Performance Evaluation of Large-scale Parallel Simulation Codes and Designing New Language Features on the HPF (High Performance Fortran) Data-Parallel Programming Environment

Project Representative
Yasuo Okabe Kyoto University

Author
Hitoshi Murai NEC Corporation

High Performance Fortran (HPF) is provided for parallelizing your programs on the Earth Simulator (ES). Our goal in this project is to parallelize large-scale real applications and study more efficient use of HPF. The achievements of this year are: (1) a global atmospheric simulation code and the SPEC OMP benchmarks are parallelized and evaluated on the ES; (2) a preprocessor that provides the functionality of pipelined parallelization is developed; and (3) the HPF/ES compiler is improved and enhanced in some points.

Keywords: High Performance Fortran, HPF, atmospheric simulation, benchmark

1. Introduction

We believe that a parallelizing means not only easier to use than but also as efficient as MPI is essential for future parallel supercomputing, and that High Performance Fortran (HPF) can play the role.

An HPF compiler HPF/ES is provided on the Earth Simulator (ES). We plan to parallelize large-scale real applications from various fields, such as atmosphere, ocean, plasma, FEM and aerodynamics, with HPF to evaluate them on the ES and investigate the results in detail to learn more efficient use of HPF. We will also study the programming methods of hierarchical parallelization with HPF, because it is important to take advantage of all of the inter-node parallelization, intra-node parallelization and vector processing to fully exploit the performance of the ES. Improvements or new features required to make HPF more useful will be detected and proposed.

The achievements of this year include evaluation of a global atmospheric simulation code, parallelization of the SPEC OMP benchmarks, development of a preprocessor that adds the functionality of pipelined parallelization to HPF compilers, and improvements of the HPF/ES compiler.

2. Evaluation

2.1. Real-world application

We parallelized with HPF/ES a global atmospheric simulation code based on the Global Atmospheric Model (GAM) originally developed by the Australian Bureau of Meteorology.

This program simulates mainly a global kinetic process of atmosphere. It is a Fortran program of about 35,000 lines, which includes 280 lines of HPF directives and 40 lines of EXTRINSIC prefixes, where main arrays are distributed by cyclic. Although we now parallelize the program in the flat manner, we plan to adopt the hybrid manner for higher performance.

The resolution of the simulation is T239L29 (480 grids along the longitude axis and 240 along the latitude).

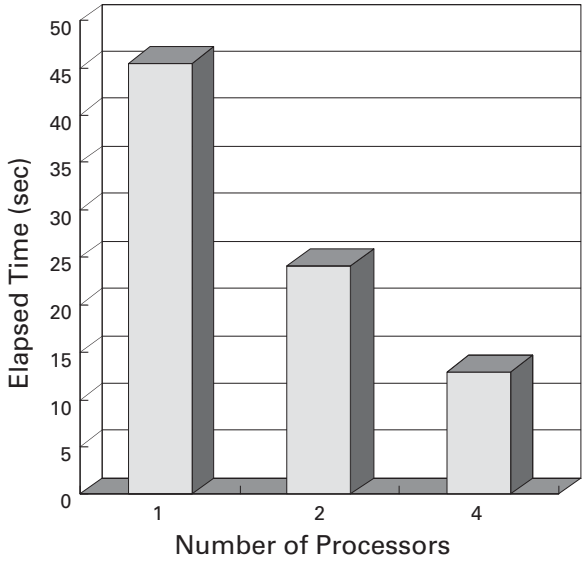

Fig. 1 Evaluation Result of the Global Atmospheric Model (Elapsed Time for executing three steps of the kernel loop)

The preliminary evaluation of the kernel loop, which is free of I/O, shows that it achieves a 3.5 times speedup in the 4-APs execution on the ES (Fig. 1). This speedup degradation is mainly due to the overhead of re-distributing cyclic-distributed large arrays in forward and inverse Legendre transformations and forward and inverse Fourier transformations performed in each time step. These transformations dominate the whole execution time.

2.2. Benchmark

We plan to parallelize the programs from SPEC OMP[1] with HPF/ES. The goal of this study is to:

- compare the usability and effectiveness of HPF with those of OpenMP;
- store knowledge of HPF programming; and
- develop the "SPEC HPF" benchmark.

The development of the programs is now completed but the evaluation on the ES has not been done yet. See [5] for the evaluation results on other platforms.

This study tells us that for regular or structured problems, it is easy to achieve high performance with HPF/ES if the programmer specifies good data distribution of arrays to reduce communication. On the other hand, it is difficult to investigate the cause for poor performance of a program. However, if once the cause is investigated, it is relatively easy to remove it.

