14.9 TFLOPS Three-dimensional Fluid Simulation for Fusion Science with HPF on the Earth Simulator

Hitoshi Sakagami[1], Hitoshi Murai[2], Yoshiki Seo[3] and Mitsuo Yokokawa[4]

[1] Computer Engineering, Himeji Institute of Technology
2167 Shosha, Himeji, 671-2201 Hyogo, Japan
sakagami@comp.eng.himeji-tech.ac.jp
[2] Earth Simulator Center, Japan Marine Science and Technology Center
3173-25 Showa-machi, Kanazawa-ku, Yokohama, 236-0001 Kanagawa, Japan
murai@es.jamstec.go.jp
[3] Internet Systems Research Laboratories, NEC Corporation
4-1-1 Miyazaki, Miyamae, Kawasaki, 216-8555 Kanagawa, Japan
y-seo@ce.jp.nec.com
[4] Japan Atomic Energy Research Institute
6-9-3 Higashi-Ueno, Taito-ku, 110-0015 Tokyo, Japan
yokokawa@gaia.jaeri.go.jp

Abstract. We succeeded in getting 14.9 TFLOPS performance when running a plasma simulation code IMPACT-3D parallelized with High Performance Fortran on 512 nodes of the Earth Simulator. The theoretical peak performance of the 512 nodes is 32 TFLOPS, which means 45% of the peak performance was obtained with HPF. IMPACT-3D is an implosion analysis code using TVD scheme, which performs three-dimensional compressible and inviscid Eulerian fluid computation with the explicit 5-point stencil scheme for spatial differentiation and the fractional time step for time integration. The mesh size is 2048x2048x4096, and the third dimension was distributed for the parallelization. The HPF system used in the evaluation is HPF/ES, developed for the Earth Simulator by enhancing NEC HPF/SX V2 mainly in communication scalability. Shift communications were manually tuned to get best performance by using HPF/JA extensions, which was designed to give the users more control over sophisticated parallelization and communication optimizations.

1 Introduction

The Rayleigh-Taylor instability, which is one of fluid dynamics phenomena, can occur whenever a heavy dense fluid is accelerated by a light sparse fluid. Small perturbations on an interface between two fluids will grow in time and eventually form nonlinear bubble-spike structures, which can lead to the Kelvin-Helmholtz like instability.

In inertial confinement fusion with ablatively accelerated targets, the Rayleigh-Taylor instability can be induced both in acceleration and stagnation phases, and can destroy spherical symmetry of the imploding target. In the stagnation phase, the

0-7695-1524-X/02 $17.00 (c) 2002 IEEE

perturbation at the pusher-fuel contact surface is unstable to the Rayleigh-Taylor instability. One-dimensional simulations show that nuclear reaction takes place predominantly near the maximum compression and neutron yield increases by more than two orders of magnitude in the stagnation phase. It is, therefore, very important in this research field to investigate this instability since the pusher-fuel mixing, which is associated with the Rayleigh-Taylor instability, reduces the total nuclear reaction yield to significantly lower than the value predicted by one-dimensional simulations.

We have been investigating linear and nonlinear features of the fully three-dimensional Rayleigh-Taylor instability in spherically stagnating targets through numerical simulations [1-3]. The Rayleigh-Taylor instability in spherical geometry is quite different from that in planar geometry because acceleration and wavelength vary in space and time. Nonlinear evolution in three-dimensional simulations also significantly differs from that obtained by two-dimensional simulations. A single processor can only compute lower mode phenomena due to the limitation of computer ability. Even if lower modes are dominant to determine overall implosion dynamics, higher modes are generated by mode coupling and rapidly grow than lower modes. Thus it is also important to track them by simulations with parallel computers to precisely analyze experimental results.

In general, researchers in the computer science field are interested in a computer itself, but those in the computational science field want to investigate science itself using computers as convenient tools. Thus parallel programming is a usual work and even a research subject for computer scientists, but it is never an object for computational scientists. Computational scientists think that many bothersome works associated with the computer science, such as the parallel programming, should be reduced automatically as much as possible, and are hoping to concentrate their attention on their essential research. Thus most of computational scientists consider parallel computers as tools on which many jobs for a parameter survey can be run simultaneously at many computer centers in Japan. This fact is one of reasons that have inhibited the spread of parallel computers, and computational scientists have not received actual benefits and potential ability of parallel computers for high performance computing yet. Not only computer scientists, but also computational scientists want to use distributed memory parallel machines if applicable. In order to adapt the distributed memory parallel computer from a special kind of machine for computer scientists to a general convenient tool for computational scientists, the high-level language that can easily describe parallelization in programs is essential.

