# A Comparison of HPF-like Systems: Early Prototypes*

V. Getov[1] T. Brandes[2] B. Chapman[3] T. Hey[1] D. Pritchard[1]

[1]Department of Electronics and Computer Science
University of Southampton, Southampton SO17 1BJ, U.K.

[2]German National Research Center for Computer Science
P.O. Box 1316, 53731 St. Augustin, Germany

[3]Institute for Statistics and Computer Science
University of Vienna, Bruenner Str. 72, Vienna A-1210, Austria

HPCC94-009

## Abstract

This paper describes our experience with some of the prototype HPF parallelisation tools, namely Adaptor, Fortran 90D and xHPF90, as they were available to us at the end of 1993. The comparison and evaluation of these early HPF compilers was equally difficult and important in order to gain some feedback to the compiler designers but also to accumulate some initial skills in writing efficient HPF programs. The available HPF-like systems were applied to the existing GENESIS benchmarks that have already been rewritten in Fortran 90 with HPF data distribution directives. At the beginning of the paper, an overview of the three systems is given with emphasis on their specific features and limitations, together with comparative analysis based on carefully designed methodology. The comparison includes a detailed study of the extent of implementation for both Fortran 90 and subset HPF attributes into the prototype systems, as well as the compilation characteristics of those systems and the range of new language features covered by the test programs. In order to ensure a basis for comparative performance evaluation, all benchmark measurements were taken on iPSC/860 as it was the only common platform for the three prototype systems. The subset HPF compiler tools under consideration have been having different levels of development during the evaluation exercise, and therefore the extent of benchmark measurements is different for different systems. The evaluation results include discussion and comments on compiling and runing the benchmark codes with focus on subset HPF features and some optimisation strategies for compilation of data-parallel programs in terms of communication, temporary storage requirements, and processor utilisation. The benchmark measurements are finally presented along with a comparison to the corresponding hand-written message-passing versions of benchmarks and interpretation of performance results.

*This work was supported in part by ESPRIT (CEC) through project P6643 (PPPE), the Engineering and Physical Sciences Research Council (UK) under contract # GR/J50309 and the Austrian Ministry for Science and Research (BMWF).

# 1 Introduction

The specification of High Performance Fortran (HPF) language [14] represents an extension to Fortran 90 [17] which addresses the problem of automatic parallelisation of data parallel programs within the Single Program Multiple Data (SPMD) model of parallel computations. The main extension streams of the language are data distribution features (new directives ALIGN, DISTRIBUTE, TEMPLATE, PROCESSORS, etc.), concurrent execution features (FORALL construct and several new intrinsics) and EXTRINSIC procedures. In order to allow early compiler availability, subset HPF has been defined within the HPF language specification document. It is intended that subset HPF will be capable of being implemented more rapidly by vendors, which will ensure that subset HPF codes will be able to run at an early date on a variety of different machines. In general, subset HPF includes data distribution features, Fortran 90 array language, dynamic storage allocation, FORALL statement and long names. It does not include dynamic data distribution, FORALL construct, EXTRINSIC procedures, generic functions and the Fortran 90 free source form.

The central idea of the automatic parallelisation is to distribute the large data structures like arrays among the available processors. This should be performed in such a way that most operations could be done locally without need of communication. The corresponding message passing statements are inserted automatically where global operations are necessary. Therefore the data parallel program with global data references is translated together with a user specified or implicitly defined data distribution, into a program with local and non-local references, where the latter are satisfied by automatically inserted message-passing statements. There were already some HPF-like systems available at the end of 1993 that make an automatic partitioning of data parallel Fortran 90 programs. These include:

- ADAPTOR is a tool developed at the GMD, Germany;
- The Fortran 90D compiler has been developed at the University of Syracuse;
- FORGE-90/xHPF90 is an automatic paralleliser for HPF developed by Applied Parallel Research, Inc.

These tools were applied to existing applications (GENESIS Benchmark Suite) that have already been rewritten in Fortran 90 with HPF data distribution directives. The goal of the task is to accumulate some early experience, which would allow us to find out how efficient such automatic partitioning tools are, for which kind of applications they are useful and for which applications they fail.

This paper gives a detailed description of the experiences, possible improvements for the tools, hints how to write efficient applications that can be translated with these tools, and a qualification whose applications are not suitable for an efficient translation. The paper also presents a comparison of capabilities for the prototype systems listed above and recommendations for the HPF compiler designers.

# 2 Description of prototype HPF-like systems

## 2.1 Adaptor

Adaptor (Automatic Data Parallelism Translator) is an HPF system developed at the GMD for translating data parallel Fortran programs into equivalent Fortran 77 message passing programs [6, 7]. Most features of Connection Machine Fortran (CMF) and many features of HPF are

supported. The parallel program can be written in such a way that it could be developed on a serial machine and is also suitable for vector machines or parallel machines with shared memory. Many features supported by Adaptor result also in good execution times for these architectures. In this way, it helps to design programs that run efficiently on nearly all architectures.

Adaptor only takes advantage of the parallelism in the array operations and of the parallel loops. It has no features for automatic parallelisation.

The source language of Adaptor can be defined briefly in the following way:

- Fortran 77
- the array extensions of Fortran 90 (inclusive dynamic arrays and intrinsic functions for arrays),
- layout directives to specify the data distribution,
- features of Connection Machine Fortran and High Performance Fortran (`FORALL` statement, new intrinsics, timing, random numbers).