3. Study of New Language Features

3.1. Implementation

The pipelined parallelization is one of the functionality that the HPF specification does not support. Although some of the existing HPF compilers support it [2], HPF/ES has not yet.

We developed a preprocessor named HPFX which translates an HPF source program annotated by the HPFX's `PIPELINE` directive into a normal HPF one, and ran the generated program on the ES to evaluate the performance of pipelined parallelization in HPF programs.

The syntax and semantics of the `PIPELINE` directive is as follows:

E101	pipeline-directive	is	!HPFX PIPELINE(pipeline-array-list)
E102	pipeline-array	is	array-name(pipeline-region-list) (pipeline-spec-list)
E103	pipeline-region	is	int-expr or [int-expr]:[int-expr]
E104	pipeline-spec	is	int-expr

Constraint: A `PIPELINE` directive must be followed by an `INDEPENDENT` directive with ignoring comments.

Constraint: The dimension of the template with which a dimension of the array *array-name* is aligned must be distributed by neither `CYCLIC` nor `INDIRECT`.

A `PIPELINE` directive is an assertion that there is a loop-carried dependence described by *pipeline-spec-list* on the array *array-name*.

The preprocessor generates communications to parallelize the following loop in the pipeline fashion. The region of the array described by *pipeline-region-list* is to be communicated or, in other words, is the region accessed in the loop.

A simple example of the `PIPELINE` directive is shown below.

```
      REAL A(100)
!HPF$ PROCESSORS P(4)
!HPF$ DISTRIBUTE A(BLOCK) ONTO P

!HPFX PIPELINE(A(:),(1))
!HPF$ INDEPENDENT
      DO I=1, 99
!HPF$   ON HOME(A(I)), LOCAL
        A(I) = A(I) + A(I-1)
      END DO
```

The `PIPELINE` directive asserts to the preprocessor that there is a loop-carried dependence of distance one, which means that one element on the distribution boundary is to be sent to the neighbor processor, in the whole region of the array `A`.

The preprocessor translates the above into the code below.

```
      call recv1_real(A, wdth, 1, lb, ub)
!HPF$ INDEPENDENT
      DO I=2, 100
!HPF$   ON HOME(A(I)), LOCAL
        A(I) = A(I) + A(I-1)
      END DO
      call send1_real(A, wdth, 1, lb, ub)
```

It can be seen that one call statement is inserted before the loop and another after the loop. The called subroutines `recv1_real` and `send1_real` are generated from a template by the preprocessor. They are extrinsic subroutines of kind `HPF_LOCAL`, each of which are embedded with the calls to MPI subroutines and communicates with each other at runtime.

The runtime sequential behavior of each processor for the example is as follows:

0. Processor `P(1)` go through the call to `recv1_real` while the others wait here for arriving data from their neighbor.
1. `P(1)` performs the loop to process the array `A`.
2. `P(1)` sends `A(25)` to `P(2)`.
3. `P(2)` receives `A(25)` from `P(1)` and exits from `recv1_real`.
4. `P(2)` performs the loop to process the array `A`.
5. ...

Fig. 2 illustrates these steps.

3.2. Evaluation

We parallelized the LU benchmark (class C) in the NPB[3] with the preprocessor in two manners: one is the one-dimensional pipeline and another two-dimensional. We ran both of them on the ES, and the results are illustrated in Fig. 3. The results of some other implementations are also illustrated for comparison. The vertical axis represents the speedup relative to the single-CPU execution of the Fortran

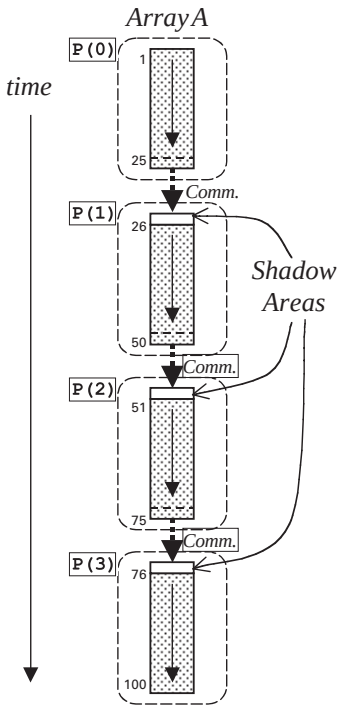

Fig. 2 Runtime behavior of Pipelined Execution

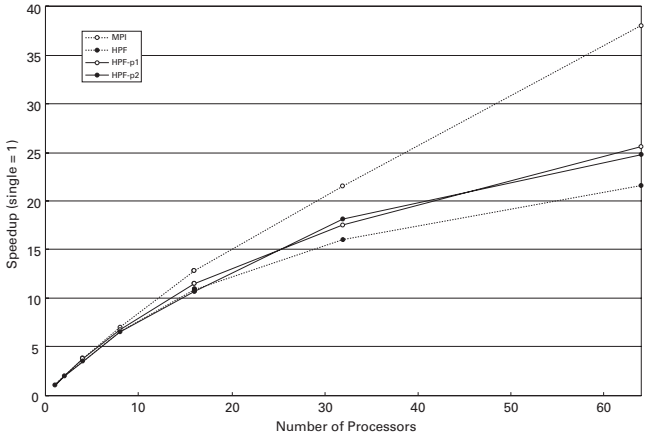

Fig. 3 Scalability of the NPB LU benchmark

implementation (NPB3.1-serial).