We used High Performance Fortran as the high-level language and inserted its directives into the plasma simulation code that is designed to simulate three-dimensional Rayleigh-Taylor instability in spherical systems. 12.5TFLOPS sustained performance was achieved with automatic optimizations for shift communications by HPF/ES compiler on the Earth Simulator with 2048x2048x4096 meshes. With the further optimization using the HPF/JA extensions, which is a set of HPF extensions designed to give the users more control over sophisticated parallelization and communication optimizations, we accomplished 14.9TFLOPS sustained performance, 45% of the peak performance of the 512 node Earth Simulator.

2 Three-Dimensional Fluid Code

IMPACT-3D (IMPlosion Analysis Code with TVD scheme) is fully Eulerian and this is necessitated from the requirement to model rotational flows accurately. The Rayleigh-Taylor instability leads to formation of nonlinear bubble-spike structures, which generate vorticity in the unstable region of flows and distort the contact surface tremendously. Thus, flows induced by the Rayleigh-Taylor instability are typically rotational and standard Lagrangian algorithms are not accurate enough for such flows without rezoning (reconstruction of the mesh), which introduces an unacceptable amount of numerical diffusion. A Cartesian coordinate system is employed in IMPACT-3D to model convergent asymmetric flows precisely. Other coordinate systems contain a singularity at the origin that would cause inaccuracies. The TVD scheme can capture discontinuities such as shock wave fronts and contact surfaces within a few meshes even after many time steps, and it has a second order accuracy both in space and time without introducing non-physical oscillations at the discontinuity. The origin of the target is located at the center of the mesh system with a uniform grid spacing, and the physical values on all boundaries are obtained by extrapolating the inside values according to the open free boundary condition. The non-physical perturbations induced at boundaries do not affect the Rayleigh-Taylor instability at the contact surface because there is a sufficiently thick ablator layer between them and the non-physical perturbations are smoothed out.

Perturbations at the interface exponentially grow in the linear stage. After saturation of the linear growth, the instability shifts to the nonlinear free-falling phase and forms bubble-spike structures [2]. Time evolution of isovalue-surfaces of the density, corresponding to the unstable interface, is shown in Fig. 1 when an initial perturbation is given by a spherical harmonics function with mode number (n, m) = (6, 3) and 101x101x101 meshes. The interface typically shows the nonlinear bubble-spike structures in the three-dimensional spherical system. During the free-fall phase, bubbles are gradually isolated from each other and surrounded by spikes, while spikes are combined with each other. The bubbles float into the heavy fluid (pusher) and vortex rings are developed to feed bubbles by blowing off the light fluid (fuel) into them, especially around the base of the bubbles. As the pusher spikes penetrate into the center of the target, the vortex rings are tightened and enhance this feeding mechanism. The growing speed in the nonlinear stage of the Rayleigh-Taylor instability is characterized by those vortex rings that are induced in the nonlinear bubble-spike structure [3]. The pusher-fuel contact surface and the vorticity in this final stage are shown in Fig. 2.

Recently advanced laser systems for the inertia confinement fusion can be highly aligned and they can irradiate fusion targets more uniformly than ever. Thus to inspect experimental results requires more precise simulations which calculate not only lower mode but also higher mode phenomena. In addition, Fast Ignition that is an entirely different scheme to achieve fusion burning was proposed and preliminary experiments have just begun [4]. Ultra large-scale simulations are required to analyze those experimental results and phenomena with enough precision, and they will be accomplished only with parallel supercomputers.

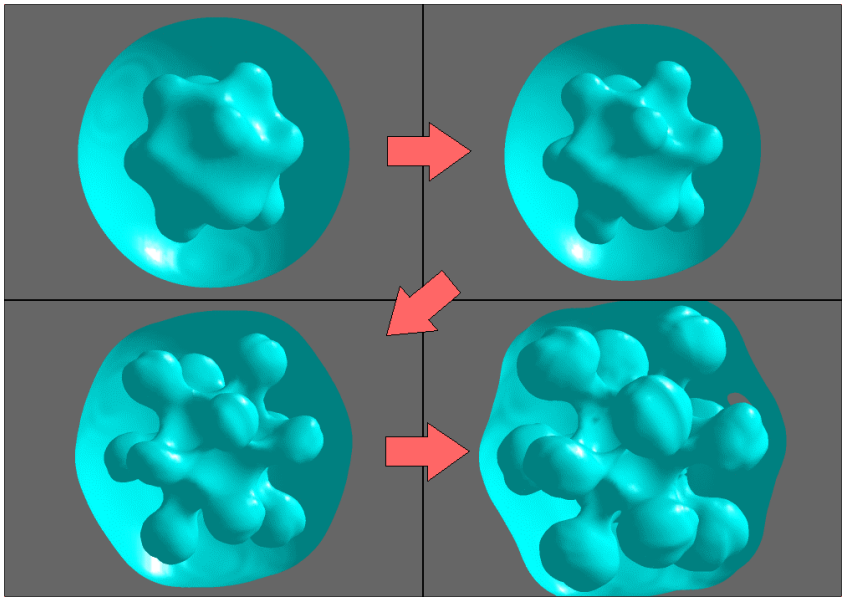

Fig. 1. The nonlinear time evolution of the Rayleigh-Taylor instability in the stagnating system. The perturbation is induced as a spherical harmonics function (n,m)=(6,3).