The user can define host arrays, replicated arrays and distributed arrays. Similar directives as in CMF or HPF are also used for the specification of data layouts in Adaptor. As well as that, the parallel `FORALL` statement supported by Adaptor has the same syntax and semantics as proposed in these data parallel languages.

Many features of Fortran 90 and HPF cannot be used with Adaptor. The most serious restrictions are that Adaptor supports no modules, no pointers, no array-valued functions and no assumed-shaped arrays. Adaptor only supports block distributions along one dimension. More distributions will be supported in future releases.

Explicit alignment can be used in CMF and HPF to reduce communication. For Adaptor this feature has not been supported until now, but of course there is an implicit alignment of arrays that are declared and distributed in the same way.

Although the user will need to understand some issues of parallelism and has to know for efficiency reasons where message passing will be generated, the effectiveness of Adaptor is based on the fact that the user does not have to know any message passing commands, neither does he have to manage the control of the data partitioning. The programmer can change types of variables (e.g. single to double precision) and data distributions without rewriting any other statements in the program. There is no need to write two versions of code (host and node program) and many global array operations are translated to the most efficient code for the underlying architecture. The user can select between the following three programming models:

- If the HOST-NODE programming model is selected, Adaptor will generate a host program and a node program. The node program runs on all available nodes of the parallel machine, while the host program contains all I/O operations that will be executed on the front end system.
- In the HOSTLESS programming model, only one program will be generated, running on all available nodes. There is no host program. The first node takes care of all I/O operations.
- A program that runs only on a single node is generated when using the UNIPROC programming model. The program has no communication and therefore it ought to be faster than the previous one running on a single node. By choosing this model, programs with array operations, not available in Fortran 77 can be translated into sequential Fortran 77 programs.

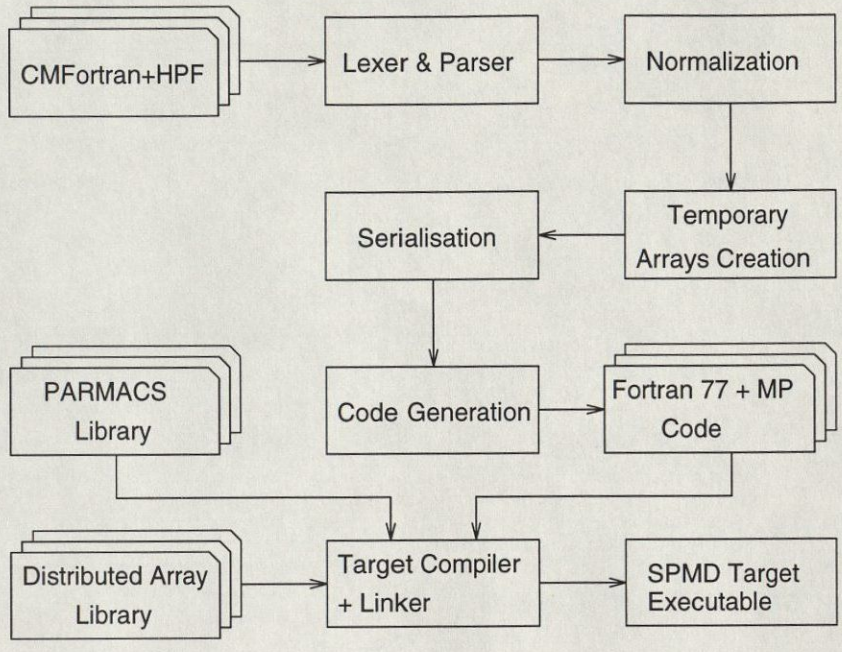



Figure 1: Overview of Adaptor

The following steps are done during the source to source transformation of Adaptor (see Figure 1):

1. The source program is parsed and an abstract syntax tree is generated. Symbol tables are created and used for a semantic analysis.
2. The program (abstract syntax tree) is normalized as far as possible to reduce the complexity of the translation system.
3. The program statements are splitted up into local and non-local operations. Temporary variables are created where necessary.
4. The array operations are translated into `FORALL` statements in the serialisation phase and the parallel loops are analysed.
5. The parallel loops are restricted to the local part owned by every processor in the final translation phase and communication statements for exchanging non-local data are generated. The new internal abstract syntax tree is unparsed back to source text.

Adaptor generates also a `Makefile`. Compiling and linking the generated message passing programs is performed by the Fortran compiler available on the target parallel machine. In order to perform the communication needed for global operations on distributed arrays, many library functions that build the distributed array library (DALIB) are implemented. This library contains:

- low level communication (send, receive, wait, etc.);
- high level communication (broadcast, reduction, barrier, etc.);
- data movements based on regular and irregular communication patterns;
- timing functions and tracing facilities;

- a parallel random number generator.

DALIB is implemented in C and can be considered as the runtime system of the whole compilation system. Most of the routines in this library are portable between different parallel platforms. Only the low level message passing commands, the timing functions and the random number generator have to be adapted to the target hardware architecture.

The following parallel machines are currently supported: CM-5, iPSC/860, Meiko CS1 and CS2, Parsytec GCel, KSR 1, SGI Multiprocessor Systems, and Alliant FX/2800. One version of DALIB is also implemented upon the public domain software PVM [9]. It guarantees the portability of the generated parallel programs to all machines where PVM is available.

