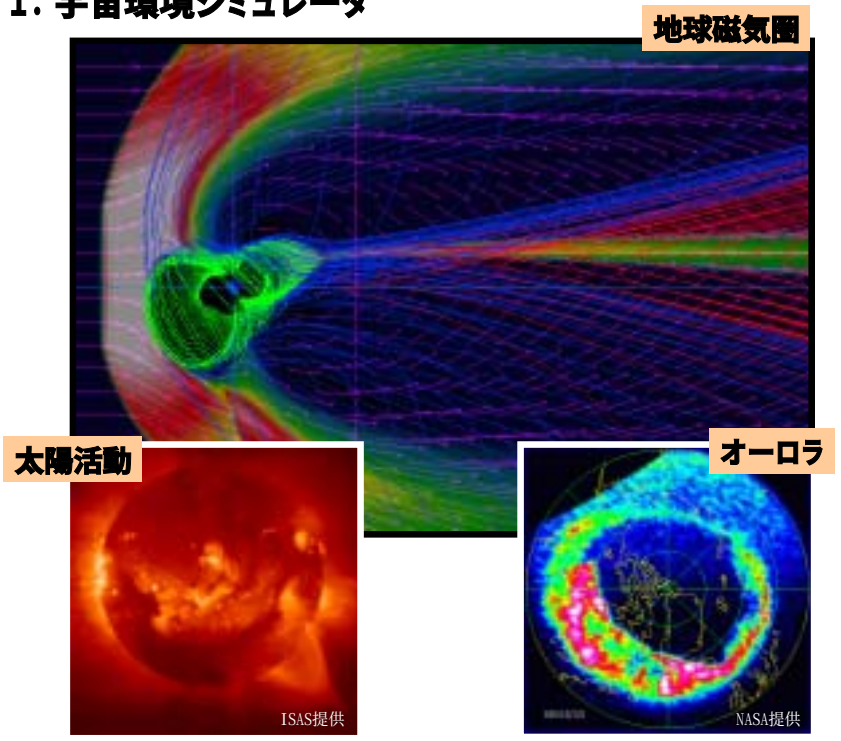
宇宙環境シミュレータのMHDコードにおけるHPFの効能

荻野竜樹 (名古屋大学太陽地球環境研究所)
寺田直樹 (名古屋大学太陽地球環境研究所)
大村善治 (京都大学宙空電波科学研究センター)

1. 宇宙環境シミュレータ
2. 地球磁気圏のMHDシミュレーション
3. MHDコードにおけるHPF/JA (High Performance Fortran)
4. MHDコードにおけるMPI (Message Passing Interface)
5. MHDコードにおけるHPFとMPIの比較
6. まとめ

1. 宇宙環境シミュレータ



研究目的

先進的宇宙環境利用、宇宙開発のための
宇宙プラズマ・飛翔体環境の定量データ取得、
およびその宇宙開発へのフィードバック

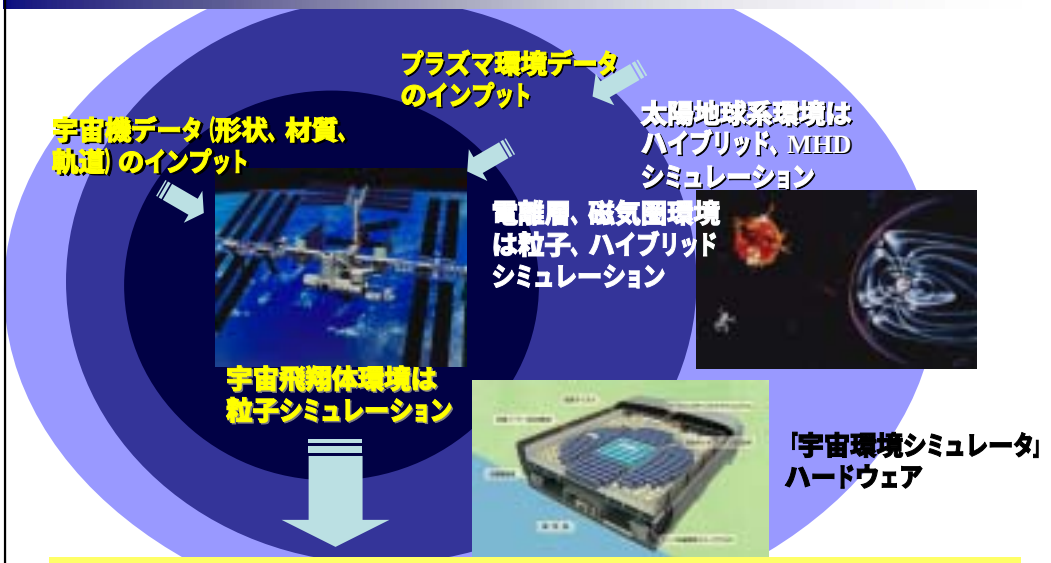
宇宙環境利用、宇宙開発のための宇宙プラズマシミュレーションへ
（Space Development への応用）