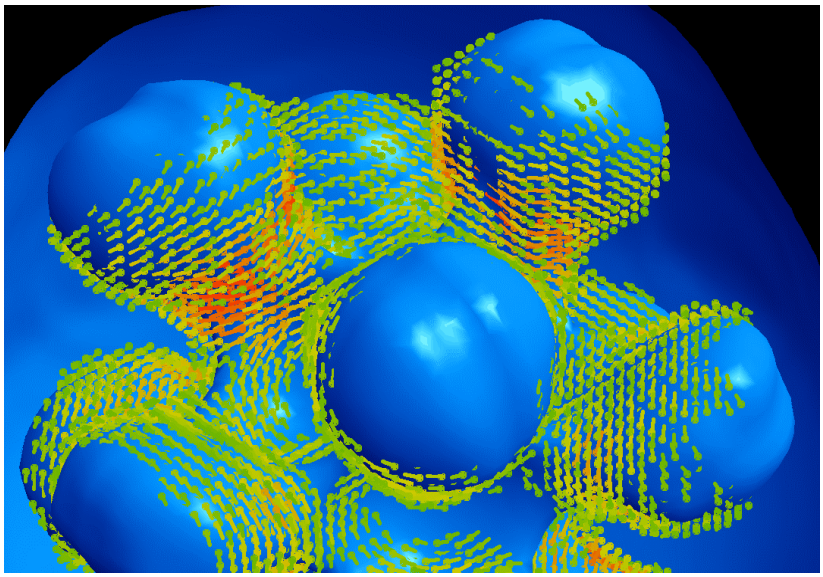

Fig. 2. The pusher-fuel contact surface and the vorticity. The perturbation is induced as a spherical harmonics function (n,m)=(6,3). Red, yellow and green vectors indicate strong, medium and weak vorticities, respectively.

3 Environment

3.1 High Performance Fortran

In order to develop parallel programs on high-speed parallel supercomputers, MPI (Message Passing Interface)[5] is generally used. However, since communications among processors must be described explicitly in MPI, it is not easy for ordinary users. On shared memory parallel computers, OpenMP reduces the difficulty of parallel programming, but it cannot be used on distributed memory parallel computers. High Performance Fortran (HPF) is a parallel programming language developed in 1993 by the High Performance Fortran Forum to provide a high-level and easy-to-use portable programming interface for distributed memory parallel systems. It can be classified as a data parallel language. The users can parallelize their programs with a single-threaded execution model mainly by specifying mapping of array data on distributed memories. Since a program is described with a pure data-parallel paradigm in HPF, it has high affinity with the programming style using traditional Fortran. Relatively low performance of parallel execution had barred the spread of HPF until now. But the recent progress of HPF compilers has changed the situation. HPF is now ready to be used for real-world applications and ready to be a very important tool for computational scientists to take an advantage of distributed memory parallel computers.

From an advanced user's point of view, HPF2.0 language specification [6] lacks functionality for optimizing data transfers. JAHPF (Japan Association for HPF) [7], an informal coalition of HPC users and computer vendors in Japan setup in 1997 to promote the HPF language, has defined HPF/JA 1.0 [8] as a set of HPF extensions to give the users more control over sophisticated parallelization and communication optimizations. The extensions include parallelization of loops with complicated reductions, asynchronous communication, user-controllable shadow, and communication pattern reuse for irregular remote data accesses.

In parallelizing IMPACT-3D, we used REFLECT and LOCAL directives to tune communications. In the HPF2.0 specification, the consistency of the shadow data with the original data is controlled by a compilation system. There are few ways for the user to explicitly control it. This frequently results in redundant communications. In HPF/JA the shadow specification is extended so that the user can control communication on it. The REFLECT directive is regarded as an execution statement that keeps the coherency of the shadow data. The LOCAL directive guarantees the accesses to arrays in a list do not require inter-processor communications. The user can eliminate redundant communications for the shadow area by the combined use of the REFLECT and LOCAL directives.

Recently Japanese computer venders have released HPF compilers equipped with the HPF/JA extensions for their vector-parallel supercomputers. These HPF compilers are very reliable and facilitate getting good performance for real-world scientific applications [9-12]. JAHPF is now expanded into a new organization, HPF Promoting Consortium (HPFPC) [13], where various HPF promotion activities are supported.