The functionality and stability of Adaptor has improved dramatically if compared with version 0.1 from October 1992. Now the tool can also be used to translate a data parallel program to a serial Fortran 77 program. The source files of the current version 1.1 (September 1993) of Adaptor, the documentation files in PostScript and a number of example programs are available via 'anonymous ftp'. The address of the ftp-server is `ftp.gmd.de` (129.26.8.90), the files are in the directory `gmd/adaptor`.

Adaptor is a prototype compilation system for High Performance Fortran. Although the current version of Adaptor has many restrictions, the system gave useful insights in how to design efficient HPF compilers and to develop and integrate useful optimization strategies. Many users could take advantage of the Adaptor tool to develop future HPF applications.

## 2.2 Fortran 90D compiler

Fortran 90D is a Fortran 90 version of the Fortran D language [8]. Since Fortran D essentially adds to Fortran 90 extensions for data partitioning very similar to HPF, the authors from Syracuse University call their compilation system the Fortran 90D/HPF compiler. This compiler exploits only the parallelism in the data parallel constructs. It does not attempt to parallelise other constructs, such as do-loops and while-loops, since they are used only as naturally sequential control constructs in this language. The foundation of the design lies in recognizing commonly occurring computation and communication patterns. These patterns are then replaced by calls to optimized run-time support system routines. The run-time support system includes parallel intrinsic functions, data distribution functions, communication primitives and several others miscellaneous routines.

Figure 2 shows the basic components of the Fortran 90D compiler. The first step of the compilation is to check the syntax correctness of the source code. Given syntactically correct Fortran 90D/HPF program, the second step of the compilation is to generate a parse tree. The front-end module for the compiler, which incorporates both lexer and parser for Fortran 90 programs has been obtained from ParaSoft Corporation. In this module, the compiler also transforms each array assignment statement and where statement into equivalent FORALL statement with no loss of information. In this way the subsequent steps need only deal with FORALL statements.

The partitioning module processes the subset HPF data distribution directives; namely, decomposition distribute and align. Using these directives, it partitions data and computation (only FORALL statements) among processors. After partitioning, the parallel constructs in the node program are sequentialised since it will be executed on a single processor. The Fortran 90D compiler has to sequentialise Fortran 90 parallel constructs into sequential loops as there are few Fortran 90 compilers available for the nodes of MIMD distributed memory computers. This is performed by the sequentialisation module. Array operations and FORALL statements in the original program are transferred into loops or nested loops.

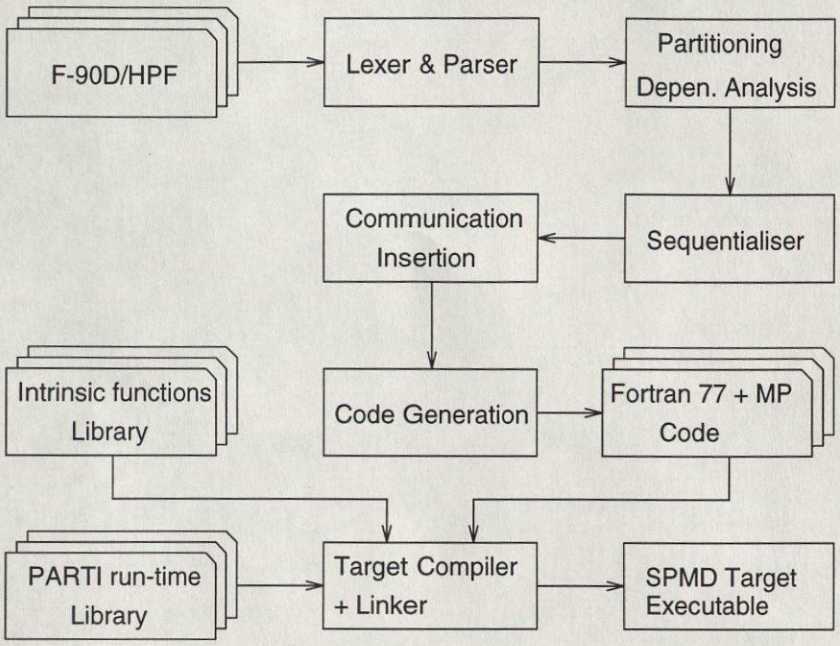



Figure 2: Steps in creating a SPMD program using the Fortran 90D/HPF compiler

The communication module detects communication requirements and inserts appropriate communication primitives. The Fortran 90D compiler produces calls to collective communication routines instead of generating individual processor send and receive calls inside the compiled code. There are three main reasons for using collective communication to support interprocessor communication in the compiler:

- Improved performance of HPF programs - this can be achieved by developing a separate library of interprocessor routines where each routine can be optimized;
- Increased portability of the Fortran 90D compiler - by separating the communication library from the basic compiler design only the machine specific low-level communication calls need to be changed when porting the compiler to a new platform;
- Improved performance estimation of communication costs - the costs of collective communication routines can be determined more precisely, thereby enabling the compiler to generate better distributions.

Finally, the code generator produces loosely synchronous SPMD code. The generated code is structured as alternating phase of local computation and global communication. Local computations consist of operations carried out by each processor on the data in its own memory. Global communication includes any transfer of data among processors, possibly with arithmetic or logical computation on the data as it is transferred (e.g. reduction functions). In these circumstances, processes do not need to synchronize during local computation. But, if two or more nodes interact, they will be implicitly synchronized by global communication.

