Evaluation of High Performance Fortran through Application Kernels

Hon W. Yau
Syracuse University, Northeast Parallel Architectures Center

Geoffrey C. Fox
Syracuse University, Northeast Parallel Architectures Center

Ken Hawick
Syracuse University, Northeast Parallel Architectures Center

Proc. HPCN, Vienna 1997, Technical Note DHPC-005

Evaluation of High Performance Fortran through Application Kernels

H W Yau[*], G C Fox and K A Hawick[**]

Northeast Parallel Architectures Center
Syracuse University
111 College Place
Syracuse NY 13244-4100
USA
E-mail: `hwyau@epcc.ed.ac.uk` & `gcf@npac.syr.edu` &
`khawick@cs.adelaide.edu.au`
Phone: +44 131 650 5957
Fax: +44 131 650 6555
URL: `http://www.npac.syr.edu/hpfa/`

Abstract. Since the definition of the High Performance Fortran (HPF) standard, we have been maintaining a suite of application kernel codes with the aim of using them to evaluate the available compilers. This paper presents the results and conclusions from this study, for sixteen codes, on compilers from IBM, DEC, and the Portland Group Inc. (PGI), and on three machines: a DEC Alphafarm, an IBM SP-2, and a Cray T3D. From this, we hope to show the prospective HPF user that scalable performance is possible with modest effort, yet also where the current weaknesses lay.

1 Introduction

In this paper, we shall first motivate the use of the High Performance Fortran language as a means of exploiting the parallelism within a program. We shall then clarify the purpose of the NPAC HPF Applications suite, and explain the methodology by which these code have been benchmarked. In essence, we shall be comparing the performance of the codes against the ideal of a perfectly scaling code with no overhead from the use of the HPF language. Finally, a discussion will be made on how near or far the compilers and code tested are in meeting this aggressive standard, and the possible reasons why.

2 The High Performance Fortran Language

High Performance Fortran is a definition agreed on by vendors and users on exploiting the data parallelism already implicit in the Fortran 90 language. The aim is to provide additional constructs with which the user and compiler can produce a scalable executable, with performances comparable to hand-tuned message passing code. The principle means by which this is achieved is through the use of compiler directives, statements which a traditional Fortran compiler would ignore as a commented line, but which an HPF compiler would use to ascertain how data arrays

[*] Current address: Edinburgh Parallel Computing Centre, University of Edinburgh, Edinburgh EH9–3JZ, Scotland/UK.
[**] Current address: Department of Computer Science, University of Adelaide, South Australia 5005, Australia.

are to be distributed and how the code may be executed in parallel [THPFF94]. In addition, HPF has introduced several new features into the Fortran language; the most obvious of which are new intrinsic functions (mostly through the `HPF_LIBRARY` module) and the `FORALL` statement/construct for more generalized array expressions than are possible with Fortran 90 array syntax and `WHERE` statement/construct. It is interesting to note that at present it appears that some of these HPF language features are currently being considered for inclusion in the forthcoming Fortran 95 standard.

The HPF approach has several strengths as a means of parallelizing existing code [RYHF96]. First of all, the HPF computation model has been defined so as to have a single execution thread. Whilst this does present problems of efficiency in ensuring all non-distributed data are kept identical across all processors, it also means that any changes from HPF statements are *by definition* benign with respect to the code's behaviour when compiled for one or for many processors. This should be contrasted with the message-passing case where it is more usual to have separate parallel and serial versions, which the developer is then obliged to individually maintain.

The other major attribute of HPF is that it is a standard, designed and agreed on by major vendors and users. This protection of software investment means that, like the MPI standard for message passing [TMPIF94], users can compile the same code for different platforms ranging from a single workstation to a dedicated massively parallel processing machine. Whilst the idea of data-parallel languages have been discussed since the late 1960's[Ric95], HPF is the first —and thus far only— portable standard available for consideration.