3.2 The Earth Simulator [14]

The Earth Simulator is a high-speed parallel vector computer developed for research on natural and environmental changes. Target sustained performance of the Earth Simulator is set to 1,000 times higher than that of the most frequently used supercomputers around 1996 in climate research field. Execution performance of 35.86TFLOPS was approved by the result of the Linpack benchmark and the Earth Simulator was registered as the world's fastest supercomputer at June 20th, 2002. It is expected that the Earth Simulator will bring about great progress in the earth science research.

The Earth Simulator is a distributed memory parallel system that consists of 640 processor nodes (PNs) connected by a 640 x 640 internode crossbar switch. Each node is a shared memory system composed of eight arithmetic processors (APs), a shared memory system of 16GB, a remote control unit (RCU), and an I/O processor (IOP). Each AP contains a vector unit (VU), a scalar unit (SU), and a main memory access control unit, which are mounted on a one-chip LSI operating at clock frequency of 500MHz, partially 1GHz. The VU consists of 8 sets of processing units, each of which has six kinds of vector pipelines (add/shift, multiply, divide, logical, mask, and load/store) and 72 vector registers of 256 elements. Eight pipelines of the same kind work together by a single vector instruction, and those of different types can operate concurrently. The SU is a 4-way super-scalar processor with a 64KB instruction cache, a 64KB data cache, and 128 general-purpose scalar registers. The SU employs branch prediction, data prefetching and out-of-order instruction execution. The memory system (MS) in each PN is symmetrically shared by 8 APs. It is configured by 32 main memory package units (MMU) with 2048 banks. Each AP has a 32GB/s memory bandwidth, and one node has 256GB/s in total. The RCU in each PN is directly connected to the crossbar switch by two ways of sending and receiving, and controls internode data communications. The interconnection network (IN) consists of two parts: one is an internode crossbar control unit (XCT) that coordinates switching operations; the other is an internode crossbar switch (XSW) that functions as an actual data path. XSW is composed of 128 separate switches, each of which has 1Gbits/s bandwidth operating independently. Any pairs of the switches and nodes are connected by electric cables. The theoretical data transfer rate between every two PNs is 12.3GB/s x 2 ways. The peak performance of each AP is 8GFLOPS. The total number of processors is 5,120 and the total peak performance and the main memory capacity are 40TFLOPS and 10TB, respectively. Advanced CMOS technology and air-cooling system is used. The hardware specification is summarized in Table 1.

An operating system running on a processor node is a UNIX-based system to support large-scale scientific computations. Compilers of Fortran90, HPF, C and C++ are available with automated support for vectorization and parallelization. A message passing library based on MPI-1 and MPI-2 is also available. HPF/ES is an HPF compiler developed for the Earth Simulator by enhancing NEC HPF/SX V2 [15] in many respects. It provides some unique extensions as well as the features of HPF 2.0, its approved extensions, and HPF/JA to support efficient and portable parallel programming. The unique extensions include vectorization directives, features optimization of irregular communications, parallel I/O, and so on. HPF/ES detects

SHIFT-type communications automatically and generates calls to the highly optimized runtime routines of the SHIFT-type communication. Furthermore, the communication schedule generation is reused when the same communication patterns are iterated across loop executions.

The Earth Simulator system has a three-level hierarchy of parallelism; vector processing in an AP, SMP parallel processing among APs in a PN, and parallel processing among PNs. In this evaluation, however, we do not exploit the hierarchical parallelism explicitly, but map HPF abstract processors onto APs in a flat fashion.

Table 1. The Hardware specification of the Earth Simulator.

Total number of processor nodes:	640
Number of AP for each node:	8
Total number of AP:	5120
Peak performance of each AP:	8 GFLOPS
Peak performance of each PN:	64 GFLOPS
Peak performance of total system:	40 TFLOPS
Shared memory of each PN:	16 GB
Total main memory:	10 TB

4 Accomplished Performance

4.1 Automatic Communications

IMPACT-3D performs three-dimensional compressible and inviscid Eulerian fluid computation with the explicit 5-point stencil scheme for spatial differentiation and the fractional time step for time integration. Therefore, it is easy to parallelize this code with an ordinary domain decomposition method [16]. The first dimension of the three-dimensional computation space is used for vectorization and the third dimension for HPF parallelization. As the section size of vector registers is 256 and the vector length for half performance is around 50, we can secure enough vector length to bring out vector performance.