The run-time support system of the Fortran 90D compiler consists of functions which can be called from the node programs of a distributed memory machine. Intrinsic functions support many of the basic data parallel operations in Fortran 90. Not only do they provide a concise means of

expressing operations on arrays, but they also identify parallel computation patterns that may be difficult to detect automatically. The intrinsic functions that may induce communications can be divided into five categories as follows:

- structured communications - CSHIFT, EOSHIFT
- reduction - DOTPRODUCT, ALL, ANY, COUNT, MAXVAL, MINVAL, PRODUCT, SUM, MAXLOC, MINLOC
- multicasting - SPREAD
- unstructured communications - PACK, UNPACK, RESHAPE, TRANSPOSE
- special routines - MATMUL

The first category requires data to be transferred using less overhead structured shift communication operations. The second category of intrinsic functions requires computations based on local data, followed by the use of a reduction tree on the processors involved in the execution of the intrinsic function. The third category uses multiple broadcast trees to spread data. The fourth category is implemented using unstructured communication patterns. The fifth category is implemented using existing research on parallel matrix algorithms. The current implementation of Fortran 90D ($\beta$ release, August 1993) has more than 500 parallel run-time support routines divided into two libraries - INTRIN (Intrinsic) and PARTI (Parallel Automatic Runtime Toolkit from ICASE) built on the top of Express message passing primitives [18]. Hence, in order to switch to any other message passing system, it will only be necessary to replace the calls to the communication primitives in the two libraries.

## 2.3 xHPF translator

The FORGE 90 package consists of several modules that are currently available for user-guided or automatic parallelisation of Fortran programs. The baseline system is an analysis tool for Fortran 77 programs and is required in order to use any other modules. The remaining parts include a distributed memory parallelisation tool (DMP), and a subset HPF translator (xHPF).

The xHPF translator is the most recent part of the FORGE 90 parallel programming environment [4]. It performs automatic parallelisation of subset HPF programs. The xHPF system is identical to the xHPF77 system, with the addition that the Fortran 90 syntax is initially converted into Fortran 77. The xHPF system is limited to the Fortran 90 features identified by the HPFF.

Fortran 90 programs must first be converted into Fortran 77 and then the Fortran 77 code examined. This is performed by KAPR, a special modification of the KAP pre-processor from Kuck and Associates Inc. (KAI), which acts as a front-end for xHPF (see Figure 3).

The xHPF77 is a new module offered by APR which is available as an addition to the DMP. The xHPF77 system accepts Fortran 77 with HPF data distribution directives and translates this to Fortran 77 with message passing. The xHPF77 module does not implement any additional features for parallelising code over the DMP. Accordingly data distribution in the xHPF77 system is restricted to one dimension.

The HPF directives specify the data partitions only, while the DMP requires both data distributions and loops for parallelisation to be specified. The approach xHPF77 follows is that all loops that contain the distributed data are parallelised. As this translator is built upon the DMP, all the limitations of the DMP still apply. Hence no code restructuring takes place resulting in possible inefficiencies in the parallelised code.

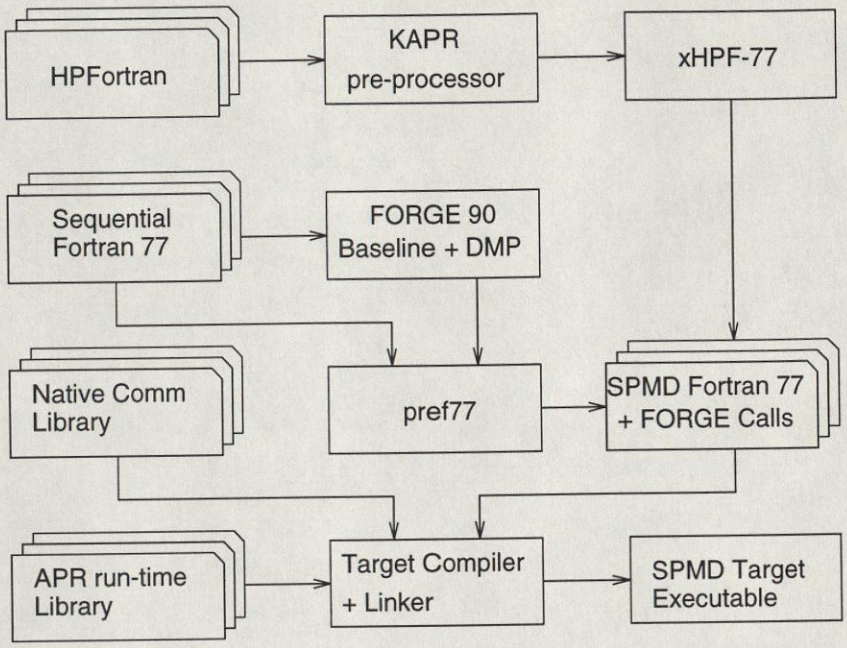



Figure 3: Steps in creating a SPMD program using the FORGE 90 system

The data partitioning commands and loop distribution commands are stored separately from the sequential code. There is a preprocessor called 'pref77' which is used to create the SPMD program from the sequential code and the commands stored by the DMP. This has the advantage that it is possible to maintain the sequential code and execute the same parallelisation commands to result in a new parallel program.

The parallel program produced by 'pref77' contains calls to FORGE's native communications library. This code must then be compiled by the target system compiler and linked to a specific communication syntax. Currently FORGE have communication libraries for native INTEL, native nCUBE, PVM, EXPRESS and P4. Calls to the subset HPF intrinsic functions and subroutines that are not already part of Fortran 77 are linked to routines provided by the APR's run-time library.

