Compiling High Performance Fortran to Message Passing *

Thomas Brandes
German National Research Center for Information Technology (GMD)
Schloß Birlinghoven, D-53754 St. Augustin, Germany
e-mail: brandes@gmd.de

Abstract

ADAPTOR is a public domain High Performance Fortran compilation system that provides the comfortable data parallel programming paradigm on parallel machines with distributed memory. Therefore, the data parallel programs with their global view of data are translated to programs that work on the local parts of the distributed data and exchange the other needed data via message passing.

This paper describes the design and the implementation of the ADAPTOR system and discusses its portability by using a portable message passing interface. The advantages of using the new message passing interface MPI are outlined. Furthermore, the combination of data parallel programs with efficient message passage programs is discussed that will allow the use of existing libraries.

1 Introduction

High Performance Fortran (HPF) is the new de facto standard language for writing data parallel programs for shared and distributed memory parallel architectures [9]. HPF provides a much more friendly programming paradigm for users, who just want their results fast, without needing to worry about details of the parallelism.

An HPF program should run effectively on a wide range of parallel machines, including distributed memory machines. On these, programmers have become used to writing "message passing" programs, e.g. using the portable message passing interface PVM [16] or MPI [14]. HPF programs are much easier to write and to read than conventional message passing programs. They contain no message passing at all though the compiler has to insert some into the final code.

ADAPTOR is a public domain High Performance Fortran compilation system [3] that provides the comfortable data parallel programming paradigm on distributed

*This work was funded partly by the Esprit-6643 project PPPE (Portable Parallel Programming Environment)

memory machines. Therefore the data parallel programs with their global view of data are translated to programs that work on the local parts of the distributed data and exchange the other needed data via message passing.

All the routines for handling the distributed data and for the communication are provided in the HPF runtime system. The portability of such a runtime system between different message passing systems is an important issue. Using a portable message passing interface guarantees independency of the underlying hardware architecture. The runtime system of the ADAPTOR tool can be used on all machines where PVM or MPI is available. Though most users of ADAPTOR currently prefer using PVM, in section 3 we discuss the benefits of MPI for our future HPF runtime system.

HPF programs are usually less efficient as their hand-written message passing counterparts. Therefore it is important to provide an interface from HPF to well tuned message passing libraries. Section 4 shows how this can be realized and how this has been implemented within ADAPTOR for interfacing HPF with ScaLAPACK [4], a library of high performance linear algebra routines for distributed memory MIMD computers.

2 Description of the ADAPTOR Tool

2.1 System Overview

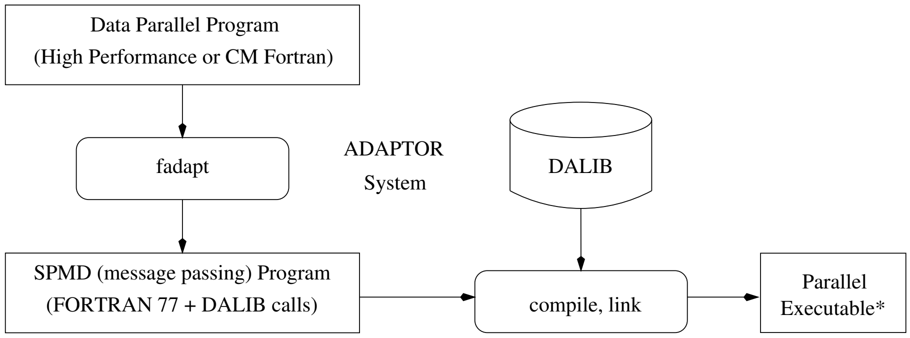

Figure 1: Overview of the ADAPTOR tool

ADAPTOR has two major components (see Figure 1). A source-to-source transformation translates the data parallel program written in HPF to an equivalent SPMD program (single program, multiple data) working on the local parts of the distributed arrays and exchanging the data via message passing. The generated program is a FORTRAN 77 program that has to be compiled by a native compiler and to be linked with the DALIB runtime system that is the second component of the ADAPTOR system. The parallel executable can be started on the available nodes of the parallel machine.