We distributed all three-dimensional array variables with (*,*,BLOCK) and shadow regions are allocated for the distributed dimension. All parallelizable loops except one in the code were automatically parallelized without inserting INDEPENDENT directives. The only one exception was a loop that includes reduction operations. For this loop, an INDEPENDENT directive followed by a REDUCTION clause was inserted. A typical HPF code fragment of IMPACT-3D is shown in Fig. 3. Communications required for IMPACT-3D is SHIFT-type communications and reductions. HPF/ES succeeded in generating optimized communications for them and eliminating all the redundant data transfers [10]. It performed message coalescing optimization for all the possible opportunities. In other words, more than one SHIFT- type communications of the same pattern for a loop nest are packed into one message to make the message length larger.

```
      parameter(lx=1024, ly=1024, lz=2048)
!HPF$ PROCESSORS es(NUMBER_OF_PROCESSORS())
      dimension sr(lx,ly,lz), sm(lx,ly,lz)
!HPF$ DISTRIBUTE (*,*,BLOCK) onto es :: sr, sm
!HPF$ SHADOW (0,0,0:1) :: sr, sm
      do iz = 1, lz-1
        do iy = 1, ly
          do ix = 1, lx
            …
            wu1 = sm(ix,iy,iz+1) / sr(ix,iy,iz+1)
            …
          end do
        end do
      end do
      …
      do iz = 1, lz-1
        do iy = 1, ly
          do ix = 1, lx
            …
            wu1 = sm(ix,iy,iz+1) / sr(ix,iy,iz+1)
            …
          end do
        end do
      end do
c-----
!HPF$ INDEPENDENT, REDUCTION(MAX:sram)
      do iz = 1, lz
        do iy = 1, ly
          do ix = 1, lx
            …
            sram = max( sram, … )
          end do
        end do
      end do
```

Fig. 3. A typical HPF code fragment of IMPACT-3D. All array variables are distributed with (*,*,BLOCK) and shadow regions are defined for efficient SHIFT-type communications. An INDEPENDENT directive followed by a REDUCTION clause was inserted for a loop of reduction operations.

Communications generated by HPF/ES are composed of two phases, communication schedule construction and message transfer. The schedule is a set of information required for transferring messages in the second phase, such as buffers to be transferred, communication pattern, pairs of the sender and receiver. The structure of the schedule is designed to be independent of a target array. The schedule constructed in a communication can, therefore, be shared with others if the arrays to be transferred have the same shape and distribution, and are accessed in the same index pattern. HPF/ES performs the optimization of communication pattern reuse. It generates just one communication schedule for the off-processor array accesses of the same pattern inside of a subroutine boundary. Many techniques to optimize SHIFT-type communications was developed [17,18], but our optimization method is superior to them in the point that schedule construction and message transfer can be optimized separately.

In the example shown in Fig. 3, SHIFT-type communications for *sm* and *sr* are generated just once before the first loop nest, and no communication is generated for

the second nest. The communication schedule for *sm* is reused for *sr*. Furthermore, the communications for *sm* and *sr* are coalesced.

For measuring effective performance, we used an instrumentation facility provided by MPI/ES, which underlies the runtime system of HPF/ES. MPI/ES collects runtime performance information using hardware counters. This facility displays performance information in detail as shown in Fig. 4. Performance figures of maximum, minimum and average are displayed. A sustained performance figure is calculated by dividing the total floating operation count by elapsed time. The total floating operation count are computed by multiplying average floating operation count (FLOP count) and the number of processors. The elapsed time is measured separately by mpi_wtime().

The measurement environment is summarized in Table 2. A special compiler option, *-Moverlap=size:0*, was used to suppress the compiler's automatic allocation of SHADOW region .

```
Global Data of 4096 processes:          Min [U,R]              Max [U,R]          Average
==============================

Real   Time (sec)           :      387.892 [0,2734]       390.076 [0,2375]       390.029
User   Time (sec)           :      378.252 [0,3519]       386.147 [0,13]         379.812
System Time (sec)           :        0.067 [0,13]           0.926 [0,2127]         0.185
Vector Time (sec)           :      318.009 [0,4095]       335.689 [0,8]          330.992
Instruction Count           :  47383497003 [0,4095]   51329038206 [0,11]     50823805078
Vector Instruction Count    :   9969541284 [0,4095]   13102470101 [0,8]      13063204823
Vector Element Count        :2548332077307 [0,4095] 3345775534394 [0,4]    3337405620723
FLOP Count                  : 840544981245 [0,4095] 1162667553602 [0,1024] 1162483025543
MOPS                        :     6810.288 [0,4095]      8930.044 [0,4093]      8886.425
MFLOPS                      :     2213.811 [0,4095]      3073.787 [0,3519]      3060.688
Average Vector Length       :      254.422 [0,15]         255.699 [0,254]        255.481
Vector Operation Ratio (%)  :       98.553 [0,4095]        98.883 [0,2413]        98.881
Memory size used (MB)       :     1870.260 [0,8]         1892.447 [0,0]         1886.204
MIPS                        :      124.798 [0,4095]       134.389 [0,4093]       133.813
Instruction Cache miss (sec):        0.074 [0,4095]         0.231 [0,4031]         0.102
Operand     Cache miss (sec):        1.368 [0,4095]         1.589 [0,3511]         1.458
Bank     Conflict Time (sec):        0.363 [0,4095]         0.566 [0,3608]         0.538
```

Fig. 4. Runtime performance information is provided by MPI/ES. The total number of floating operations can be calculated by multiplying average FLOP count and the number of processors, and total sustained performance can be calculated by this divided by elapsed time separately measured using mpi_wtime().

