J. Plasma Physics (2006), *vol.* 72, *part* 6, *pp.* 1087–1090. © 2006 Cambridge University Press
doi:10.1017/S0022377806005496 Printed in the United Kingdom

Parallelization of SPEC OMP benchmark programs with HPF and evaluation of their performances

HITOSHI SAKAGAMI[1] and HIROYUKI MORII[2]

[1]National Institute for Fusion Science, 322-6 Oroshi-cho, Toki 509-5292, Japan
[2]Computer Engineering, University of Hyogo, 2167 Shosha, Himeji 671-2201, Japan

(Received 15 August 2005 and accepted 23 February 2006)

Abstract. OpenMP is a directive-oriented paradigm for parallel programming applicable to shared memory parallel computers. On the other hand, High Performance Fortron (HPF) is also a directive-oriented paradigm but for distributed memory machines. Thus, it could be very interesting to compare their programming applicability and parallel performances. We have converted four SPEC OMP 2001 benchmark programs, which are written with OpenMP and designed to evaluate the performance of shared memory parallel computers, into HPF, and evaluated parallel performances. We also point out problems that we had faced during conversion, and discuss solutions for them. Finally, we have improved the HPF performance by modifying the original source code. We could get good performance with HPF quite similar to OpenMP with SWIM, and moderate performance with APSI.

1. Introduction

On shared memory parallel computers, OpenMP reduces the difficulty of parallel programming, but it cannot be used on distributed memory parallel computers. Thus, High Performance Fortran (HPF), which is a data-parallel language and enables parallel execution on distributed memory computers only by embedding minimum directive lines into original Fortran sequential programs, has been proposed. Since a program is described with a pure data-parallel paradigm (single control thread and data treated in global name space) in HPF, it has high affinity with traditional Fortran in programming styles. Recently, a Japanese computer vender, NEC, has released a HPF compiler equipped with the HPF/JA extensions [1] for the vector-parallel supercomputers, SX series and the Earth Simulator. The HPF compiler is very reliable and facilitates getting good performance for real-world scientific applications [2–4]. Since both OpenMP and HPF are directive-oriented paradigms for parallel programming, we are very interested in comparing their programming applicability and parallel performances. So, in this research project, we have converted some SPEC OMP 2001 benchmark programs [5], which are written with OpenMP and designed to evaluate performance of shared memory parallel computers, into HPF. We have investigated how we could convert OpenMP programs into HPF without much effort and tried to get the same performance.

https://doi.org/10.1017/S0022377806005496 Published online by Cambridge University Press

```
      dimension a(10),b(9)
!HPF$ PROCESSORS proc(2)
!HPF$ DISTRIBUTE (BLOCK) &
!HPF$     ONTO proc :: a,b
      do i = 1, 9
        a(i+1) = ...
        b(i)   = ...
      end do
```

(a) Typical DO loop

```
      allocate(atemp)
      do i = 1, 9
         atemp(i) = ...
         b(i)     = ...
      end do
      do i = 1, 9
         a(i+1) = atemp(i)
      end do
      deallocate(atemp)
```

(b) Compiler generated code

Figure 1. Sample program code of SWIM.

We have also point out problems that we had faced during conversion, and discuss solutions for them and how we could improve the HPF performance by modifying the original source code [6].

2. Parallelization and performance

As a general rule, HPF directives are inserted to execute DO loops, which are specified for parallel processing by OpenMP, in parallel. We run parallel programs on the shared memory vector-parallel computer, one node of the NEC SX-5/128M8 system (16 processors in a node, 160 GFLOPS and 28 GB memory per node) at Cybermedia Center, Osaka University for both HPF (HPF/SX V2 Rev.1.9.2) and OpenMP (FORTRAN90/SX Version 2.0 for SX-5 Rev.280).