3 The High Performance Fortran Applications Suite

The NPAC HPF Applications (HPFA) suite is a set of programs collected and developed over a number of years to provide feedback on available HPF compilers. From this, one would be able to provide quantitative details on the strengths and weaknesses of these compilers.

All the HPFA kernel suite of codes benchmarked have had one of two origins: they were either ports from existing Fortran programs, or they were written from scratch. Where the codes were originally Fortran 77 this usually required extensive rewriting to make use of the HPF array data parallelism syntax. However, where the codes originated from implementations for machines such as the MasPar or the Thinking Machines CM series, the work required was usually a simple one-to-one replacement of function names or language feature.

The codes benchmarked for this paper, and the language features & intrinsic functions which they exploit to express their parallelism are:

1. Solution of 2-dimensional Poisson equation by the alternating direct implicit (ADI) method: array syntax, `TRANSPOSE()`.
2. 2-dimensional fast Fourier transformation: `INDEPENDENT` do loops, passing of array sections into subroutines, `TRANSPOSE()`.
3. Rewritten HPF version of the NASA NAS embarrassingly parallel benchmark [BBL*Eds*93]: `WHERE`, array syntax.
4. 2-dimensional convolution: `INDEPENDENT` do loops, passing of array sections into subroutines, `TRANSPOSE()`.
5. Generation of random numbers with Gaussian deviates by the Box-Muller algorithm: `FORALL`, `MERGE()E`, array syntax.
6. 2-dimensional spanning percolation: `FORALL`, `CSHIFT()`, `WHERE`, `MERGE()`, array syntax.
7. Q-state Potts model simulation: `MERGE()`, `CSHIFT()`, `FORALL`, array syntax.

8. Solution of the Cahn-Hilliard-Cook field equation: `CSHIFT()`, array syntax.
9. Gaussian elimination with partial pivoting: `FORALL`, `SUM()`, `MAXLOC()`, array syntax.
10. Direct N-Body simulation: `CSHIFT()`, array syntax.
11. Bubble sort algorithm: `WHERE`, `EOSHIFT()`, array syntax.
12. Wavelet image processing: `FORALL`, array syntax.
13. Binomial stochastic options pricing simulation: `EOSHIFT()`, `WHERE`, `SUM`, array syntax.
14. Cholesky factorization: `FORALL`, `SPREAD()`, `SUM()`, array syntax.
15. Hough image transformation: `COUNT_SCATTER()`, `FORALL`, array syntax.
16. Hopfield neural network simulation: `MATMUL()`, `MAXVAL`, `DOT_PRODUCT`, `WHERE`, array syntax.

Where each code is typically under 500 lines in length. The distribution of the arrays for most of these problems are along one dimension, and typically block or cyclic-1. Where distributed arrays are passed into subroutines, descriptive mapping is used to assure the compiler of the correct data distribution. The reasons for these somewhat conservative decisions are largely historical, when it was felt complete and efficient HPF implementation would not have been immediately available. In a similar vein, whilst the intent is to cover as wide a range of different applications as is feasible, a balance had to be made in using codes which could be parallelized, and which would fit into the present HPF regular data framework.

These codes are also available from the NPAC website, for use by anyone to test their HPF implementation [NPA96].

3.1 Compilers and Platforms

For the benchmarking, the following compilers and platforms configurations were available, executing on 1,2,4 and 8 processors.

- Portland Group Inc. PGHPF v2.1-1 compiler on an IBM SP-2, installed July 1996. This is a largely complete HPF implementation, and none of the missing features had any impact on the HPFA codes.
- IBM XLHPF v1.1 compiler on an IBM SP-2, installed March 1996. This is an implementation of the subset specification of HPF, plus some other features.
- DEC Fortran 90/POE v4.0 compiler on a DEC Alphafarm connected via a Gigaswitch, installed on February 1996. This is a full implementation of the HPF language, albeit with certain parallelism features disabled.
- PGI PGHPF v2.1 compiler on a Cray T3D, installed June 1996. As for the IBM implementation, this is a largely complete HPF implementation.