# 3 Comparison methodology

## 3.1 Compiler features of prototype systems

### 3.1.1 Supported features of Fortran 90

Table 1 gives a general overview of the implementation status of the Fortran 90 features as defined in the subset HPF definition [14] within the compiler tools under evaluation. The information is based on the existing documentation, papers, etc. and also on the experiments with the GENESIS benchmarks and some other small examples where necessary.

The information in Table 1 shows that all Fortran 90 features in subset HPF have been implemented in the xHPF90 compiler. Array-valued functions, allocatable arrays, interface blocks and optional

Table 1: Implementation of Fortran 90 features in subset HPF

| Features | Adaptor | Fortran 90D | xHPF90 |
|---|---|---|---|
| MIL-STD-1753 features | yes | yes[a] | yes |
| Array syntax | yes | yes[b] | yes |
| WHERE construct | yes | yes | yes |
| Array valued functions | no | no | yes |
| Automatic arrays | yes | no | yes |
| Allocatable arrays | no | no | yes |
| Intrinsic functions | not all | not all | yes |
| New type declarations | yes | yes | yes |
| Interface blocks | ignored[c] | no | yes |
| Optional arguments | no | no | yes |
| Syntax improvements | yes | yes[d] | yes |

[a]There are problems with the implementation of the IMPLICIT NONE statement (see section 4.2).
[b]Currently only the WHERE statement is implemented
[c]Feature is parsed and recognized but not really supported.
[d]The '!' form of comments is not implemented for the HPF directives.

Table 2: Implementation of subset HPF features

| Features | Adaptor | Fortran 90D | xHPF90 |
|---|---|---|---|
| Processors directive | ignored[a] | yes | ignored[a] |
| Template directive | yes | yes | yes |
| Distribution dimensions | 1 dim | 2 dim | 1 dim |
| Decomposition attributes | BLOCK | BLOCK, CYCLIC | BLOCK,CYCLIC[b] |
| Simple alignment | yes | yes | yes |
| Full alignment $(a * I + b)$ | no | no | no |
| FORALL statement | canonical[c] | yes | yes |

[a]Feature is parsed and recognized but not really supported.
[b]xHPF90 provides also several other attributes to specify the memory allocation for distributed arrays.
[c]The indices must be simple functions of the loop variables of the FORALL statement.

arguments are not implemented in Adaptor and Fortran 90D yet. Both compiler tools support most of Fortran 90 array intrinsic functions (see sec. 2.1 and 2.2) with some exceptions regarding data types, construction functions, number of dimensions, etc.

Some Fortran 90 features (free source form, modules, pointers, derived types, recursion, new control constructs, etc.) are not in the subset. In most cases the HPF systems under consideration do not support these features as they have committed themselves to the subset HPF language.

### 3.1.2 Supported HPF features

Table 2 shows which subset HPF directives are recognized and processed by the compiler tools under evaluation.

The `PROCESSORS` directive is vital for Fortran 90D, whereas it is accepted but not implemented by Adaptor and xHPF90. The latter case is the basis for generating an output code which is independent of the number of processors (see Table 4). One of the most important parameters of HPF-like compiler tools is the number of distribution dimensions. These are one of the strongest

restrictions being only one or two at the time of evaluation exercise. The HPF `DISTRIBUTE` directive provides `BLOCK` and `CYCLIC` decomposition attributes in most of the compiler tools (Adaptor is the only exception). xHPF provides some extra explicit (non-standard) decomposition attributes to widen the variety of possible mappings of data objects to abstract processors.

We consider the case when all data objects are aligned to the template using simple constant expressions, as a simple alignment. This alignment simplifies data decomposition and does not imply the need of generating complex alignment trees, as the data objects are directly aligned to the template. In this case at least one template directive must exist in the program. The full (in terms of subset HPF) alignment lifts these restrictions but does not accept alignment subscripts more complicated than first order with constant offset $(a * I + b)$.

Although very rarely, the HPF-like systems under consideration support some of the features of full HPF. Table 3 contains the information on this matter.

Table 3: Implementation of full HPF features not in subset HPF

| Features | Adaptor | Fortran 90D | xHPF90 |
|---|---|---|---|
| Redistributions | no | no | no |
| Inherited dist. | no | no | yes |
| FORALL construct | no | no | yes |
| HPF library | only scatter | no | yes |
| Extrinsic Procedures | possible | no | yes |
| PURE attribute | yes | no | no |

### 3.1.3 Compilation characteristics

It is important for the user to know how to deal with a translation tool. Table 4 gives a summary of the compilation characteristics for the four HPF-like systems under consideration. It can be possible that the tool has its own driver program that translates, compiles and links the program with the run-time system. One would also expect the tool to generate a `Makefile`. Some tools can be used in a batch manner and/or interactively.

When considering larger programs the HPF-like systems should be able to perform separate compilation. This means that the whole program can be split up in several files which can be compiled independently.

The executable of the generated SPMD program will be loaded on the nodes of the parallel machine. Sometimes, the code runs only on the number of processors that has been specified for the translation with the tool (static case). It would be more convenient for the users if the executable is not dependent on the number of processors the parallel program will be running on(dynamic case).