2.2 The Translation System

The essential idea of the source-source-translation is to assign the arrays of the source program to the memory of the node processors as specified by the HPF mapping directives (see Figure 2).

The control flow and statements with scalar code are replicated on all nodes. The parallel loops and array operations are restricted to the local part owned by the processor. Communication statements for exchanging non local data will be generated automatically as subroutine calls to the runtime system.

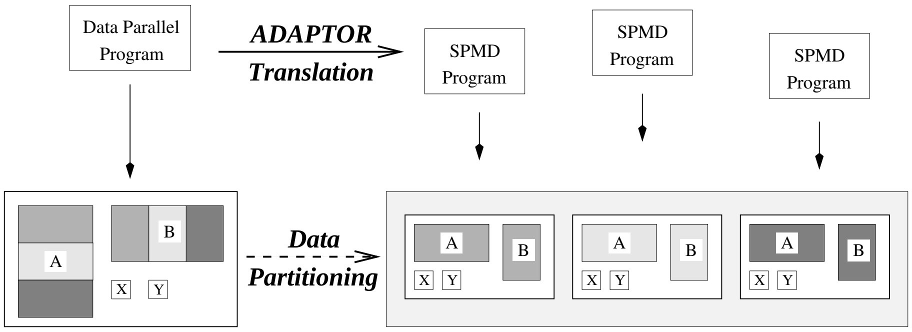

Figure 2: MIMD partitioning with ADAPTOR

Nearly for the whole source-to-source transformation ADAPTOR utilizes a toolbox for compiler construction [6].

2.3 The Runtime System

Beside the translation system, we have developed a runtime system called DALIB (distributed array library) that will be linked with the generated message passing program (see Figure 1). It consists of a lot of routines for

- managing processor arrays,
- defining template and array descriptors (distributions, alignments),
- low level communication (send, receive, wait, ...),
- high level communication (groups, broadcast, reduction, barrier, ...),
- updating overlap areas,
- data movements based on regular and irregular communication patterns,
- transposition, gather and scatter, circular shifting,
- timing functions and tracing facilities,
- and a parallel random number generator.

The DALIB is implemented in C. Most part of this library is portable between the different parallel machines. Beside the timing functions and the Fortran-C interface, only the message passing commands remain to be adapted to a given machine. By using a portable message passing layer like PVM [16] or MPI [14] these adaptation is no longer required.

2.4 Options for the Translation

ADAPTOR provides a lot of options to drive the translation. The most important options are:

- The user can choose between Connection Machine Fortran (CM Fortran) [17] and HPF as the input language. CM Fortran is the data parallel extension of Fortran by Thinking Machines, but only supported on their Connection Machines. As CM Fortran exists about four years longer than HPF, many industrial and scientific programs are already written in this language.
- ADAPTOR can also translate HPF or CM Fortran programs to serial FORTRAN 77 programs to be run on a single processor. This program has no communication and therefore it ought to be faster than the parallel one running on a single node.
- It is possible to generate a lot of protocol files giving important information about the different phases of the translation. Furthermore, the call graph for all the subprograms might give useful hints about the structure of the program.
- Usually, ADAPTOR operates in batch mode. In addition, we provide a graphical environment to use the tool interactively.

2.5 Availability

The source files of ADAPTOR, documentation files in PostScript and a number of example programs are available via 'anonymous ftp' from:

```
ftp.gmd.de        (129.26.8.84)
in subdirectory   gmd/adaptor
```

The total implementation has now about 95.000 lines of code: 23.000 lines C code for the DALIB and 72.000 lines for the source-to-source transformation, where 62.000 lines are compiler specifications that will be translated to C code by the compiler tools.