Table 2. The summary of measurement environment.

OS:	ESOS Release 1.1
HPF compiler:	HPF/ES Rev.1.7(585)
compiler options:	-O3 -Moverlap=size:0

We made the performance evaluation using three kinds of mesh size, namely 1280x1280x1280, 1024x1024x2048 and 2048x2048x4096. As the third dimension is distributed, it determines the maximum number of processors effectively utilized. Total sustained performance and its efficiency to the peak performance as a function of number of nodes are shown in Fig. 5. We achieved 12.5 TFLOPS, 39% of the peak performance with the largest mesh size on 512 nodes (4096 processors). We are very encouraged to get this outstanding performance on a real-world scientific application parallelized with HPF. For this achievement, 45 lines of HPF directives are inserted to 1334 lines (without comment lines) of the original Fortran program.

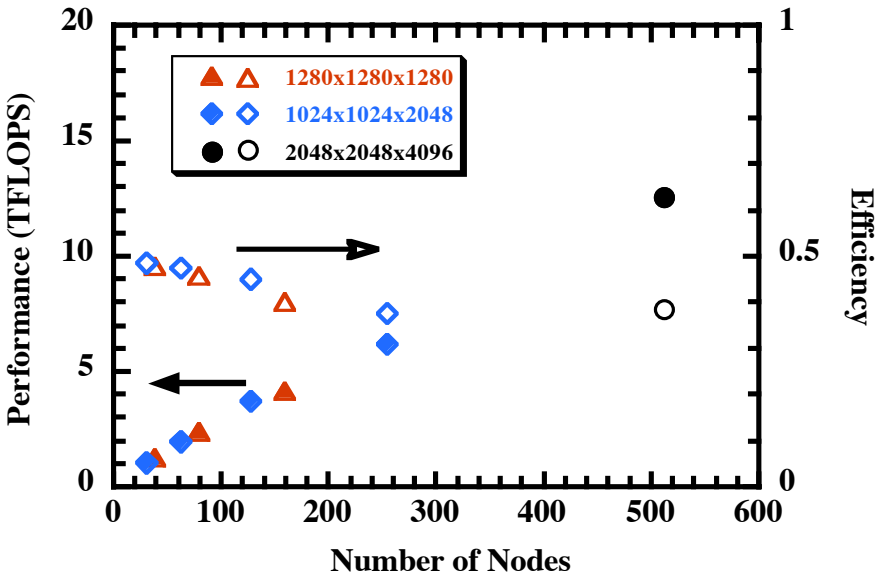

Fig. 5. Total sustained performance (solid) and its efficiency to the peak performance (hollow) as a function of number of nodes. Triangle, diamond and circle markers correspond to mesh size of 1280x1280x1280, 1024x1024x2048 and 2048x2048x4096, respectively. As each node consists of eight processors, total number of processors can be obtained by multiplying number of nodes by eight. For example, 512 nodes have 4096 processors.

4.2 Manual Communication Optimization with HPF/JA

We made a further performance tuning by using the REFLECT and LOCAL directives defined in the HPF/JA language specification. The directives are designed so that a user can explicitly control updates to a shadow region by REFLECT, and tell the compiler no necessity of communications for a specific array access by LOCAL.

As HPF/ES could perform extensive communication optimizations, there were no differences in the inserted communications themselves whether or not the directives are inserted. However, we were able to reduce the number of communication schedule generations by using the directives. When the compiler makes the automatic optimization (*automatic*), it has to generate the schedule computation at least once for a subroutine invocation, because the communication pattern is computed using the runtime information of array access pattern. On the other hand, when the directives are explicitly specified (*manual*), the communication pattern can be determined statically, because the array elements to be transferred can be computed just by the shape, mapping, and shadow of the array, which are specified in declarative statements. Thus, the *manual* optimization can reduce the frequency of the schedule construction to only once for each array throughout the whole program execution. Once a schedule is generated, it is passed across subroutines so that it can be reused.

A typical HPF code fragment using the REFLECT and LOCAL directives is shown in Fig. 6. In total, 12 lines of the directives were inserted for the optimization.