In the context of message passing programming overlap areas are used for distributed arrays. The overlap area contains data that is not owned but only used by the processor. Data in overlap areas is replicated data and must therefore be the same among all the processors. If a compiler for HPF does not insert overlap areas in the generated message passing program, many additional temporary arrays are usually required. This will not increase the communication traffic, but more memory space and more local memory transfer is necessary. Sometimes it might be possible that the user specifies the overlap area by its own (restricted case), but the automatic insertion of overlap areas is obviously more convenient.

HPF is a language where actual and dummy arguments might have different distributions. Therefore implicit redistributions at the entry and after the end of the subroutine might be required.

Table 4: Some important compilation characteristics

| Features | Adaptor | Fortran 90D | xHPF90 |
|---|---|---|---|
| Compiler driver | batch | batch | batch |
| Makefile | yes | no | no |
| Separate compilation | no | no | yes |
| Number of processors | dynamic | static | dynamic |
| Overlap areas | restricted | no | no |
| Redistribution at subroutine boundaries | no | no | no |

Subset HPF compilers have to support this feature, otherwise the user must take care of the redistributions.

## 3.2 Description of test programs

Most of the GENESIS benchmarks were initially developed within ESPRIT Project P2702 – GENESIS [13] by a number of project partners including Liverpool and Southampton Universities, PALLAS GmbH, the European Centre for Medium Range Weather Forecasting and the Universitat Politechnica de Catalunya. The current release (2.2) of the GENESIS benchmark suite is based on subset HPF, whilst future releases will be able to take advantage of dynamic data distribution, data parallel programming (FORALL construct) and code tuning by EXTRINSIC procedures. A subset HPF conversion strategy, which assumes that the conversion process is divided into two phases (see Fig. 4), has been adopted. During the first phase the work is concentrated on the development of Fortran 90 version of the benchmarks. A main task in this phase is to introduce array constructs into the codes. The second phase addresses the data distribution, resulting in the subset HPF version of benchmarks [11].

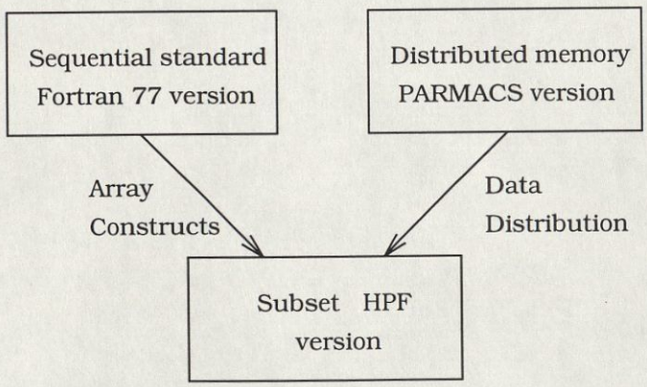



Figure 4: Subset HPF conversion strategy

Release 2.2 of the GENESIS benchmarks comprises sixteen codes that vary from synthetic code fragments measuring basic machine parameters, through important application kernels, to compact research applications [1]. This hierarchical structure allows information derived from the simpler codes to be used in explaining the performance characteristics of the more complicated codes. Thus the benchmark suite can be used to evaluate performance on a range of levels from simple machine parameters to full applications where effects due to non-parallelisable sections

of code, and memory, communication or I/O bottlenecks may become important. The following codes have been used in the evaluation of HPF-like systems:

**TICK1** Measures resolution of system clock.

Versions F-77, Subset HPF

**TICK2** Measures accuracy of system wall-clock time.

Versions F-77, Subset HPF

**TRANS1** This benchmark distributes a matrix into square submatrices for parallel transposition using naive algorithm. During the execution pairs of processors communicate with each other simultaneously exchanging very long messages (depending on the problem size and the number of processors).

Versions F-77, PARMACS, Subset HPF

**FFT1** One-dimensional FFT. Popular test with relatively high level of parallelism, but low level of vectorisation and a hypercube type communication.

Versions F-77, PARMACS, Subset HPF

**PDE1** Three-dimensional Poisson solver using red-black relaxation. Only nearest neighbour interactions are required and the number of floating point operations per grid point is very small when compared to other more complex PDEs.

Versions F-77, PARMACS, Subset HPF

**PDE2** Two-dimensional multi-grid Poisson solver. This kernel requires highly structured long distance communication. During the solution process messages and vectors ranging from a single word upwards are used. Efficient execution of this problem therefore requires good short vector performance for each processing node together with a reasonably low start-up time for message passing.

Versions F-77, PARMACS, Subset HPF

**MD1** Molecular dynamics code. This benchmark uses techniques from molecular dynamics to solve the Newtonian equations of motion for large number of interacting particles. The solution is used to calculate thermodynamic properties of the system. The benchmark uses a Lennard-Jones potential with a linked-list algorithm for solution of the equations of motion.

Versions F-77, PARMACS, Subset HPF

The utilized Fortran 90 subset features of test programs are summarised in Table 5 and the utilized subset HPF features are given in Table 6. The current set of GENESIS benchmarks includes regular problems in one, two, three and four dimensions. From this point of view all benchmarks conform to the SPMD programming model and allow HPF transformations. It is essential to know which subset HPF features are utilized in the GENESIS benchmark codes. The two tables below contain this information. 'Stack allocate' in Table 5 means that a DEALLOCATE statement deallocates both the specified array and any others allocated after it. There are no codes which use features of Fortran 90 or HPF, that are not in the subset. In the case of `MD1` one could take advantage of the HPF library by using `COPY_SCATTER`.