Due to the feedback of many users its functionality and reliability has increased step by step. It can also be used for translating huge application codes. In 1994 it has achieved the IBM ECCO Award (European Cluster and Parallel Computing Consortium Contest) in the category 'Software Development Tools'. ADAPTOR has been proven to deliver excellent results for HPF programs even now compared to hand-written message passing counterparts .

The latest version 3.0 supports almost the full HPF subset. We have left modules, derived types and pointers to the next version.

2.6 Related Work

Compilers for data parallel Fortran applications have been developed in the Superb [19], Kali [12], and Fortran 77D projects [11]. Though these systems extract the data parallelism from the sequential loops, their techniques influenced greatly the development of compilation technology for data parallel programming.

The ADAPT system from the University of Southampton [13], the data parallel C* compiler from Hatcher and Quinn [7], and the Fortran 90D/HPF compiler from the Syracuse University [2] are certainly most similar to the ADAPTOR system. Many of their ideas played an important role in the development of our HPF compilation system.

The first available commercial HPF compilers were the xHPF compiler from Applied Parallel Research [1], the `pghpf` compiler from the Portland Group [15] and the DEC Fortran 90 compiler. Other compilers have been announced or are now available.

A complete tool environment supporting HPF and other parallel programming paradigms has been developed at the CSCS in Manno [5].

3 The Importance of MPI for HPF Runtime Systems

Generally speaking, an HPF compiler should use the fastest communications available on any given machine. But there is definitively an essential conflict between portability and most efficient communication achievable on a specific hardware platform.

The new MPI standard [14] offers a portable layer with a very high functionality. First evaluations have shown that ADAPTOR can take advantage of processor groups, derived data types, and the different communication modes provided by MPI. Benefits for the user are given as soon as the MPI implementation on their hardware platform supports these features efficiently. With the development of more advanced HPF compilers new requirements for next MPI versions will become visible.

3.1 Low Level Communication

The DALIB has been realized in such a way that all high level communication routines are based on low level routines. Therefore only the low level routines have to be adapted.

Currently, we provide different versions of the DALIB runtime system that are based on one of the portable message passing systems PARMACS [8], PVM [16], or MPI [14]. There are also versions of the DALIB available that use directly the native communication routines of the parallel machine. Nevertheless, providing all the different versions of the DALIB requires a lot of effort and there is still a high potential for the optimization of the high level routines.

For the IBM SP 2 we can provide four versions of the DALIB based on the message passing systems POE 1.9, MPI-F 1.41, PVMe 3.1 and PARMACS 6.0. Table 1 shows the performance of some HPF applications running on a machine with 8 nodes. The results show that at least for this machine MPI provides currently the fastest communication. But as only low level routines are involved, we expect smaller differences for future versions.

MP system	POE 1.9	MPI-F 1.41	PVMe 3.1	PARMACS 6.0
matmul	224.7 MFlops	295.8 MFlops	263.7 MFlops	284.7 MFlops
lu decomp	114.0 MFlops	113.7 MFlops	91.4 MFlops	111.1 MFlops

Table 1: Performance of some applications on the IBM SP-2

Due to the high availability of PVM, most users of ADAPTOR have used the PVM version of the runtime system. In fact, the availability of PVM was of great help to make ADAPTOR more popular. But since HPF does not (currently) work well on heterogeneous systems, it has no need for all of the heterogeneous PVM support. Also support for task management is currently not necessary.

Now most manufactures have committed themselves to providing MPI as the standard communications. Furthermore, MPI is a standard (at least much more so than PVM), and it appears that the implementors are trying hard to keep different implementations compatible. This means that someday HPF compiler can switch to MPI for most of the relevant platforms and take advantage of all its functionality – thus increasing the performance and easing the maintenance costs.

3.2 Processor Groups and Collective Communication

Collective communications like broadcasts and global reduction operations are very important in an HPF runtime system. They will be used to realize intrinsic functions for reductions on distributed arrays or for the intrinsic function `SPREAD`.