数値チェンバーの開発と
それを用いた実用的計算機実験
（地球シミュレータの利用）

質的転換

太陽-地球環境における様々なプラズマ現象の観測、
シミュレーション解析
（Space Science のための計算機シミュレーション）

宇宙環境シミュレータ　概念図



宇宙電磁環境や宇宙機との相互作用に関する基礎データを出力

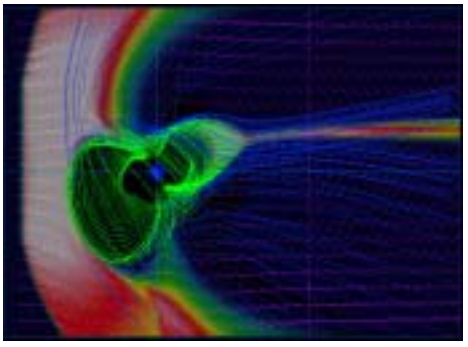
- 宇宙機への帯電、放電の可能性は？
- 電磁波の干渉は？通信・誘導システムへの影響は？
- 粒子加熱加速、船外活動への影響は？

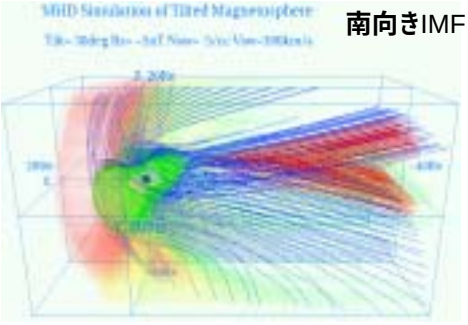
2. 地球磁気圏のMHDシミュレーション

太陽風と地球磁気圏の相互作用を再現するために、
modified leap-frog 法を利用して、
MHD方程式とMaxwell方程式を初期値境界値問題として解く。

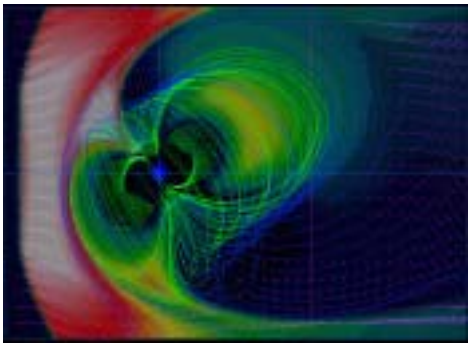
- 3次元MHD (電磁流体力学) コード
- デカルト座標(x,y,z)で一様格子
- (nx,ny,nz)=(500,318,318)と(nx2,ny2,nz2)=(502,320,320)
- z方向に1次元の領域分割

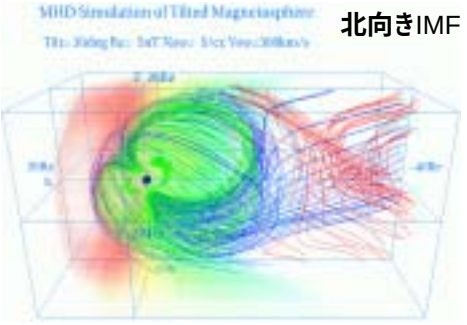
MHD Simulation of the Earth's Magnetosphere with Dipole Tilt for IMF Bz Component





