§60. Conversion of OpenMP Benchmark Programs into HPF and Evaluation of Parallel Performance

Sakagami, H., Morii, H. (Dept. Computer Eng., Univ. of Hyogo)

Most of computational scientists cannot accept to parallelize their codes using a low-level programming interface such as MPI, which is now a de facto standard in this field, and computational scientists have not received actual benefits and potential ability of parallel computers for high performance computing yet. In order to adapt the 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 indispensable. On shared memory parallel computers, OpenMP reduces the difficulty of parallel programming, but it cannot be used on distributed memory parallel computers. Data parallel language HPF (High Performance Fortran), which enables parallel execution on distributed memory computers only by embedding minimum directive lines into the original sequential program, 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 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.

Since both OpenMP and HPF are directive-based systems, we are very interesting in comparing their parallel performance and programming applicability. Thus in this research project, we have converted some SPEC OMP 2001 benchmark programs,[1] which are written in OpenMP and designed to evaluate performance of shared memory parallel computers, into HPF. We have also pointed out problems that we have faced on during conversion, and discussed about solutions for them.[2]

As a policy, HPF directives are inserted to execute DO loops, which are specified for parallel processing by OpenMP, in parallel. 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, and it causes contradiction to Owner Computes Rule (OCR) of HPF for parallel execution. In such a case, HPF compiler tries to resolve the contradiction using dynamic allocation of temporary array and copying data to/from it. This compiler's technique leads to less efficiency due to overheads of extra actions. Thus we resolve the contradiction with dividing the DO loop into two DO loops in which all of subscripts are identical and adapted to OCR. We execute both codes on the shared memory vector-parallel computer, one node of NEC SX-7 at NIFS, and can get good performance with HPF quite similar to OpenMP with this rewriting the source code of SWIM. We emphasize that the HPF code can be executed on more than one node of SX-7 even the OpenMP code is confined within one node. MGRID, which demonstrates the capabilities of a very simple multigrid solver in computing a three dimensional potential field, is the second target. In MGRID, very large one-dimensional array is declared in the main program and it is passed to subroutines as an actual argument with different start addresses while a dummy argument corresponding to this one-dimensional array is declared as three-dimensional array 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 as follows: (1) declare dummy arguments as one-dimensional array, (2) allocate temporary three-dimensional arrays on the fly, (3) copy from the one-dimensional dummy arguments to the temporary three-dimensional arrays, (4) parallelize with the temporary three-dimensional arrays by HPF, (5) copy back to the dummy arguments, and (6) deallocate the temporary arrays. The dynamic allocation/deallocation of the array and copying data to/from it are performed each time the subroutine is called, and cause large performance degradation as compared with OpenMP. APSI, which solves for the mesoscale and synoptic variations of potential temperature, horizontal wind components, and the mesoscale vertical velocity pressure and distribution of pollutants, is the third target. APSI also used above technique about arguments, but assignments of the large one-dimensional array to three-dimensional one are static unlike MGRID, and we can declare individual three-dimensional arrays instead of the unique one-dimensional array in the main program and pass them to subroutines. We use remapping to a distributed array at calling subroutine to change dimension of the three-dimensional array for appropriate parallel execution. We also rewrite actual arguments as array sections of distributed arrays to call the subroutine in parallel, and finally we can get moderate. Execution time in seconds of each code with HPF and OpenMP is summarized in Table 1. Codes are executed with one to 32 processors.

Table 1. Execution time [sec] with HPF and OpenMP.

	SWIM		MGRID		APSI	
#Proc.	OMP	HPF	OMP	HPF	OMP	HPF
1	232.8	247.9	197.2	284.1	624.2	766.5
2	122.1	126.4	99.4	158.5	315.7	394.1
4	65.1	65.3	50.7	92.9	165.3	203.7
8	36.9	35.1	26.9	59.1	88.3	111.3
16	22.8	21.1	19.5	43.9	52.8	75.0
32	17.3	18.1	12.1	41.3	44.7	77.4

References
1) http://www.specbench.org/.
2) Morii, H. et al., Tech. Report of Himeji Inst. Tech., **56** (2003) 43.