A collective communication in an HPF application will not necessarily involve all processes. It might be restricted to the processes belonging to one abstract processor array or just to one dimension of such a processor array. Therefore, it is absolutely necessary to specify the group of processes that is involved in a collective communication.

Within DALIB own modules have been implemented that allow the definition of processor groups and collective communication between the processes of one group. These modules have a similar functionality as the corresponding routines of MPI. With MPI these modules become redundant.

3.3 Derived Datatypes

Usually, it is only possible to send and receive data that lies in a contiguous memory area. For sending and receiving noncontiguous data this data must be packed and unpacked explicitly.

If an array movement is split up into corresponding send and receive statements, subsections of distributed arrays will be exchanged between the processes. Subsections of arrays are usually noncontiguous data. Also unstructured data movements will require exchanging of noncontiguous data.

Therefore the efficient realization of packing and unpacking of data is an important issue of an HPF runtime system. It requires additional memory-to-memory copy operations at both sites.

The MPI standard provides derived datatypes. They allow to send and receive communication buffers that reside in noncontiguous memory areas. These datatypes can be defined in such a way that they match layout of array subsections and other records.

The following observations have been made using preliminary versions of MPI implementations:

- Most of the current implementations pack and unpack the data of a derived datatype. An advantage in performance will only be given if there is a communication engine with stride or scatter-gather capabilities or the implementation can pipeline packing and communication to reduce buffering.

- Derived datatypes are constructed with corresponding MPI functions. These datatypes can be reused and datatypes can also be combined to a new datatype. For this reason the datatypes are a very useful concept for implementing communication schedules within the runtime system that can be reused and combined.

An HPF runtime system can benefit extremely of efficiently implemented datatypes.

3.4 Nonblocking Communication

Nonblocking communication guarantees that communication and computation can be overlapped on systems where hardware allows such an overalp.

In an HPF program, the generated program differs between phases of communication (array movements) and phases of independent computations (local array assignments, parallel loops). Every communication phase consists of either sending or receiving messages. If the data of such a communication is not needed directly for the next computation, it might be better to use nonblocking communication.

The following approach will be implemented in ADAPTOR and has been proven to be useful:

- Every communication belongs to one communication phase that has a corresponding label.
- If the transferred data is not needed for the next computation, the phase will be executed in a nonblocking mode.
- The execution of the program waits for the completion of all communications from one phase before the computation that needs the data.
- As there might be outstanding communication of different communication phases, the label of the phase is used to identify the outstanding messages.

For some kind of communication phases it is very useful to take advantage of a special operation that combines the sending of a message and the receiving of another message in one command (e.g. circular shift operations). Such a send-receive operation can be used in a safe way so that cyclic dependences will not result in a deadlock. MPI supports also this operation. Tests on some hardware platforms have shown that this combined operation is also more efficient.

4 Interfacing HPF and Message Passing Programs

It is obvious that the data parallel programming paradigm is not sufficient for all kind of applications and that message passing programs are more efficient in many cases. Therefore it is desirable to combine both programming models. Especially the interface from HPF to existing libraries based on the efficient message passing paradigm is very important.

4.1 Extrinsic Procedures

HPF provides the `EXTRINSIC` mechanism to interface HPF with message passing programs. By the keyword `HPF_LOCAL` the compiler knows that in this procedure only the local part of the distributed arrays is visible.

The following code shows a solution of a matrix multiplication with an extrinsic subroutine. For this code the align directives guarantee that no communication is necessary as all data is local. But it might also be possible to exchange data with other processes by message passing commands.

```
      EXTRINSIC (HPF_LOCAL) SUBROUTINE MY_MATMUL (A, B, C)
      REAL, DIMENSION (:,:), INTENT (IN) :: A, B
      REAL, DIMENSION (:,:), INTENT (OUT) :: C
!HPF$ ALIGN A(I,J) WITH *C(I,*)
!HPF$ ALIGN B(I,J) WITH *C(*,J)
      INTEGER I, J

      DO I = LBOUND(C,1), UBOUND(C,1)
         DO J = LBOUND(C,2), UBOUND(C,2)
            C(I,J) = DOT_PRODUCT (A(I,:),B(:,J))
         END DO
      END DO
      RETURN
      END SUBROUTINE
```

HPF provides a lot of routines `GLOBAL_xxx` to get more information about the global view of the arrays. These routines give access to the information of the array descriptors internally used in the HPF runtime system.