Fig. 7 shows total sustained performance and its efficiency for two kinds of mesh size, 1024x1024x2048 and 2048x2048x4096. By inserting the HPF/JA directives, 14.9 TFLOPS, 45% of the performance was achieved on the 512 nodes (4096 processors).

```
!HPF$ REFLECT sm, sr
      do iz = 1, lz-1
!HPF$ ON HOME(sm(:,:,iz)), LOCAL BEGIN
        do iy = 1, ly
          do ix = 1, lx
            …
            wu1 = sm(ix,iy,iz+1) / sr(ix,iy,iz+1)
            …
          end do
        end do
!HPF$ END ON
      end do
      …
      do iz = 1, lz-1
!HPF$ ON HOME(sm(:,:,iz)), LOCAL BEGIN
        do iy = 1, ly
          do ix = 1, lx
            …
            wu1 = sm(ix,iy,iz+1) / sr(ix,iy,iz+1)
            …
          end do
        end do
!HPF$ END ON
      end do
```

Fig. 6. A typical HPF code fragment using the REFLECT and LOCAL directives. By explicitly specifying the directives, the number of communication schedule generations can be reduced.

4.3 Performance Comparison with an MPI Version

We made the performance comparison between the HPF coding and the MPI coding. To parallelize the three-dimensional fluid code with MPI, we made the modifications on the following points:

- loop bounds for all the parallelized DO loops modified to local bounds
- all mapped arrays to be distributed
- all array references to the distributed data are modified
- SHIFT-type communications inserted for all variables to exchange boundary data between neighboring processors
- allreduce communication inserted for reduction operations.

Table 3 summarizes the results for the 1024x1024x2048 mesh execution. HPF performances without inserting HPF/JA directives, HPF performances with the directives, and MPI performances are shown in the columns of *HPF(auto)*,

HPF(manual), and *MPI*, respectively. Performance ratios of the HPF executions compared to MPI are also shown. In the 256 node execution, the MPI program achieved 46% of the peak performance, and the HPF(auto) obtained more than 80% of this MPI performance. Furthermore, HPF(manual) achieved 95% of the MPI performance. Even with the optimization with HPF/JA extensions, HPF/ES generates communication schedule construction at least once for each distributed array, but users can directly specify the communication pattern in the MPI calls without creating the schedules. This explains the performance degradation of HPF. However, we are fully satisfied with the HPF performance considering the MPI programming requires much more work than HPF coding.

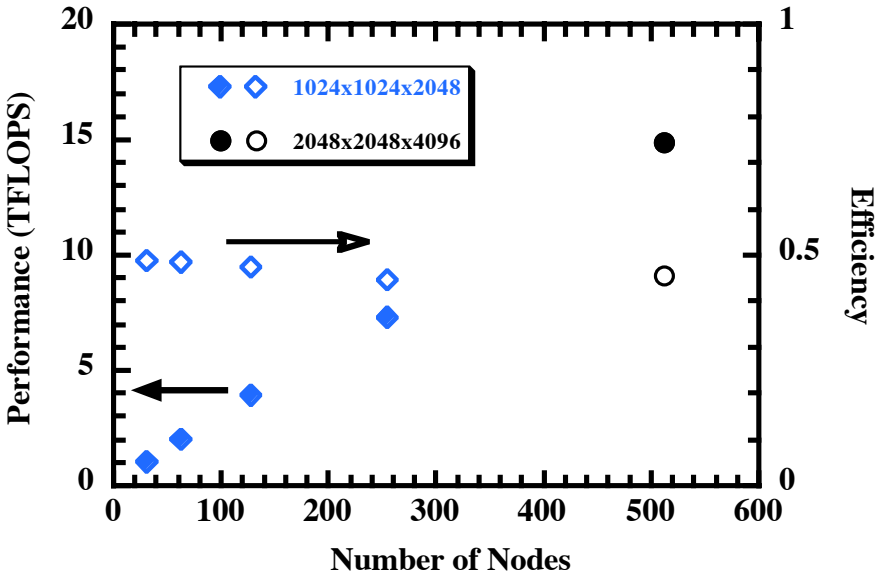

Fig. 7. Total sustained performance (solid) and its efficiency to the peak performance (hollow) as a function of number of nodes. Diamond and circle markers correspond to mesh size of 1024x1024x2048 and 2048x2048x4096, respectively. We can finally get 14.9 TFLOPS of the total sustained performance, which is corresponding to 45% of the peak performance.

Table 3. Total sustained performances for HPF and MPI, and performance ratios between HPF and MPI with 1024x1024x2048 meshes. HPF(auto) indicates that communications are optimized by the compiler as described in Section 4.1, but communications are manually tuned with HPF/JA extensions in HPF(manual) as described in Section 4.2.

