Published in Fortran Journal, pp. 2–6, March/April 1996.

`shpf`: a Subset High Performance Fortran compilation system

John Merlin, Bryan Carpenter*and Tony Hey†
Department of Electronics and Computer Science,
University of Southampton,
Southampton, SO17 1BJ, UK.

email: `jhm@ecs.soton.ac.uk`

February 1, 1996

Abstract

`shpf` is a public domain Subset High Performance Fortran compilation system. This paper gives a brief overview of its structure, the language it supports, and the optimisations that are currently implemented and planned. It also contains details of how to obtain `shpf` and a related tool, `ida`, which is an inter-procedural analyser that can help users convert existing programs into efficient HPF programs.

1 Introduction

'`shpf`' is a public domain Subset High Performance Fortran (HPF) compilation system. It must be used in conjunction with a Standard Message-passing Interface (MPI, [1]) implementation and a Fortran 90 compiler, and is portable to any computer or computer network that has these. It has been developed at Southampton University by enhancing the earlier ADAPT system [2, 3], which originally accepted Fortran 90 supplemented by *ad hoc* data distribution extensions, to HPF.

`shpf` was released to a number of test sites in June 1995 and is now publically available by anonymous ftp. Now that the initial version of the system is finished, we intend to use it as a testbed for the research and development of advanced optimisations for HPF and for the implementation of new language features.

This paper gives a brief overview of `shpf`, the language it supports, and the optimisations that are currently implemented and planned. It also contains details of how to obtain `shpf` and a related tool, `ida`, which is an inter-procedural analyser for Fortran programs that can help users convert existing programs into efficient HPF programs.

*Present address: NPAC, Syracuse University, 111 College Place, Syracuse, New York 13244-4100, USA. Email: `dbc@npac.syr.edu`.

†We gratefully acknowledge support from the Engineering and Physical Sciences Research Council under grant number GR/J89507, from the JISC New Technology Initiative, and from NAG Ltd.

2 Overview of shpf

Figure 1 shows the structure of **shpf**. The **shpf** release package contains two main components: a translator, 'ADAPT', and a runtime library, 'ADLIB'. The translator transforms a Subset HPF program into an SPMD (Single Program Multiple Data) program that runs on each node or process of the target computer. The latter program is in standard Fortran 90 and can be compiled by a normal Fortran 90 compiler for the target machine. In it, declarations for distributed arrays are transformed into declarations for just the locally-stored segments of the arrays, and operations and assignments are restricted to locally stored data. Accesses to non-local data are achieved by calls to communications routines in the ADLIB runtime library. These routines perform high-level, array-based collective communications rather than low-level point-to-point communications. Further details of ADLIB are given in Section 5, and [3] gives details of the translation performed by ADAPT.

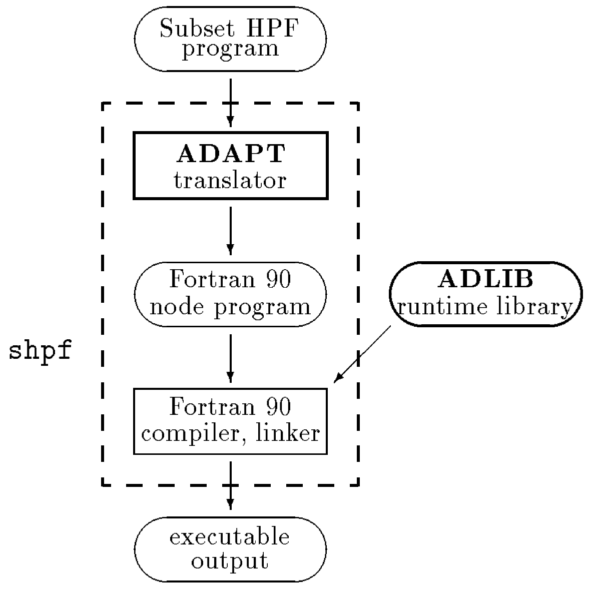

Figure 1: The **shpf** compilation system

The **shpf** release package provides the source code for ADAPT and ADLIB. ADAPT is written in C, and ADLIB in C++ with communications implemented in the standard message-passing interface, MPI [1]. Therefore **shpf** can be used on any platform that has a C++ compiler and an MPI installation—to compile and run ADLIB—and a Fortran 90 compiler to compile the output of the translator. Public domain implementations of C++ and MPI are available, as detailed in the installation notes.