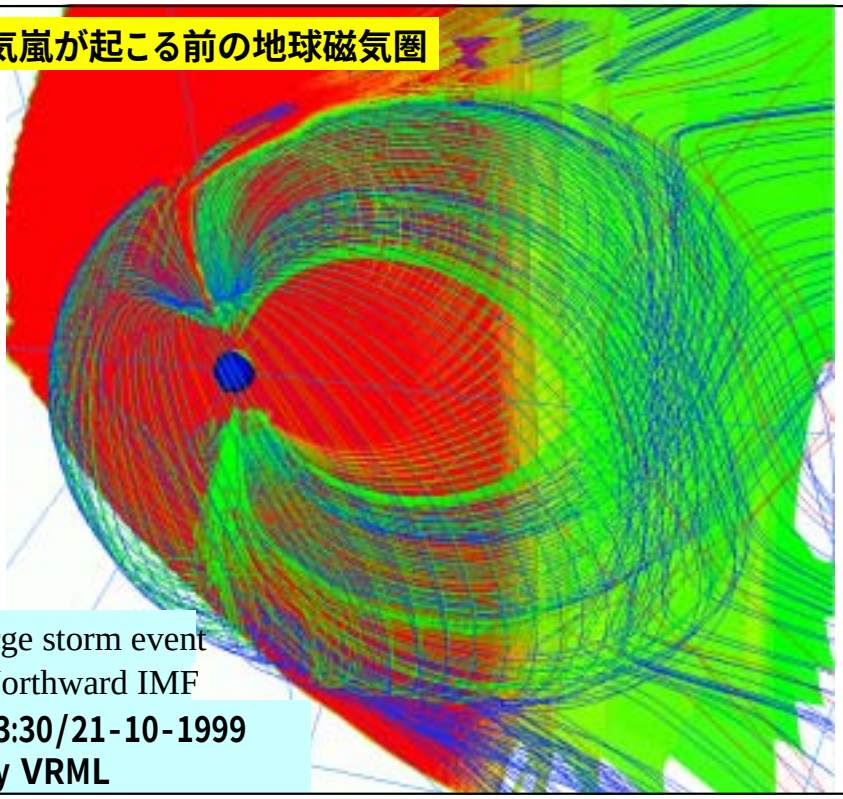
南向きIMF





北向きIMF

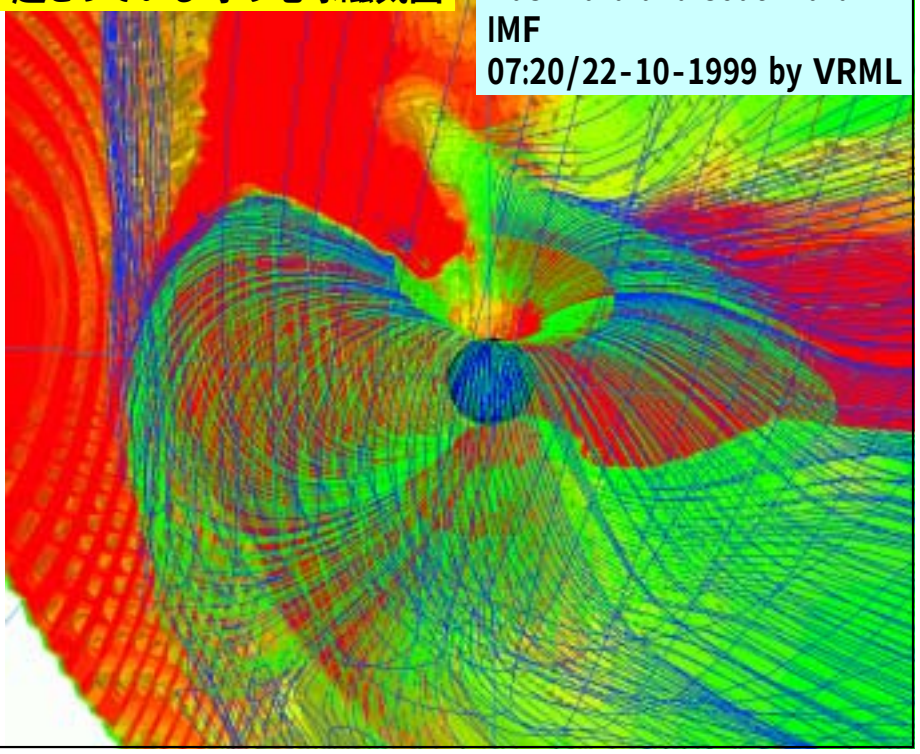
磁気嵐が起こる前の地球磁気圏



Large storm event
Northward IMF
23:30/21-10-1999
by VRML

磁気嵐が起こっている時の地球磁気圏

Duskward and southward
IMF
07:20/22-10-1999 by VRML



MHDコード

3次元MHDコードの現状

・modified leap-frog 法でMHD方程式を初期値境界値問題として解く
・z方向の領域分割を用いてHPF/JAとMPIで並列化
・境界を含む格子点数 (nx2,ny2,nz2)=(502,320,320)
・一様格子点間隔 dx=dy=dz=0.3 Re

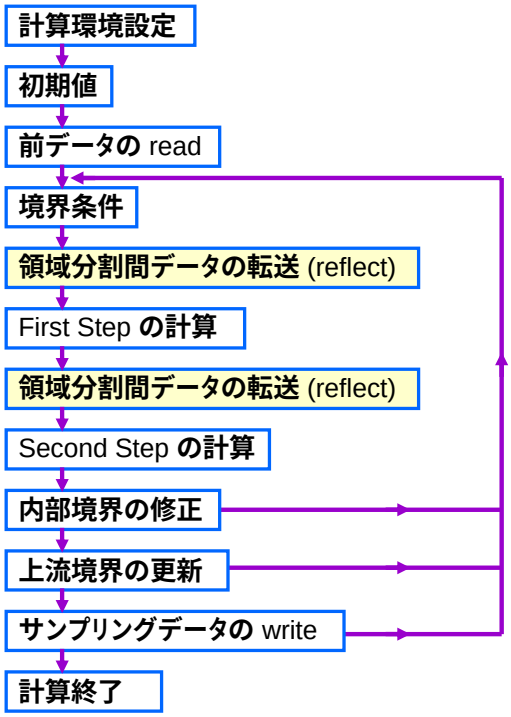
ESでのMHDシミュレーションの目標

(1)外部磁気圏と内部磁気圏統合の高精度MHDシミュレーション
・時空間マルチスケール法
(2) 磁気圏電離圏結合の3次元拡張MHDシミュレーション
・非一様グリッド利用の modified leap-frog 法
(3) 宇宙ステーション・SPSを取り囲むグローバルな宇宙環境を定量評価