Note that all of the HPF and MPI implementations derive from the code of NPB3.1. We exploit flat parallelization for all of the implementation.

HPF-p1 in the graph is the result of the one-dimensional and HPF-p2 the two-dimensional. They have little difference in performance and outperform the normal HPF implementation (illustrated by HPF), which exploits the hyperplane method[4] to parallelize the DOACROSS loop. However the MPI one gives still higher performance.

The MPI implementation exploits the two-dimensional pipeline algorithm that is the same as our HPF-p2 implementation. Our investigation reveals that the reason MPI outperforms HPF-p2 is mainly the overhead of calling the mapping inquiry subroutines `GLOBAL_ALIGNMENT` and `GLOBAL_DISTRIBUTION`, which are included the HPF extrinsic local library, and constructing the communication schedule.

Since HPFX is not a compiler module but a preprocessor, it can refer to neither the information from compiler analysis nor the runtime information. Accordingly, such overheads of calling the mapping inquiry subroutines and constructing the communication schedule are rather difficult to avoid in the current preprocessor implementation.

The overheads should be removed if the compiler supports the functionality of the preprocessor, and therefore it can be said that the evaluation results show the effectiveness of the pipeline parallelization.

4. Improvements and Enhancements of the HPF/ES Compiler

We have requested the developer to improve and enhance HPF/ES in some problems revealed by our study in FY2003 and FY2004. The recent update resolves some of the problems as follows:

(a) overlapped execution of the shadow area (EXT_HOME) is available;

(b) partial REFLECT is available; and

(c) some communication patterns performs better.

– multidimensional shift communication

– array re-distribution

5. Conclusion

We parallelized a global atmospheric simulation code and SPEC OMP code with HPF/ES, and evaluated them on the ES. The results show the detail of the advantage and disadvantage of HPF. We developed a preprocessor for HPF/ES that processes pipelined parallelization to verify the effectiveness of such pipelined parallelization in HPF. It is shown from the evaluation using the NPB LU benchmark that pipelined parallelization is superior to the conventional hyperplane methed in performance. However, it cannot perform as well as MPI because of its limited access to the compiler's information. This performance degradation should be improved if the compiler supports the functionality of the preprocessor.

There are following future works planned:

- evaluation of real-world applications (contd.);
- cross-platform evaluation of benchmarks; and
- study of new language extensions and features.

Bibliographies

1) Standard Performance Evaluation Corporation, "SPEC OMP," `http://www.spec.org/omp/`.

2) Nishitani, Y. et al., "Techniques for compiling and implementing all NAS parallel benchmarks in HPF," Concurrency and Computation - Practice ¥& Experience, Vol.14, No.8-9, Wiley, pp.769-787 (2002).

3) D.E. Bailey, et al., "The NAS Parallel Benchmarks," Technical Report RNR-94-007, NASA Ames Research

Center, 1994.
4) H. Murai and Y. Okabe, "Implementation and Evaluation of NAS Parallel Benchmarks with HPF on the Earth Simulator," In proc. of SACSIS2004, Sapporo, Japan, May 2004.
5) Morii, H. et al., Evaluation of Parallel Performance with HPF, IPSJ SIG Notes 2004-HPC-99, pp.271-276, 2004.

並列処理言語HPF（High Performance Fortran）を用いた大規模並列実行の性能検証および新規機能の検討

プロジェクト責任者
岡部　寿男　　京都大学
著者
村井　　均　　NEC

地球シミュレータ(ES)では、High Performance Fortran (HPF)を用いてプログラムの並列化を行うことができる。本プロジェクトは、HPF によって大規模実アプリケーションを並列化し、HPF のより効率的な活用法を研究することを目的とする。今年度の成果は次の3つである: (1) 大気シミュレーションコードと SPEC OMP ベンチマークを並列化し、ES 上で評価を行った; (2) パイプライン並列化を行うプリプロセッサを開発した; (3) HPF/ES コンパイラの改良および強化を行った。

キーワード: High Performance Fortran，HPF，大気シミュレーション，ベンチマーク