The system is used via two Unix scripts, **shpf** and **hpfrun**, which provide commands for respectively compiling and linking, and running, HPF programs. The **shpf** command has been designed to be similar to commands for standard Unix compilers. For example:

```
shpf -o md md.hpf timer.hpf
```

compiles and links the HPF program contained in source files **md.hpf** and **timer.hpf**, and writes the executable output to file **md**.

```
hpfrun 4 md
```

runs it on 4 processors. A Unix 'manual page' is provided for the `shpf` command.

3 Language supported by `shpf`

`shpf` accepts the official 'Subset HPF' language, as defined in Section 8 of [4], with a few restrictions that are listed in the release notes [5]. It recognises *full* HPF, including full Fortran 90, and gives error messages upon encountering unsupported features.

From the outset, a primary objective was to implement Subset HPF's data mapping features (i.e. alignment and distribution) in their full generality, and this has been fairly well achieved. For example, there are no restrictions on the number of levels or the complexity of alignments; all distribution formats are supported, in any number of data and processor array dimensions (up to the Fortran limit of 7); distributed data can be used in any context in executable statements, including subscript expressions; any number of levels of indirect addressing, or equivalently vector subscripting, are allowed; regular and irregular sections of arrays can be passed as arguments to procedures; and dummy arguments can have a different distribution from the corresponding actual arguments.

Parallel execution is obtained from the use of Fortran 90 array syntax, that is, array expressions and assignments, array external and intrinsic functions, and the `WHERE` statement and construct. Shortly we will also implement user-defined `ELEMENTAL` procedures, allowing functional as well as data parallelism—see Section 9. `DO`-loops are *not* parallelised. Some speedup may be observed for `DO`-loops enclosing assignments to distributed array elements, but this is 'incidental' parallelism, due to the fact that iterations do not perform any assignment on processors that do not store the element(s) being assigned. The HPF `FORALL` statement is supported but is sequentialised in the current version.

Foremost among the language features *not* supported are sequence and storage associations, namely the `EQUIVALENCE` statement, partitioning a common block differently in different program units, reshaping or resizing arrays across procedure boundaries, and associating an array element actual argument with a dummy array. We have neglected these features because we do not believe that they can be efficiently supported, in general, for distributed data in HPF.

4 Converting 'dusty deck' Fortran programs for `shpf`

As we have indicated, an HPF program that is to be compiled by `shpf` should use array syntax as much as possible in order to maximise its exploitation of data parallelism, and furthermore it must not contain any sequence or storage associations. It is evident that to convert a 'dusty-deck' Fortran program into efficient HPF will often require some extra rewriting beyond the mere addition of HPF data mapping directives. Fortunately the required modifications are 'clean', in that they should make the code clearer and perhaps also more concise, as well as improving its performance.

However, the conversion task is considerably complicated by the need to analyse the program globally (i.e. across procedure boundaries) for such purposes as finding and removing sequence and storage associations, determining efficient data mappings, etc.

As a by-product of developing `shpf` we have also developed a tool, `ida`, which can greatly assist this task. It is an inter-procedural analyser for Fortran programs which provides information such as call graphs, traces of variables, common block partitioning and usage, and procedure references and argument associations. It accepts the same input language as **shpf**. It is described in [6], which also gives a case study of its use in converting a 1600 line Fortran 77 conjugate gradient solver to HPF.

Like **shpf**, **ida** is available in the public domain, as described in Section 7.

5 ADLIB

The runtime library, ADLIB, provides high-level, array-based collective communications. It is a C++ class library with communications implemented in the standard message-passing interface, MPI [1]. As well as providing HPF runtime support, it can also be used directly for distributed data parallel programming in C++. [7] describes its class structure and functions. Here we shall briefly describe its Fortran 90 interface, via which it is invoked by the output programs generated by ADAPT.