3次元MHDコードのESでのテストの問題点

・HPF/JAとMPIでのベクトル化と並列化に無駄がないか再度調査？
・processor数が少ない時、MPIはHPF/JAより効率が高い理由？
・MPIでprocessor数を増加するとき、並列化の効率が下がる理由？

3．MHDコードにおけるHPF/JA



Translation the MHD codes from VPP Fortran to HPF/JA

(1)We can use the same domain decomposition (z-direction in 3 dimension) as in VPP Fortran, and overlap data at the boundary of distributed data which was handled by "sleeve" in VPP Fortran can be replaced by "shadow" and "reflect" in HPF/JA.
(2) Parallelizing formalities for decomposition are performed by "independent" sentence as is done in VPP Fortran.
(3) Lump transmission of distributed data can be done by "asynchronous“ sentence.
(4) ”Asynchronous" sentence can be used for lump transmission between the distributed data with different decomposition directions.
(5) Unnecessary communication, which was problem in HPF, can be removed by instructions of "independent new ( )" and "on home local ( )".
(6) Maximum and minimum can be calculated in parallel by "reduction".

hpf/ja (High Performance Fortran)

```
c
      dimension f(nx2,ny2,nz2,nb),u(nx2,ny2,nz2,nb),
     1          ff(nx2,ny2,nz2,nb),p(nx2,ny2,nz2,nbb),
     2          pp(nx2,ny2,nz2,3),fdd(mfd,nfd)

c
!hpf$  distribute f(*,*,block,*) onto pe
!hpf$  distribute u(*,*,block,*) onto pe
!hpf$  distribute pp(*,*,block,*) onto pe
!hpf$  distribute ff(*,*,block,*) onto pe
!hpf$  distribute p(*,*,block,*) onto pe
!hpf$  shadow     f(0,0,1:1,0)
!hpf$  shadow     u(0,0,1:1,0)
!hpf$  shadow     pp(0,0,1:1,0)
!hpf$  shadow     ff(0,0,1:1,0)
!hpf$  shadow     p(0,0,1:1,0)
!hpf$  asyncid id1
```

```
!hpf$  reflect f
c
c      first step
c
!hpf$  reflect u
!hpf$  reflect ff
!hpf$  reflect pp
c
c      second step
c
!hpf$  independent,new(i,j,k,m)
       do 371 k=1,nz1
!hpf$  on home(u(:,:,k,:)),local(f,u,i,j,m) begin
       do 3711 m=1,nb
       do 3711 j=1,ny1
       do 3711 i=1,nx1
       u(i,j,k,m)=0.125*(f(i,j,k,m)+f(i+1,j,k,m)
     1          +f(i,j+1,k,m)+f(i+1,j+1,k,m)
     2          +f(i,j,k+1,m)+f(i+1,j,k+1,m)
     3          +f(i,j+1,k+1,m)+f(i+1,j+1,k+1,m))
 3711  continue
!hpf$  end on
```

4. MHDコードにおけるMPI

```
mpi (Message Passing Interface)
c
CC MPI START
      parameter(nzz=(nz2-1)/npe+1)
      dimension f(nx2,ny2,0:nzz+1,nb),u(nx2,ny2,0:nzz+1,nb),
     1          ff(nx2,ny2,0:nzz+1,nb),p(nx2,ny2,0:nzz+1,nbb),
     2          pp(nx2,ny2,0:nzz+1,3)
CC MPI END
c
c No.1 mpi_send + mpi_recv and no wild card
c
!hpf$  reflect f
CC MPI START
      do m=1,nb
        if (irank.gt.0) then
          call mpi_send(f(1,1,ks,m),n2,mpi_real,irank-1,
     &                  100,mpi_comm_world,ier)
        end if
        if (irank.lt.isize-1) then
          call mpi_recv(f(1,1,ke+1,m),n2,mpi_real,irank+1,
     &                  100,mpi_comm_world,istatus,ier)
        end if
      end do
CC MPI END
c
```

c No.2 mpi_send + mpi_recv and wild card

```
c
!hpf$  reflect f
CC MPI START
      iright = irank + 1
      ileft  = irank - 1
      if(irank.eq.0) then
        ileft = MPI_PROC_NULL
      else if(irank.eq.isize-1) then
        iright = MPI_PROC_NULL
      end if
      do m=1,nb
         call mpi_send(f(1,1,ks,m),n2,mpi_real,ileft,
     &                 100,mpi_comm_world,ier)
         call mpi_recv(f(1,1,ke+1,m),n2,mpi_real,iright,
     &                 100,mpi_comm_world,istatus,ier)
      end do
CC MPI END
c
```