In addition, Fortran 90 single processor runs were made so as to ascertain the additional overhead of using HPF on each of these machines:

- PGI PGHPF on the IBM SP-2 without the '`-pghpf`' execution flag, for comparisons with the PGI-PGHPF IBM SP-2 runs.
- IBM XLF90 for comparisons with the IBM XLHPF runs.
- DEC Fortran 90 without the '`-wsf`' parallel software environment compiler flag, for comparisons with the DEC HPF runs.
- Cray CF90 for comparisons with the PGI-PGHPF Cray T3D runs.

All three machines examined are distributed memory multiprocessor machines. It is generally expected that the HPF language will perform more closely to hand-written message passing codes for shared memory (or virtual-shared memory, as in the case of the Hewlett-Packard/Convex Exemplar series) memory architecture machines.

In all cases, at least eight timings were made at each configuration, and the minimum execution times used. These timings referred to the wall-clock execution time, as provided by the Fortran 90 '`SYSTEM_CLOCK()`' intrinsic function.

The hardware for the IBM SP-2 runs were made courtesy of the Cornell Theory Center, the DEC Alphafarm via the Northeast Parallel Architectures Center, and the Cray T3D via the Edinburgh Parallel Computing Centre.

4 Benchmark Results

The results presented are an attempt to show each code's behaviour with respect to the number of participating processors. The information we wish to extract are the overhead induced by the use of HPF over that from an Fortran 90 execution on a single processor, and the subsequent scaling in the execution times. In addition, timing calls have been inserted into the codes so as to determine the times spent on purely computational tasks and on combined communication & computation —the latter as it is sometimes impossible to separate the times spent on communications and computation *within* a HPF program statement or intrinsic function. The graphical profiler from the PGI compiler was used to determine the parts of the code which contain communications, and the observations fed back into the programs by inserting explicit calls to timing routines around the areas of interest. This methodology obviously suffers from extrapolating the PGI implementation to those from the other vendors, but since we have not seen any obvious inaccuracies in the PGI profiler's report on which lines of code are dependent on communications, we believe this indeed provides a realistic picture.

From these data, it would be possible to indirectly determine the performance of these codes compared to the (usually unobtainable) ideal situation of:

– No difference in execution times between the serial Fortran 90 and the HPF code on one processor.
– The reciprocal of the execution times scale down linearly with the number of processors.

Within the parallel computing community, the question often asked is how an HPF code compares with a functionally equivalent hand-coded message passing version. Writing —and presumably optimizing— message passing calls into the eighteen codes in this study would be the ideal means by which to answer this question. However, this was deemed infeasible under the available timeframe and instead the results presented here will compare the performance with the ideal situation listed above as the metric on 'how good' were the compilers tested.

Table 1 shows the speed-up figures for the HPF implementations by PGI and IBM on an IBM SP-2. Speed-up here is defined as the execution time taken by a particular configuration divided by the time taken by the one-processor HPF execution time. As a guide to the overhead of using HPF, this is also done for the Fortran 90 version of the code.

An identical exercise is done in table 2 for the HPF implementations by DEC on an eight workstation Alphafarm, and PGI on a Cray T3D. It should be noted that the Cray T3D Fortran 90 runs were performed with the Cray Fortran 90 compiler, rather than the PGI product; mainly because unlike the case with the IBM SP-2, it was not immediately obvious how to 'switch off' the HPF features of the PGI compiler on the T3D.

Finally, table 3 gives the wall clock execution times of the four configurations examined on the sixteen HPFA codes, on a single processor running the HPF code.