Some additional comments:

- `INHERIT` directive is used in FFT1 and PDE1, but the corresponding subroutines will always be executed with the same distribution and no redistribution is required.
- `MD1` has one- and three-dimensional distributions.

Table 5: Utilized Fortran 90 subset features

| Features | TRANS1 | FFT1 | PDE1 | PDE2 | MD1 |
|---|---|---|---|---|---|
| MIL-STD-1753 features | yes | yes | yes | yes | yes |
| array syntax | yes | yes | yes | yes | yes |
| intrinsic functions | TRANSPOSE | - | MAXVAL | MAXVAL | RESHAPE CSHIFT, SPREAD |
| automatic arrays | no | yes | no | no | yes(stack allocate) |
| full allocatable arrays | no | no | no | no | no |
| array valued functions | no | no | no | no | no |
| new type declarations | yes | yes | yes | yes | yes |
| optional arguments | no | no | no | no | no |
| interface blocks | no | no | no | no | no |
| syntax improvement | yes | yes | yes | no | no |

Table 6: Utilized subset HPF features

| Features | TRANS1 | FFT1 | PDE1 | PDE2 | MD1 |
|---|---|---|---|---|---|
| processors directive | yes | yes | yes | no | no |
| template directive | no | no | no | no | no |
| distribute (BLOCK) | no | yes | no | no | yes |
| distribute (BLOCK,BLOCK) | yes | no | no | yes | no |
| distribute (BLOCK,BLOCK,BLOCK) | no | no | yes | no | yes |
| simple alignment | yes | yes | yes | yes | yes |
| full alignment $(a * I + b)$ | no | no | no | no | no |
| FORALL statement | no | no | no | no | yes |

- `FFT1` and `MD1` use indirect addressing.

Concerning the importance of the different codes for the HPF compilers the following can be pointed out:

- TICK1 and TICK2 are only used for testing timer functions.
- TRANS1 and PDE1 contain very important features and should run efficiently. PDE1 would take advantage from overlap areas.
- PDE2 should run efficiently and proves how well the compiler can deal with structured communications and array syntax. It uses parameter functions.
- FFT1 tests indirect addressing and array movements, but the current version does not seem to be so efficient.
- MD1 will test many HPF features, but the HPF version is very slow, compared to the serial program.

# 4 Evaluation results

## 4.1 Introductory comments

The subset HPF compiler systems under consideration have been having different level of development during the evaluation exercise. Therefore the extent of evaluation results is different for different systems. Similarly, the different subset HPF systems have been having different level of availability. The HPF versions of the GENESIS benchmarks have also been a subject of development and alterations during the evaluation procedure. The changes to the benchmarks can be classified in three categories as follows:

- First set of changes. A few syntax errors have been discovered as the subset HPF GENESIS benchmarks were produced with the NAG's Fortran 90 compiler and had not been tested with a real HPF system before.
- Second set of changes. The input language of different HPF tools conforms only to a subset of subset HPF. The data partitioning has to be modified quite often so that the HPF product could parallelise the code. The main restriction here was the limited number of dimensions for data decomposition. This peculiarity of the current HPF-like compilers has been a source for a number of editions in order to achieve successful compilation. These editions and the corresponding limitations of every HPF tool under evaluation are described at the beginning of the following subsections.
- Many specific changes have been attempted in order to help the compilers to overcome particular problems. Most of them are discussed in detail in the specific evaluation subsections below.

The main emphasis of the benchmarking measurements has been focused on evaluating where possible the HPF-like systems under consideration on iPSC/860 machines. The iPSC/860 is the best parallel computer to perform this evaluation, as it is the only common platform for all the three HPF-like translators, which would allow comparative performance results. Another interesting and useful comparison is between the time measurements when compiling with the prototype HPF systems and the time measurements from the hand-written code with explicit message passing primitives (PARMACS macros [12] in our particular case).

The benchmark measurements were taken on four different iPSC/860 machines - the 64-node configuration in Daresbury (U.K.), the 32-node hypercube in Jülich (Germany), a 32-node machine available to APR, Inc. (U.S.A.) and a small development platform at K.U. Leuven (Belgium). Although the convenient and easy access has always been important, the main reason for using so many different platforms has been the availability of all necessary pieces of system software and programming environments. The evaluation details of the three hypercubes are given in Table 7.

## 4.2 TRANS1 benchmark

Figure 5 shows time measurements for the matrix transpose benchmark on the iPSC/860 and plots the elapsed time against number of processors for different HPF-like systems and the hand-written code for the PARMACS message passing environment. The execution time of the standard Fortran 77 single-processor version (0.214 sec.) is also shown for comparison. The problem size for the TRANS1 measurements was defined as MAXD = 480, which means a $480 \times 480$ square matrix. It was not possible, however, to run the Fortran 90D measurements with this problem size and the benchmarking results for this particular case were taken for a smaller problem size (MAXD = 120). We see at least two reasons for that:

Table 7: Evaluation details of the iPSC/860 parallel platforms

| Benchmarking Characteristics | ADAPTOR | Fortran 90D | xHPF | PARMACS |
|---|---|---|---|---|
| No. of nodes | 32 | 4 | 32 | 64 |
| Local memory | 16 Mbyte | 8 Mbyte | 16 Mbyte | 16 Mbyte |
| iPSC System s/w | Release 3.3.2 | Release 3.3.2 | Release 3.3.2 | Release 3.3.2 |
| Fortran compiler | if77 Rel4.0 | if77 Rel4.0 | if77 Rel4.0 | if77 Rel4.0 |
| Compiler switches | -Mvect -O2 | -Mvect -O4 | -Mvect -O4 | -Mvect -O4 |
| M/P environment | PARMACS 5.1 | Express 3.2.1 | NX-2 | PARMACS 5.1 |

- The compiler generates too many temporary arrays and other extra code;
- The local memories on the hypercube at Leuven are smaller then the local memories on the other platforms.