c No.3 mpi_sendrecv and wild card

```
c
!hpf$  reflect f
CC MPI START
      iright = irank + 1
      ileft  = irank - 1
      if(irank.eq.0) then
        ileft = MPI_PROC_NULL
      else if(irank.eq.isize-1) then
        iright = MPI_PROC_NULL
      end if
      do m=1,nb
      call mpi_sendrecv(f(1,1,ks,m),n2,mpi_real,ileft,100,
     &                  f(1,1,ke+1,m),n2,mpi_real,iright,100,
     &                  mpi_comm_world,istatus,ier)
      end do
CC MPI END
c
```

c No.4 mpi_isend + mpi_irecv and wild card

```
c
!hpf$  reflect f
CC MPI START
      iright = irank + 1
      ileft  = irank - 1
      if(irank.eq.0) then
        ileft = MPI_PROC_NULL
      else if(irank.eq.isize-1) then
        iright = MPI_PROC_NULL
      end if
c
      do m=1,nb
      call mpi_isend(f(1,1,ks,m),n2,mpi_real,ileft,
     &                100,mpi_comm_world,isend1,ier)
      call mpi_irecv(f(1,1,ke+1,m),n2,mpi_real,iright,
     &                100,mpi_comm_world,istatus,irecv1,ier)
      end do
      call mpi_wait(isend1,istatus,ier)
      call mpi_wait(irecv1,istatus,ier)
CC MPI END
c
```

MHDコードにおけるVPP FortranとHPF/JAとMPIの比較 (VPP5000)

Number of PE	Number of grids	VPP Fortran cpu time	VPP Fortran Gflops	VPP Fortran Gf/PE	HPF/JA cpu time	HPF/JA Gflops	HPF/JA Gf/PE	MPI cpu time	MPI Gflops	MPI Gf/PE
2PE	800x200x478	10.659	15.33	7.66	10.742	15.21	7.60	10.428	15.67	7.83
4PE	800x200x478	5.351	30.53	7.63	5.354	30.52	7.63	5.223	31.28	7.82
8PE	800x200x478	2.738	59.67	7.46	2.730	59.85	7.48	2.696	60.61	7.58
12PE	800x200x478	1.865	87.58	7.30	1.911	85.49	7.12	1.771	92.25	7.68
16PE	800x200x478	1.419	115.12	7.19	1.389	117.66	7.35	1.342	121.81	7.61
24PE	800x200x478	0.975	167.54	6.98	0.976	167.45	6.98	0.905	180.59	7.52
32PE	800x200x478	0.722	226.33	7.07	0.717	227.72	7.12	0.690	236.63	7.39
48PE	800x200x478	0.534	305.70	6.36	0.515	317.26	6.61	0.469	348.38	7.25
56PE	800x200x478	0.494	330.95	5.91	0.464	352.49	6.29	0.433	377.73	7.74
64PE	800x200x478	0.465	351.59	5.49	0.438	373.41	5.83	0.389	420.45	6.57
4PE	800x200x670	7.618	30.06	7.52	8.001	28.62	7.16	7.433	30.81	7.70
8PE	800x200x670	3.794	60.36	7.54	3.962	57.81	7.23	3.683	62.17	7.77
12PE	800x200x670	2.806	81.61	6.80	3.005	76.21	6.35	2.696	84.95	7.08
16PE	800x200x670	1.924	119.00	7.44	2.012	113.85	7.12	1.854	123.53	7.72
24PE	800x200x670	1.308	175.10	7.30	1.360	168.44	7.02	1.254	182.61	7.60
32PE	800x200x670	0.979	233.85	7.31	1.032	221.88	6.93	0.955	239.77	7.49
48PE	800x200x670	0.682	335.62	6.99	0.721	317.80	6.62	0.662	346.21	7.21
56PE	800x200x670	0.595	384.61	6.87	0.628	364.87	6.52	0.572	400.59	7.15
64PE	800x200x670							0.519	441.50	6.90

5. MHDコードにおけるHPFとMPIの比較

Table 1. Comparison of computer processing capability between HPF/JA and MPI in a 3D global MHD code by using Earth Simulator (ES) and Fujitsu VPP5000/64, earthd (nx,ny,nz)=(500,318,318).