HPFA Code	PGHPF & IBM SP-2		XLHPF & IBM SP-2	
	PGHPF	PGHPF	XLF90	XLHPF
0001 ADI	1.6	1.4,1.9,2.4	1.3	1.9,3.6,6.8
0003 2D FFT	0.5	2.0,3.9,7.6	1.6	1.0,0.9,0.8
0004 NAS EP	1.0	2.3,4.7,9.0	0.9	2.1,4.2,8.3
0008 2D Convolution	0.5	2.0,3.8,7.5	1.7	1.0,0.9,0.8
0009 Box-Muller	1.0	2.0,2.2,4.9	1.0	1.7,2.8,3.9
0011 2D-Percolation	1.5	1.8,1.5,2.2	1.3	1.7,2.6,3.5
0013 Potts Model	1.0	1.8,2.9,3.9	–	–
0014 Cahn-Hilliard	1.7	2.0,3.8,7.2	1.6	2.0,3.9,7.5
0022 Gaussian Factor	0.3	1.8,3.1,3.9	2.5	1.9,3.3,5.1
0025 Direct N-Body	1.4	2.0,3.8,7.3	0.9	2.0,3.8,7.1
0039 Bubble Sort	0.9	1.9,3.4,5.4	1.0	1.7,3.1,5.5
0041 Wavelet	1.0	2.0,3.7,7.3	1.4	2.0,3.9,7.8
0048 Options Pricing	1.0	2.0,3.8,7.2	–	–
0049 Cholesky Factor	1.2	1.9,4.3,8.3	1.2	1.9,3.8,7.0
0052 Hough Transform	1.8	1.6,2.5,3.7	–	–
0053 Hopfield Network	1.1	0.9,0.6,0.4	0.8	0.5,0.7,0.8

Table 1. Speed-up results for the PGI PGHPF and IBM XLHPF/XLF90 compilers on the IBM SP-2. The numbers presented here are speed-ups with respect to the one-processor HPF codes, for the Fortran 90 serial run, and the 2,4 & 8 processors HPF runs; by definition the speed-up for the one processor HPF runs is '1.0'. The configuration(s) which gave the best speed-up has been highlighted. The dash represents where the HPF compiler was unable to compile the code, whether due to documented limitations or due to unknown compilation errors.

These provide an indication of the spread in execution times amongst the different products.

The following subsections §4.1–4.4 will describe the behaviour of the HPFA codes on the compiler and hardware configurations listed in §3.1, as well as elaborating on the results given in tables 1–3.

4.1 PGI PGHPF Timings on the IBM SP-2

Of the sixteen codes examined, eight displayed a speed-up of 7.0 or higher at eight processors, with six also having a low Fortran 90 to HPF overhead. Moreover, two of the codes had a speed-up higher than 8.0, due to better cache hits with the smaller problem size given to each processor. From these results, one may infer that the following characteristics are implemented well by the PGHPF compiler on the IBM SP-2: `INDEPENDENT` do loops, `WHERE` mask operations with no communications, simple near-neighbour `CSHIFT()` operations, and the `SUM()`, `TRANSPOSE()` & `SPREAD()` functions. On the otherhand, three codes performed noticeably badly. The features which appear to have caused problems are: masked `CSHIFT()` operations with communications, and `MATMUL()` with communications.

On the whole, this configuration performs well on codes with little or interprocessor communications. Although there is the exception of the ADI code, which being mostly embarrassingly parallel, should also have scaled well.

4.2 IBM XLHPF & XLF90 Timings on the IBM SP-2

The IBM XLHPF is a subset-HPF compiler, and the major effect on the HPFA codes have been the inability to make use of the '`HPF_LIBRARY`' intrinsic functions,