All data communications, except for those implicit in array intrinsic functions, are implemented by one of 3 subroutines: `AD_REMAP`, `AD_GATHER` and `AD_SCATTER`.[1].

`AD_REMAP` copies a regular section of an array to a regular section of another array, where the input and output arrays may have any HPF mapping (i.e. alignment and distribution). For example, if the following array assignment requires data communications:

```
    A (L1:U1:S1, 10, L2:U2) = B (:, L3:U3)
```

then it is translated to this ADLIB call in the output program:

```
    CALL AD_REMAP (A, A_MAPVEC, B, B_MAPVEC)
```

`A_MAPVEC` and `B_MAPVEC` are 'mapping vectors', which completely describe the HPF mapping of a whole array or a regular section. `AD_REMAP` also handles broadcasts and communications of scalar values, including array elements, which are special cases.

`AD_GATHER` and `AD_SCATTER` copy an *irregular* section of an array (that is, an array reference with vector subscript(s)) to a *regular* section, or vice versa, where the input and output arrays may again have any HPF mapping. For example, if `VEC1` and `VEC2` are vectors, then:

```
    A (L1:U1:S2, :) = B (VEC1, VEC2)
```

is translated to:

```
    CALL AD_GATHER (A, A_MAPVEC, B, B_MAPVEC,                 &
                    VEC1, VEC1_MAPVEC, VEC2, VEC2_MAPVEC)
```

Incidentally, these routines require that the mapping of each vector subscript argument matches that of the array dimension it subscripts, which the translator arranges by remapping if necessary.

ADLIB also provides distributed versions of the Fortran 90 array intrinsic functions, for example `MATMUL`, `DOT_PRODUCT`, `SUM`, `ALL`, `MAXVAL`, `SHIFT`, `SPREAD`, etc, which in general accept arguments that are regular sections of arrays with any HPF mapping. Finally, ADLIB also contains routines for initialisation and cleanup, etc.

Currently there are no special routines for I/O, so all of the data in an I/O statement are buffered on the processor performing the I/O. This is expensive in memory usage, and we hope to improve upon it in a later version.

[1]In fact there is a separate entry point to each of these routines for every possible argument datatype, which is required to satisfy strict Fortran 90 argument type checking.

6 Current optimisations in `shpf`

In developing the initial version of **`shpf`** we have concentrated mainly on functionality rather than optimisation. However, we have already implemented some basic but important optimisations.

Foremost among these is that the translator always generates in-line expressions, rather than external runtime library calls, for locality tests and for converting subscripts to local subscripts with respect to the locally-stored array segment. Furthermore, the translator performs extensive expression simplification in these contexts in order to make these expressions as simple as possible. Tests show that this is a very important optimisation. For example, for a particular test program in which the timed region only makes local accesses to distributed arrays (that is, it involves no communications), **`shpf`** generates code that is between 12 and 130 times faster than that of two other HPF systems with which it was compared [8]. Both of the other systems use external library calls for subscript conversion and locality tests, and the observed performance difference is wholly attributable to this factor.

Other 'basic' optimisations that have been implemented recently include better dependence testing to avoid the introduction of unnecessary intermediate temporaries in assignments, and the elimination of duplicate 'mapping vectors'. [8] gives some evaluation results obtained prior to implementing the last 2 optimisations.

7 Availability of `shpf` and `ida`

`shpf` can be obtained by anonymous ftp from:

`ftp.ccg.ecs.soton.ac.uk`, in directory **`/pub/packages/shpf`**

or via the World Wide Web from:

`http://www.ccg.ecs.soton.ac.uk/shpf/shpf.html`

or by email from **`jhm@ecs.soton.ac.uk`**. The release package contains the source code for ADAPT and ADLIB, Unix scripts **`shpf`** and **`hpfrun`**, a Unix 'manual page' for **`shpf`**, installation and release notes, and a large set of HPF example and test programs.

`ida` can be obtained from the same ftp site, in directory **`/pub/packages/ida`**, where the tool, a user guide and example programs are located. It can also be obtained from the above World Wide Web and email addresses.

8 Related work

Much work has been done in the area of developing and compiling HPF-like languages for parallel computers, and there is insufficient space here to give a comprehensive survey of it. [9] surveys some of the important projects in this area.