Kind of computer	Number of cpu	HPF/JA or MPI cpu time (sec)	Gflops	Gf/PE	MPI cpu time (sec)	Gflops	Gf/PE	Date
		HPF/JA reflect			mpi_send and mpi_recv			
ES	1node x 1PE= 1PE	17.8244	(6.06)	6.06	( )			2003.01.21
ES	1node x 2PE= 2PE	9.3535	(11.55)	5.77	8.6676	(12.46)	6.23	2003.01.21
ES	1node x 4PE= 4PE	4.7028	(22.97)	5.74	4.3607	(24.77)	6.19	2003.01.21
ES	1node x 8PE= 8PE	2.3609	(45.76)	5.72	2.2075	(48.94)	6.12	2003.01.21
ES	2node x 8PE=16PE	1.1998	(90.03)	5.63	1.1229	(96.20)	6.01	2003.01.21
ES	4node x 8PE=32PE	0.6056	(178.38)	5.57	0.6023	(179.36)	5.61	2003.01.21
ES	8node x 8PE=64PE	0.3095	(349.00)	5.45	0.3803	(284.08)	4.44	2003.01.21
ES	10node x 8PE=80PE	0.2517	(429.17)	5.36	0.3529	(306.12)	3.83	2003.01.21
		mpi_isend and mpi_wait			mpi_sendrecv			
ES	1node x 2PE= 2PE				8.6499	(12.49)	6.24	2003.03.11
ES	1node x 4PE= 4PE				4.3484	(24.84)	6.21	2003.03.11
ES	1node x 8PE= 8PE	3.9438	(27.39)	3.42	2.1976	(49.15)	6.14	2003.03.11
ES	2node x 8PE=16PE				1.1020	(98.02)	6.13	2003.03.11
ES	4node x 8PE=32PE				0.5552	(194.56)	6.08	2003.03.11
ES	8node x 8PE=64PE	0.7031	(153.64)	2.40	0.2812	(384.15)	6.00	2003.03.11
ES	10node x 8PE=80PE	0.5631	(191.83)	2.40	0.2265	(476.92)	5.96	2003.03.11

Earth Simulator（ES）におけるHPF/JAとMPIの比較

Kind of computer	Number of cpu	HPF/JA or MPI cpu time (sec)	Gflops	Gf/PE	MPI cpu time (sec)	Gflops	Gf/PE	Date
		HPF/JA reflect			mpi_sendrecv			
ES	1node x 1PE= 1PE	17.8244	(6.06)	6.06	( )			2003.03.11
ES	1node x 2PE= 2PE	8.7212	(12.39)	6.19	8.6499	(12.49)	6.24	2003.04.04
ES	1node x 4PE= 4PE	4.3891	(24.61)	6.15	4.3484	(24.84)	6.21	2003.04.04
ES	1node x 8PE= 8PE	2.2221	(48.61)	6.08	2.1976	(49.15)	6.14	2003.04.04
ES	2node x 8PE=16PE	1.1168	(96.73)	6.05	1.1020	(98.02)	6.13	2003.04.04
ES	4node x 8PE=32PE	0.5683	(190.10)	5.94	0.5552	(194.56)	6.08	2003.04.04
ES	8node x 8PE=64PE	0.2943	(367.07)	5.74	0.2812	(384.15)	6.00	2003.04.04
ES	10node x 8PE=80PE	0.2410	(448.29)	5.60	0.2224	(485.65)	6.07	2003.04.04
		HPF without communication			MPI without communication			
ES	1node x 2PE= 2PE	8.1007	(13.33)	6.67	8.6499	(12.49)	6.24	2003.04.04
ES	1node x 4PE= 4PE	4.0669	(26.56)	6.64	4.3484	(24.84)	6.21	2003.04.04
ES	1node x 8PE= 8PE	2.0183	(53.52)	6.69	2.1976	(49.15)	6.14	2003.04.04
ES	2node x 8PE=16PE	1.0001	(108.01)	6.75	1.1020	(98.02)	6.13	2003.04.04
ES	4node x 8PE=32PE	0.4884	(221.16)	6.91	0.5552	(194.56)	6.08	2003.04.04
ES	8node x 8PE=64PE	0.2324	(464.86)	7.26	0.2546	(424.29)	6.63	2003.04.04
ES	10node x 8PE=80PE	0.1812	(596.11)	7.45	0.2034	(531.12)	6.64	2003.04.04