HPFA Code	DEC HPF & Alphafarm		PGHPF & Cray T3D	
	F90	F90 & WSF	CF90	PGHPF
0001 ADI	1.5	0.3,0.5,0.9	1.3	1.3,1.5,1.6
0003 2D FFT	1.3	1.0,0.9,0.5	4.0	2.0,4.0,7.8
0004 NAS EP	1.4	0.1,0.6,0.6	0.6	2.0,3.9,7.0
0008 2D Convolution	0.6	0.9,0.8,0.5	3.6	2.0,4.0,7.9
0009 Box-Muller	1.2	0.4,0.4,1.2	1.2	2.0,4.0,7.7
0011 2D-Percolation	1.1	0.9,0.6,1.2	0.8	1.7,2.5,2.7
0013 Potts Model	0.7	0.9,1.3,1.3	0.8	1.5,1.6,1.3
0014 Cahn-Hilliard	1.8	1.0,2.0,3.7	1.1	2.0,3.6,5.9
0022 Gaussian Factor	0.6	0.5,0.6,0.5	1.5	1.6,1.9,1.3
0025 Direct N-Body	0.9	0.1,0.2,0.4	1.2	1.9,3.0,3.8
0039 Bubble Sort	0.8	0.2,0.4,0.6	0.9	1.3,1.9,1.7
0041 Wavelet	1.2	0.4,0.7,1.4	0.7	2.0,3.9,7.5
0048 Options Pricing	0.9	0.8,1.6,2.7	0.7	1.9,3.4,5.0
0049 Cholesky Factor	1.4	1.5,2.6,4.2	0.8	1.9,3.2,4.5
0052 Hough Transform	11.0	0.3,0.5,0.9	4.6	1.4,1.9,3.1
0053 Hopfield Network	3.5	0.3,0.3,0.3	0.6	1.1,1.1,0.6

Table 2. Speed-up results for the DEC F90 & HPF on an eight-processor Alphafarm and Cray CF90 & PGI PGHPF on a Cray T3D. The numbers presented here are speed-ups with respect to the one-processor HPF codes, for the Fortran 90 serial run, and the 2,4 & 8 processors HPF runs; by definition the speed-up for the one processor HPF runs is '1.0'. The configuration(s) which gave the best speed-up for each compiler-machine have been highlighted.

as used by the Hough transformation code. In addition, it was found that two other codes caused unknown compile-time errors.

Of the remaining thirteen codes, five codes had a speed-up of 7.0 or better at eight processors, with two codes also having a low Fortran 90 to HPF overheads. The features which the XLHPF compiler implemented well appear to be: `WHERE` mask operations with no communications, simple near-neighbour `CSHIFT()`, and the `SUM()`, `TRANSPOSE()` and `SPREAD()` intrinsics. The features which performed badly are: `INDEPENDENT` do loops which called pure subroutines with array sections, and `MATMUL()` with interprocessor communications.

As with the PGI PGHPF on the IBM SP-2, the IBM XLHPF compiler appear to perform best on embarrassingly parallel problems —it notably scaled better on the ADI code than the PGHPF compiler. However, it still suffers from being subset HPF, and the implementation of `INDEPENDENT` do loops is still lacking.

4.3 DEC HPF & Fortran 90 Timings on the DEC Alphafarm

DEC was the first vendor to offer a syntactically complete HPF compiler, but on the system benchmarked it had the most disappointing performance. Perhaps the most telling statistics is that of the sixteen codes, nine had their single processor Fortran 90 timings comparable or better than the eight processor HPF runs. Of the other seven codes, three had a speed-up figure above 2.0 with the rest delivering performances comparable to that from a single processor. In mitigation with two of the codes, it should be mentioned that the `INDEPENDENT` directive does not function with the compiler release which was used.

4.4 PGI PGHPF Timings on the Cray T3D

The Cray T3D is generally acknowledged as having a superior communications network to that of the IBM SP-2, in particular with a better latency. However, this

HPFA Code	PGHPF/SP-2	XLHPF/SP-2	DEC HPF/Alpha	PGHPF/T3D
ADI	32.1	71.6	68.1	77.6
2D FFT	2.8	3.6	2.5	8.9
NAS EP	25.8	71.9	73.2	46.6
2D Convolution	8.6	11.0	7.2	25.7
Box-Muller	9.5	18.2	33.5	18.2
2D-Percolation	13.6	52.0	61.8	29.8
Potts Model	24.1	–	113.0	72.9
Cahn-Hilliard	39.5	173.5	211.4	106.7
Gaussian Factor	9.1	33.9	29.1	27.7
Direct N-Body	57.0	205.9	207.5	202.9
Bubble Sort	36.2	113.4	108.3	99.5
Wavelet	8.5	33.7	36.6	23.4
Options Pricing	36.0	–	122.4	79.3
Cholesky Factor	52.6	131.9	154.9	92.6
Hough Transform	4.5	–	85.2	6.9
Hopfield Network	4.4	7.9	105.0	11.4