4.2 Library Interface to ScaLAPACK

ScaLAPACK is a library of high performance linear algebra routines for distributed memory MIMD computers [4]. It contains routines for solving systems of linear equations, least squares and eigenvalues problems. One component of the ScaLAPACK software is the PBLAS (parallel BLAS) library whose interface is as similar to the BLAS as possible. These libraries operate on matrices distributed in a 2D block cyclic layout, the same distribution as incorporated in the HPF standard.

Here is the example to the BLAS double precision matrix multiplication routine `DGEMM` and the corresponding PBLAS routine `PDGEMM`.

```
CALL DGEMM (TRANSA, TRANSB,        CALL PDGEMM (TRANSA, TRANSB,
            M, N, K, ALPHA,                     M, N, K, ALPHA,
            A( IA, JA), LDA,                    A, IA, JA, DESC_A,
            B( IB, JB), LDB, BETA,              B, IB, JB, DESC_B, BETA
            C( IC, JC), LDC)                    C, IC, JC, DESC_C)
```

`DGEMM` computes `C = BETA*C + ALPHA*op(A)*op(B)` where the term `op(A)` is either A or its transpose depending on `TRANSA`, `op(B)` is similar, `op(A)` is $M \times K$, and `op(B)` is $K \times N$.

The interface to these libraries is rather complicated. The call to the parallel routine requires also descriptors specifying the layout of the arrays. The data structures and the data decomposition must be specified by the user, and it might be necessary to explicitly transform these structures in between calls to different library routines.

In the context of HPF, ADAPTOR provides an user-friendly high level interface to all the parallel routines.

```
   CALL HPF_DGEMM (TRANSA, TRANSB, ALPHA, A, B, BETA, C)
```

Currently, this high level interface is directly implemented in C using the functionality of the DALIB runtime system. The internal HPF array descriptors for A, B, and

C are used to set up the right call to the `PDGEMM` routine where also the correct shapes of the arguments are checked. This is possible as the ADAPTOR descriptors have all information about the global and local storage scheme.

Some problems arose by using two communication layers, one of the DALIB and the other one from the BLACS (basic linear algebra communication system [18]) used within ScaLAPACK. These problems will disappear as soon as the BLACS is implemented upon the communication system used in the DALIB.

With ADAPTOR, the user has a very user-friendly high-level interface to one of the most important parallel libraries.

5 Conclusions

Definitively HPF 1.1 is insufficient and the great challenge is its extension and applicability for more scientific applications, especially with irregular data structures. This is considered for the next version of HPF [10]. But all experiences with ADAPTOR have shown that the data parallel programming paradigm is a good approach for many scientific applications.

For the realization of the HPF runtime system using the MPI standard is definitively the best solution. Assuming an efficient MPI implementation on the given machine, the high functionality provides a good balance between portability and efficiency.

One other major advantage of HPF is the fact that HPF will allow a very user-friendly access to message passing libraries. These libraries might be written itself in HPF or can be well tuned hand-written message passing programs. The availability of libraries with solutions for many scientific problems might increase the user community of HPF dramatically. But it is absolutely necessary that HPF runtime system and the libraries are based on the same message passing layer.

Acknowledgements

Most thanks are due to my colleague Falk Zimmermann who has contributed significantly to the ADAPTOR tool. For the many valuable technical discussions regarding this paper I am indebted especially to Larry Meadows (Portland Group, Oregon), Bernard Tourancheau (LIP, ENS Lyon), and Frédéric Desprez (LaBRI, University of Bordeaux).