SWIM, which solves the system of shallow water equations using finite difference approximations on a two-dimensional grid, is our first target. In SWIM, the parallel DO loop contains different subscripts that are used to store data in different arrays (Fig. 1(a)). When the DO index i is 5, the assignment statement for $a(i+1)$ must be executed by processor 2 but that for $b(i)$ by processor 1 due to the Owner Computes Rule (OCR) of HPF. Thus, the loop division of this DO loop for parallel execution causes contradiction to the OCR. In such a case, the HPF compiler tries to resolve the contradiction and automatically generates the code in which a temporary array is dynamically allocated, assigned values are stored to it in the same DO loop and data are restored from it to the original array (Fig. 1(b)). This technique leads to less efficiency due to overheads of extra actions. So we adjust the data distributions with the ALIGN directive [!HPF$ ALIGN b(i) WITH a(i+1)] to avoid the conflict between the OCR and subscripts. We can improve the performance without source modification, but we cannot adjust for multiple DO loops simultaneously. We resolve this problem with splitting the DO loop into two DO loops in which all of the subscripts are identical and adapted to the OCR, and then we can get good performance with HPF quite similar to OpenMP. We emphasize that the HPF code can be executed on more than one node of SX-5 even when the OpenMP code is confined within one node.

MGRID demonstrates the capabilities of a very simple multigrid solver in computing a three-dimensional potential field. In MGRID, a very large one-dimensional

https://doi.org/10.1017/S0022377806005496 Published online by Cambridge University Press

array is declared in the main program and it is passed to subroutines as an actual argument with different start addresses and sizes while a dummy argument corresponding to this one-dimensional array is declared as a three-dimensional array with different sizes in subroutines. This technique is frequently used to make a large code with Fortran77, but HPF does not support this feature for distributed arrays. Thus, we have to rewrite subroutines to allocate a temporary three-dimensional array on the fly. As the dynamic allocation/deallocation of the array and copying data to/from it are performed each time the subroutine is called, these extra actions cause the large performance degradation as compared with OpenMP. As the calculation is repeated in the main DO loop for multiple data in this benchmark program, we can put all subroutines in that DO loop into one special subroutine and replace calling all subroutines in the DO loop by calling only this new subroutine. This modification of the source code is in contrast to the general rule in which the parallel region of OpenMP should be parallelized by HPF, but we can globally parallelize MGRID by concurrent calling of the subroutine. With this modification we can get moderate performance with HPF compared with OpenMP. It is noted that we need much more memory with this method because arrays are allocated for each processor as NEW variables.

APSI solves for the mesoscale and synoptic variations of potential temperature, horizontal wind components, and the mesoscale vertical velocity pressure and distribution of pollutants. APSI also used the same technique about arguments as used in MGRID, namely the shape of actual arguments is different from that of dummy arguments. Unlike MGRID, assignments of a large one-dimensional array to a three-dimensional array are static. So we can declare individual three-dimensional arrays instead of the unique one-dimensional array in the main program, and pass them to subroutines. In some cases, the rank of the actual argument for parallel execution is different to that of the dummy argument, but the array should be distributed with proper dimension for parallel execution. We use automatic remapping to a distributed array at calling subroutine to change the rank. To concurrently call the subroutine for parallel execution, we also rewrite the actual argument as the array section of the distributed array and add the interface block with FORTRAN_LOCAL extrinsic and PURE attribute, and finally we can get moderate performance.

APPLU, which is using a nonlinear block symmetric SOR iterative scheme with lexicographic ordering, is the last program. Usually SOR algorithm cannot be parallelized due to recurrence of the array access, and we cannot parallelize the code by only inserting HPF directives. Thus, we decided to use the pseudo-hyperplane method [7] because less modification to the source code is needed, and we performed shift-type communications by calling native MPI subroutines from HPF programs. We must use HPF_LOCAL extrinsic subroutine to directly call MPI functions within the subroutine. To reduce the overhead of copying arguments, we can use the array section as an actual argument in the subroutine for pipelining. We can get reasonable performance with HPF compared with OpenMP due to inefficiency of the pseudo-hyperplane method. We use the array section as an actual argument and this causes it to copy the argument to the temporary array. This copying must be done 4×218 times with $5 \times 224 \times 224 \times 2$ data per loop, and leads to the degradation of performance.