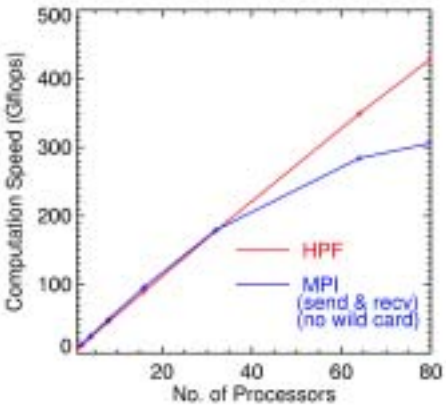
VPP5000におけるHPF/JAとMPIの比較

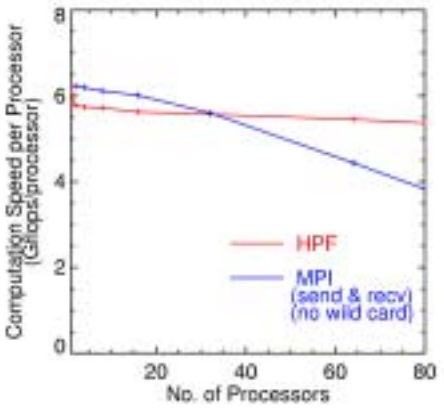
Kind of computer	Number of cpu	HPF/JA or MPI cpu time (sec)	Gflops	Gf/PE	MPI cpu time (sec)	Gflops	Gf/PE	Date
HPF/JA reflect		mpi_send and mpi_recv						
VPP5000	1PE	16.7886	(6.44)	6.44	( )			2003.01.26
VPP5000	2PE	8.3927	(12.87)	6.44	7.8305	(13.80)	6.90	2003.01.26
VPP5000	4PE	4.4073	(24.52)	6.13	4.0589	(26.61)	6.65	2003.01.26
VPP5000	8PE	2.1753	(49.67)	6.21	1.9973	(54.10)	6.76	2003.01.26
VPP5000	16PE	1.1446	(94.40)	5.90	1.0103	(106.95)	6.68	2003.03.21
VPP5000	32PE	0.6139	(176.01)	5.50	0.5116	(211.21)	6.60	3003.03.21
		mpi_isend and mpi_irecv			mpi_sendrecv			
VPP5000	2PE	7.8227	(13.81)	6.90	7.8334	(13.79)	6.90	2003.03.14
VPP5000	16PE				1.0109	(106.88)	6.68	3003.03.21
VPP5000	32PE	0.5089	(212.31)	6.63	0.5127	(210.74)	6.59	2003.03.18
		HPF without communication			MPI without communication			
VPP5000	2PE	7.8898	(13.69)	6.85	7.3934	(14.61)	7.31	2003.03.24
VPP5000	32PE	0.5016	(215.40)	6.73	0.4639	(232.90)	7.28	2003.03.19

Earth Simulatorでの大規模MHDシミュレーション

computer	grid number	sec	(GFLOPS)	GF/PE	Date
Earth Simulator (mearthd, hearthd)					
ES 128node x 8PE=1024PE (MPI)	2046x1022x1022	0.62047	(7357.1)	7.18	2003.04.04
	2046x1022x1022	0.62764	(7273.1)	7.10	2003.03.11
ES 64node x 8PE= 512PE (HPF/JA)	1022x1022x1022	0.68877	(3310.5)	6.47	2003.04.04
	1022x1022x1022	0.69604	(3275.9)	6.40	2003.03.11