The research systems most closely related to **`shpf`**, in that they have concentrated on exploiting Fortran 90 features such as array syntax, are ADAPTOR from GMD [10] and the Fortran 90D/HPF system from Syracuse University [11]. These and other projects have contributed important ideas and inspiration to this work.

9 Conclusions and future work

The **shpf** system that we have described here provides fairly full coverage of the Subset HPF language, and in particular supports completely general HPF alignments and distributions. Preliminary evaluations suggest that it generally delivers comparable performance to other systems with which it has been compared, and in some respects it is relatively highly optimised [8]. It is quick and easy to use, and gives good error and warning messages.

shpf has been extensively tested. Furthermore, ADAPT's front end and much of its analysis code are also incorporated in **ida**, which has been extensively evaluated and has been used successfully on a Fortran 77 program containing hundreds of thousands of lines of source code spread over hundreds of files [6]. For these reasons we believe that **shpf** is robust and reliable.

In the near future we will extend **shpf**'s functionality to support user-defined **ELEMENTAL** procedures, which will be introduced in the next revision of Fortran, expected in 1996. These provide a way to express functional parallelism within array expressions, and will allow more efficient implementations of some applications, for example **md1** in the **shpf** release package.

Future research will focus on the development of 'advanced' optimisations for HPF, in particular the implementation of general (i.e. non-rectangular) **FORALL**s in parallel, and the development of better heuristics than the 'owner computes' rule for the placement of expression evaluation.

Acknowledgements

We would like to thank Yew Bie Cheng for assisting with the testing of **shpf**, and everybody who has used it, in particular Adam Marshall of Liverpool University who has given us valuable and encouraging feedback.

References

[1] Message Passing Interface Forum. Document for a standard message-passing interface, version 1.0. University of Tennessee, Knoxville, February 1994.

[2] J.H. Merlin. ADAPTing Fortran 90 array programs for distributed memory architectures. In Hans P. Zima, editor, *Parallel Computation: Proc. of 1st Int'l Conf. of the Austrian Centre for Parallel Computation*, pp. 184–200, Salzburg, Austria, Sept 30–Oct 2 1991. Springer-Verlag. Lecture Notes in Computer Science, 591.

[3] J.H. Merlin. Techniques for the automatic parallelisation of 'Distributed Fortran 90'. Technical report SNARC 92-02, Dept. of Electronics and Computer Science, University of Southampton, Southampton, SO17 1BJ, U.K., November 1991.

[4] High Performance Fortran Forum. High Performance Fortran Language Specification. *Scientific Programming*, 2(1):1–170, 1993.

[5] D.B. Carpenter and J.H. Merlin. shpf *version 0.9: Release Notes.* Dept. of Electronics and Computer Science, University of Southampton, November 1995. Available on the World Wide Web at `http://www.ccg.ecs.soton.ac.uk/shpf/docs/release.ps`.

[6] J.H. Merlin and J.S. Reeve. IDA—an aid to the parallelisation of Fortran codes. Submitted to *Parallel Computing*, 1995.

[7] D.B. Carpenter. Adlib: A distributed array library to support HPF translation. Presented at the 5th Int'l Workshop on Compilers for Parallel Computers, University of Malaga, Malaga, Spain, June 28–30 1995. Available on the World Wide Web at `http://www.ccg.ecs.soton.ac.uk/shpf/hpf-workshop/adlib.ps`.

[8] A.C. Marshall. HPF activities at Liverpool University. Talk presented at the HPF workshop, Southampton, October 1995. Available on the World Wide Web at `http://www.ccg.ecs.soton.ac.uk/shpf/hpf-workshop/adamm.ps`.

[9] P. Crooks and R.H. Perrott. Language constructs for data partitioning and distribution. *Scientific Programming*, 4:59–85, 1995.

[10] T. 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*, pp. 91–96. Birkhäuser Verlag, April 1994.

[11] M.-Y. Wu and G. Fox. Fortran 90D compiler for distributed memory MIMD parallel computers. Tech. report SCCS-88b, Syracuse Center for Computational Science, Syracuse University, 111 College Place, Syracuse, NY 13244-4100, July 1991.