	TFLOPS			Performance Ratio to MPI	
node#	HPF(auto)	HPF(manual)	MPI	HPF(auto)	HPF(manual)
32	0.987	0.996	1.021	0.97	0.98
64	1.937	1.977	2.026	0.96	0.98
128	3.674	3.876	3.990	0.92	0.97
256	6.143	7.302	7.659	0.80	0.95

5 Summary

We have parallelized a plasma simulation code IMPACT-3D with HPF on the Earth Simulator. We have only distributed all three-dimensional array variables with (*,*,BLOCK). The HPF compiler automatically parallelized all of the do loops except one reduction loop and optimized SHIFT-type communications, and we could get 12.5 TFLOPS sustained performance when running the code with the mesh size of 2048x2048x4096 on 512 nodes of the Earth Simulator. We manually tuned the SHIFT-type communications by using REFLECT and LOCAL directives of HPF/JA, and 14.9 TFLOPS sustained performance, 45% of the peak performance, was finally accomplished. We would like to emphasize that this excellent achievement has been done with HPF programming.

The Earth Simulator is just available this March and we have not enough time to run the code. Although the fluid code, IMPACT-3D, is a real scientific application, never a benchmark program, the physical simulation result itself is very preliminary and we can not get any new knowledge from analyzing simulation results. But we will perform ultra large-scale simulations in the near future and boldly go into the frontiers no man has gone before. We would like to use the HPF system to parallelize many other simulation programs to accelerate the earth science.

References

1. Sakagami, H. and Nishihara, K.: Rayleigh-Taylor Instability on Pusher-Fuel Contact Surface of Stagnating Targets, Phys. Fluids B, Vol. 2, pp. 2715-2730 (1990).
2. Sakagami, H. and Nishihara, K.: Three Dimensional Rayleigh-Taylor Instability of Spherical Systems, Phys. Rev. Lett., Vol. 65, pp. 432-435 (1990).
3. Sakagami, H. and Nishihara, K.: Three Dimensional Rayleigh-Taylor Instability of Laser Fusion Target, Proc. of Int. Conf. on Plasma Physics, Foz do Iguasu, Brazil, October 31 - November 4, Vol. 1, pp. 249-252 (1994).
4. Kodama, R., et. al.: Fast heating of ultrahigh-density plasma as a step towards laser fusion ignition, Nature Vol. 412, pp. 798-802 (2001).
5. Message Passing Interface Forum: A Message-Passing Interface Standard, Int. Journal of Supercomputing Applications and High Performance Computing, Vol.8, pp. 165-416 (1994).
6. High Performance Fortran Forum: High Performance Fortran Language Specification Version 2.0 (1996).
7. http://www.hpfpc.org/jahpf/.
8. Seo, Y., Iwashita, H., Ohta, H. and Sakagami, H.: HPF/JA: extensions of High Performance Fortran for accelerating real-world applications, Concurrency and Computation: Practice and Experience, Vol. 14, pp. 555-573 (2002).
9. Ogino, T.: Three-dimensional global MHD simulation code for the Earth's magnetosphere using HPF/JA, Concurrency and Computation: Practice and Experience, Vol. 14, pp. 631-646 (2002).
10. Sakagami, H. and Mizuno, T.: Compatibility Comparison and Performance Evaluation for Japanese HPF Compilers using Scientific Applications, Concurrency and Computation: Practice and Experience, Vol. 14, pp. 679-689 (2002).

11. Nishitani, Y., Negishi, K., Ohta, H. and Nunohiro, E.: Techniques for compiling and implementing all NAS parallel benchmarks in HPF, Concurrency and Computation: Practice and Experience, Vol. 14, pp. 769-787 (2002).
12. Sakagami, H., Mizuno, T. and Furubayashi, S.: Parallelization Methods for Three-Dimensional Fluid Code using High Performance Fortran, Proc. of Int. Conf. on Parallel CFD 2002, Nara, Japan, May 20-22, (2002).
13. http://www.hpfpc.org/.
14. Tani, K.: Status of the Earth Simulator System, Proc. of the 16th Int. Supercomputer Conf., Heidelberg, Germany, Jun 19-22 (2001).
15. Murai, H., Araki, T., Hayashi, Y., Suehiro, K. and Seo, Y.: Implementation and evaluation of HPF/SX V2, Concurrency and Computation: Practice and Experience, Vol. 14, pp. 603-629 (2002).
16. Sakagami, H. and Ogawa, Y.: Performance Evaluation for 3D Fluid Code and 2D Particle Code Using HPF, Proc. of 3rd HPF User Group meeting, Redondo Beach, USA, August 1-2 (1999).
17. Chakrabarti, S., Gupta, M., and Choi, J.-D.: Global communication analysis and optimization, Proc. of ACM SIGPLAN Conference on Programming Language Design and Implementation, Philadelphia, PA, USA, May 21-24 (1996).
18. Roth, G., Mellor-Crummey, J., Kennedy, K. and Brickner, R. G.: Compiling Stencils in High Performance Fortran, Proc. of SC'97: High Performance Networking and Computing, San Jose, CA, USA, November 15-21 (1997).