References

[1] FORGE 90. xHPF 1.0 Automatic Parallelizer for High Performance Fortran on Distributed Memory Systems - User's Guide. Technical report, Applied Parallel Research, Inc., April 1993.

[2] Z. Bozkus, A. Choudhary, G. Fox, T. Haupt, and S. Ranka. Fortran 90D/HPF Compiler for Distributed Memory MIMD Computers: Design, Implementation, and Performance Results. Technical Report, Syracuse Center for Computational Science, April 1993.

[3] Th. Brandes and F. Zimmermann. ADAPTOR - A Transformation Tool for HPF Programs. In K.M. Decker and R.M. Rehmann, editors, *Programming Environments for Massively Parallel Distributed Systems*, pages 91–96. Birkhäuser Verlag, April 1994.

[4] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, A. Ostrouchov, S. Petitet, K. Stanley, D. Walker, and R.C. Whaley. ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers - Design Issues and Performance. Technical Report LAPACK Working Note 95, Department of Computer Science, University of Tennessee, 1995.

[5] K. M. Decker, J. J. Dvorak, and R. M. Rehmann. A knowledge-based scientific parallel programming environment. In K.M. Decker and R.M. Rehmann, editors, *Programming Environments for Massively Parallel Distributed Systems*, pages 127–138. Birkhäuser Verlag, April 1994.

[6] J. Grosch and H. Emmelmann. A Tool Box for Compiler Construction. *Lecture Notes of Computer Science*, 477:106–116, October 1990.

[7] P. J. Hatcher and M. J. Quinn. *Data-Parallel Programming on MIMD Computers*. Science and Engineering Computation Series. MIT Press, Cambridge, Massachusets, 1991.

[8] R. Hempel, H.C. Hoppe, U. Keller, and W. Krotz. PARMACS V6.0 Specification. Interface description, Pallas GmbH, Brühl, November 1993.

[9] High Perforamnce Fortran Forum. High Performance Fortran Language Specification. Version 1.1, Department of Computer Science, Rice University, November 1994.

[10] High Perforamnce Fortran Forum. HPF-2 Scope of Activities and Motivating Applications. Version 0.8, Department of Computer Science, Rice University, November 1994.

[11] S. Hiranandani, K. Kennedy, and C.-W. Tseng. Evaluating Compiler Optimizations for Fortran D. *Journal of Parallel and Distributed Computing*, 21:27–45, April 1994.

[12] C. Koelbel and P. Mehrotra. Compiling global name-space parallel loops for distributed execution. *IEEE Transactions on Parallel and Distributed Systems*, October 1991.

[13] J. Merlin. ADAPTing Fortran 90 Array Programs for Distributed Memory Architectures. In *Proc. 1st International Conference of the Austrian Center for Parallel Computation*, Salzburg, October 1991.

[14] Message Passing Interface Forum. Document for a Standard Message-Passing Interface. Draft Version February 1994 1.0, University of Tennessee, Knoxville, February 1994.

[15] PGHPF. Reference Manual, User's Guide. Technical report, The Portland Group, Inc., November 1994.

[16] V. Sunderam. PVM: a Framework for Parallel Distributed Computing. *Concurrency: Practice and Experience*, 3(10), December 1990.

[17] Thinking Machines Corporation. CM Fortran Programming Guide, Version 1.0. Manual, TMC, January 1991.

[18] R. C. Whaley. Basic Linear Algebra Communication Subprograms: Analysis and Implementation Across Multiple Parallel Architectures. LAPACK Working Note 73, Technical Report CS-94-234, University of Tennessee, May 1994.

[19] H. Zima, H. Bast, and M. Gerndt. Superb: A Tool for Semi-Automatic SIMD/MIMD Parallelizatin. *Parallel Computing*, January 1988.