Table 3. Wall-clock execution times in seconds for the PGI PGHPF compiler on a IBM SP-2, the IBM XLHPF/XLF90 compiler on a IBM SP-2, the DEC HPF compiler on a DEC Alphafarm, and the PGI PGHPF compiler on a Cray T3D, for a single processor HPF run.

does not appear to be reflected in its performance with respect to the SP-2 port: five codes obtained a speed-up of 7.0 or better at eight processors and these codes are essentially embarrassingly parallel, with little interprocessor communications —although the aforementioned case with the ADI code in §4.1 again shows disappointing speed-up. The codes where the IBM SP-2 port bettered the Cray T3D version are actually those with substantial near-neighbour communications, namely from `CSHIFT()` operations. On the otherhand, this port contains the same features as the SP-2 version, in particular offering the users the option of expressing their code's parallelism with the `INDEPENDENT` do-loop directive.

5 Discussion

This exercise has demonstrated that today one can write HPF codes which scales well and have acceptable Fortran 90 to HPF latencies. In this context, it would be difficult to envisage a message-passing program outperforming such codes. However, currently it would appear that such codes should preferably either have few interprocessor communications, or have them as simple operations such as `CSHIFT()` and `SPREAD()` which the compiler can easily optimize.

Of the HPFA codes examined which did not scale well, these were generally due either to obvious gaps in the implementations (*e.g.,* full `INDEPENDENT` do loops in the DEC and IBM compilers), or to comparatively complicated communication patterns (*e.g.,* `MATMUL()` on non-local data, masked `CSHIFT()` operations). That profilers are now available to determine these problematic parts of the code was of major assistance in this report. However, more information could still be given to the user to optimise their HPF codes: such as for when arrays have been remapped, or if temporary arrays have been created, or if computations have been unnecessarily duplicated.

In conclusion, we have demonstrated that present day compilers of the HPF language are capable of good scalability and low latencies. Nonetheless it is very easy to construct codes which do not scale well, and for these cases the user must be provided with the information needed to identify and perhaps bypass these bottlenecks. This is more pertinent with HPF programming than with message-passing, where the ease of coding and the freedom to re-express a given computation is much greater.

6 Acknowledgements

The authors of this paper wishe to thank the following people for their contribution to the NPAC HPFA project: G Cheng, P D Coddington, D Leskiwd, M McMahon, S Ranka and G Robinson.

References

[BBL*Eds*93] D Bailey, J Barton, T Lasinski, and H Simon (*Editors*). The NAS Parallel benchmarks. *NASA Ames, NASA Technical Memorandum 103863*, July 1993.

[NPA96] Northeast Parallel Architectures Center HPF Application Kernels. `http://www.npac.syr.edu/hpfa/`, November 1996.

[Ric95] H Richardson. EPCC Technology Watch Report: High Performance Fortran. *Edinburgh Parallel Computing Centre* `http://www.epcc.ed.ac.uk/epcc-tec/documents/`, v1.2 September 1995.

[RYHF96] S Ranka, H W Yau, K A Hawick, and G C Fox. High Performance Fortran for SPMD Programming: An Applications Overview. *NPAC SCCS Report,* `http://www.npac.syr.edu/hpfa/Papers/HPFforSPMD/`, November 1996.

[THPFF94] The High Performance Fortran Forum. High Performance Fortran Language Specification, v 1.1. *Rice University, Houston Texas* `http://www.crpc.rice.edu/HPFF/home.html`, November 1994.

[TMPIF94] The Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. *University of Tennessee, Knoxville*, May 1994.