Elapsed times of all four benchmark programs in seconds are summarized in Table 1 for both HPF and OpenMP.

https://doi.org/10.1017/S0022377806005496 Published online by Cambridge University Press

Table 1. Elapsed times of four benchmark programs in seconds for HPF and OpenMP

Number of processors	SWIM		GRID		APSI		APPLU	
	HPF	OpenMP	HPF	OpenMP	HPF	OpenMP	HPF	OpenMP
1	294.7	283.2	275.2	230.2	1020.5	928.4	3472.6	2526.1
2	153.3	150.4	149.7	118.6	541.6	477.6	2139.4	1279.9
4	82.3	41.4	76.0	63.1	267.4	235.4	1170.1	663.2
8	51.3	48.4	44.2	36.0	171.2	136.6	604.7	351.9

3. Summary

We have converted four SPEC OMP 2001 benchmark programs into HPF programs with the general rule that DO loops in parallel regions of OpenMP should also be parallelized in HPF. In SWIM, we must avoid the conflict between the OCR of HPF and subscripts by using ALIGN directives and splitting the DO loop, and can get good performance with HPF quite similar to OpenMP. As the shapes of actual and dummy arguments are different in MGRID, we must add the code of dynamic allocation and copying for the temporary array. However, using the dynamic temporary array causes much degradation of the performance, and we cannot improve the performance by HPF. In contrast to our general rule of converting OpenMP program into HPF, we can globally parallelize the program by concurrent calling of a subroutine and can get moderate performance. In APSI, we must individually declare the arrays to solve the argument shape problem and use the array section of the distributed array for parallel subroutine calling, and can get moderate performance. We use the pseudo-hyperplane method with native MPI subroutine calling for the shift-type communication in APPLU. We can improve the performance by using the array section as actual argument, but we can only get reasonable performance.

References

[1] Seo, Y., Iwashita, H., Ohta, H. and Sakagami, H. 2002 HPF/JA: extensions of High Performance Fortran for accelerating real-world applications. *Concurrency Comput.: Practice Experience* **14**, 555–573.

[2] Ogino, T. 2002 Three-dimensional global MHD simulation code for the Earth's magnetosphere using HPF/JA. *Concurrency Comput.: Practice Experience* **14**, 631–646.

[3] Sakagami, H. and Mizuno, T. 2002 Compatibility comparison and performance evaluation for Japanese HPF compilers using scientific applications. *Concurrency Comput.: Practice Experience* **14**, 679–689.

[4] Sakagami, H., Mizuno, T. and Furubayashi, S. 2003 Parallelization methods for three-dimensional fluid code using High Performance Fortran. In: *Parallel Computational Fluid Dynamics – New Frontiers and Multi-disciplinary Applications* (eds K. Matsuno, A. Ecer, J. Periaux, N. Satofuka and P. Fox), pp. 203–210. Amsterdam: Elsevier.

[5] SPEC OMP 2001 benchmark programs. http://www.specbench.org/.

[6] Morii, H., Sakagami, H., Nii, M. and Takahashi, Y. 2003 Rewriting OpenMP benchmark programs with HPF and evaluating their parallel performances. *Technical Report 56*, Himeji Institute of Technology, pp. 43–56.

[7] Murai, H. and Okabe, Y. 2004 Implementation and evaluation of NAS Parallel Benchmarks with HPF on the Earth Simulator. In: *Proceedings of SACSIS2004*, pp. 389–396 Information Processing Society of Japan.

https://doi.org/10.1017/S0022377806005496 Published online by Cambridge University Press