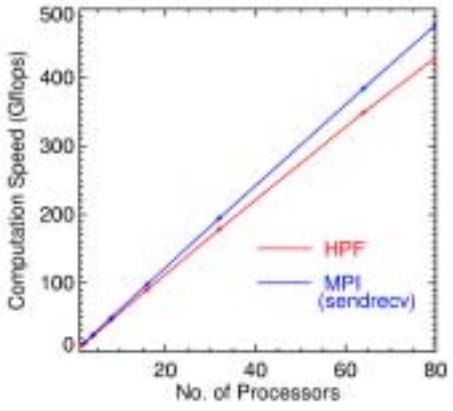
MHDコードのESでのテスト結果（1）

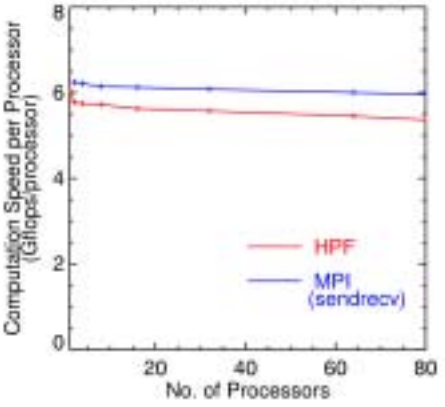




ベクトル化率 ：99.7〜98.8%
並列化率（最大） ：99.92%
最大利用可能プロセッサ数 ：1245台

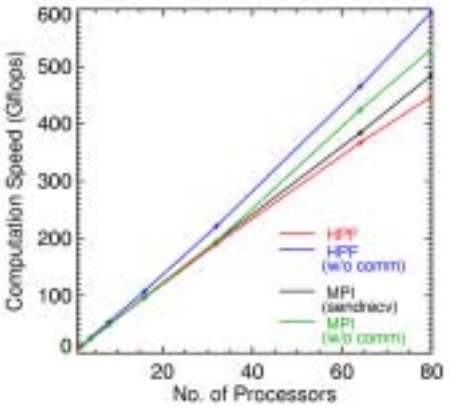
MHDコードのESでのテスト結果（2）

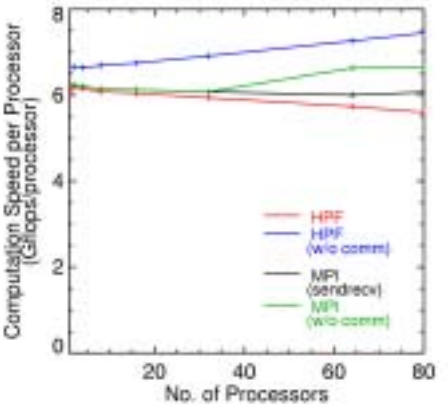




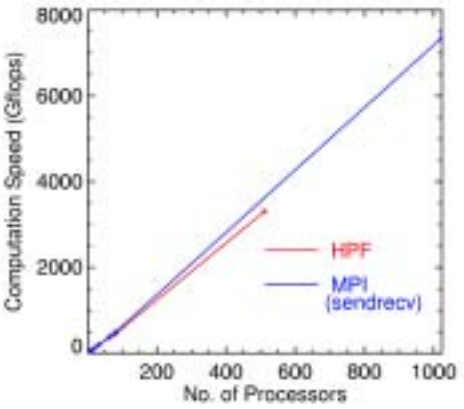
ベクトル化率 ：99.7〜98.8%
並列化率（最大） ：99.94%
最大利用可能プロセッサ数 ：1644台

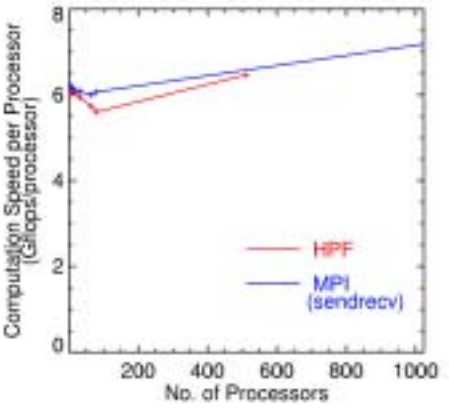
MHDコードのESでのテスト結果 (3)



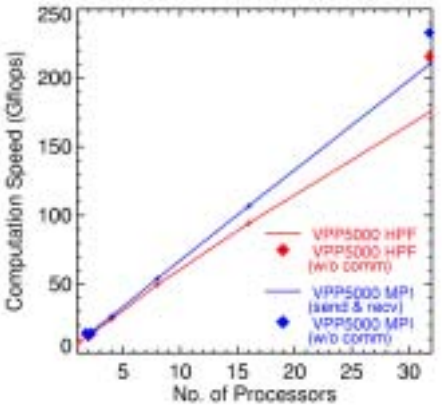


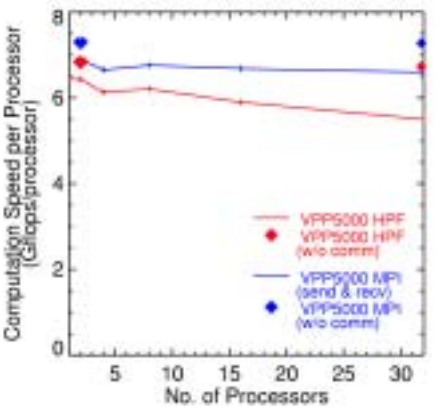
大規模MHDコードのESでのテスト結果 (4)





MHDコードのVPP5000でのテスト結果





6. まとめ

(1) HPF/JAの効能

- 比較的容易に並列化を実現できる。
- 完全独立なdo loopからなるプログラムであれば容易に並列化できる。
- かなり高効率の並列計算を実現できる(80%ー99.9%)。
- 領域分割間のデータ転送の記述がユニークに決まり(reflect)、かつ高効率のデータ転送を達成。
- 効率よく書かれたMPIのプログラムと比べてもそれほど遜色がない (約92%)。
- 領域分割間のデータ転送を切った場合の75.2%の並列計算を実現。

(2) HPF/JAとMPIの比較

・wild cardを用いないでmpi_send + mpi_recvを用いたMPIのMHDコードより、HPFの方が特に多数のcpuを用いた場合、高効率となった。

・wild cardとmpi_sendrecv用いたMPIのMHDコードがESでは最速を実現し、HPFを上回った。従って、現在ESではそのMPIのMHDコードを用いてシミュレーションを実行。

・mpi_isendとmpi_irecvの使用は、ESでは期待に反して遅くなった。

・MPIでは領域分割間のデータ転送を切った場合の91.4%の並列計算を実現。

・VPP5000のMPIでは、wild cardの使用と不使用、mpi_send + mpi_recv、mpi_sendrecv、mpi_isend + mpi_irecvの使用の差異はほとんどなかった。また、MPIがHPFより約20%高い効率であった。

(3) 現在の問題点

・領域分割間のデータ転送を実行せずに計算を行った場合（計算は正しくないがcpu間の通信を全て遮断した場合の計算速度を評価するため）、ESではHPFの方がMPIより高速であった(約112.2%)。
理由は？他方、VPP5000ではMPIの方がHPFより高速であった(約108.1%)。

・MPIを用い、(x,y,z)の3次元でz方向に領域分割して、各cpuに1枚又は2枚の格子面を割り当てた時、正常な計算が実行されなかった。各cpuに3枚の格子面を割り当てると正常に計算される。

・ESの利用ではデータ出力作業と出力した大量データの後処理に多くの時間と労力がかかっている。