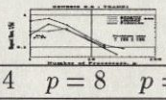
The hand-written PARMACS version of TRANS1 can only be run on square configurations of processors, as shown in the table. It has also been noted that Adaptor performs only one-dimensional distribution of the arrays. The communication patterns are therefore different from the patterns of the Fortran 90D and PARMACS versions of this benchmark.

### 4.3 FFT1 benchmark

Figure 6 shows time measurements for the 1-dimensional FFT benchmark on the iPSC/860 and plots the elapsed time against number of processors for the automatically produced code by Adaptor and the hand-written code for the PARMACS message passing environment. The execution time of the standard Fortran 77 single-processor version (1.403 sec.) is also shown for comparison. The problem size for the FFT1 measurements was defined as LOGN = 16, which means a transformation of 65536 complex data points.

The results have shown that the current ADAPTOR implementation of the FFT1 butterfly is very inefficient for parallel execution. Although this code scales well, the HPF version is more than 100 times slower than the PARMACS version. In order to further investigate the reason for this enormous delay a trace file has been generated and the timing diagram for the execution of the butterfly phase produced using ParaGraph visualisation tool (see Figure 7). The diagram shows that most of the time overhead has been spent in calculation operations. It is believed that this significant delay is not a result of the temporary arrays generated by Adaptor, but only a further investigation can give a detailed description of this problem.

Usually the communication structure of the butterfly phase of 1-dimensional hand-written FFTs employs only one type of bidirectional communication between pairs of nodes. The two messages associated with every pair of nodes are independent and therefore they can be exchanged simultaneously in order to decrease the communication time [10]. The diagram, however, shows that the communication exchange has been sequentialized by Adaptor. It is also clear from the diagram that Adaptor generates twice as many messages in comparison to the hand-written message passing version. This is because the tool can not recognize the possibility of packing two vectors (one for the real and one for the imaginary parts of the transform) in one buffer and to decrease twice the overall message start-up time.



| HPF system / MP env. | $p = 1$ | $p = 2$ | $p = 4$ | $p = 8$ | $p = 9$ | $p = 16$ | $p = 25$ | $p = 32$ |
|---|---|---|---|---|---|---|---|---|
| ADAPTOR | 0.65 | 0.78 | 0.49 | 0.23 | – | 0.12 | – | 0.09 |
| Fortran 90D(MAXD=120) | 0.31 | 0.32 | 0.38 | n/a | – | n/a | – | n/a |
| xHPF90 | 0.245 | 0.523 | 0.344 | 0.194 | – | 0.107 | – | n/a |
| PARMACS | – | – | 0.356 | – | 0.164 | 0.093 | 0.060 | – |

Figure 5: Elapsed time in seconds for the execution of TRANS1 on iPSC/860, compiled with Adaptor, Fortran 90D, xHPF90 and PARMACS. The problem size is defined by MAXD = 480.

## 4.4 PDE1 benchmark

Figure 8 shows time measurements for the 3-dimensional 'red-black' relaxation benchmark on the iPSC/860 and plots the elapsed time against number of processors for the automatically produced code by Adaptor and the hand-written code for the PARMACS message passing environment. The execution time of the standard Fortran 77 single-processor version (0.250 sec.) is also shown for comparison. The problem size for the PDE1 measurements was defined as N = 6, which means $64 \times 64 \times 64$ data points.

The use of an overlap area in the `PDE1` code does not save any communication, but it saves memory space and data movements to local temporary arrays. In order to investigate the efficiency of this solution two series of time measurements have been taken - with and without an overlap area. The results show that the version with an overlap area is significantly faster for smaller number of processors. For larger number of processors the effect of the overlap area is negligible.

| HPF system / MP env. | $p = 1$ | $p = 2$ | $p = 4$ | $p = 8$ | $p = 16$ | $p = 32$ |
|---|---|---|---|---|---|---|
| ADAPTOR | 215.0 | 110.9 | 59.3 | 33.3 | 20.6 | 14.0 |
| xHPF90 | 11.72 | 15.27 | 17.88 | 20.88 | 25.80 | 34.38 |
| PARMACS | – | 0.857 | 0.529 | 0.302 | 0.171 | 0.101 |

Figure 6: Elapsed time in seconds for the execution of FFT1 on iPSC/860, compiled with Adaptor, xHPF90 and PARMACS. The problem size is defined by LOGN = 16.

### 4.5 PDE2 and MD1 benchmarks

The efficiency of these codes is extremely poor. This is also due to the fact that the HPF programs themselves are very inefficient if compared with the sequential Fortran 77 program. Therefore no time measurements for these benchmarks are given here.

## 5 Conclusions

### 5.1 Recommendations

The HPF parallelisation tools are still in their infancy. They should be given time to develop in the areas of generality and performance before being used for commercial code parallelisation. Efficient compilation is highly non-trivial. There are often many candidate choices for the localisation of individual computations – calculation placement, which creates enough room for various optimisations. Two principal areas demanding investigation are

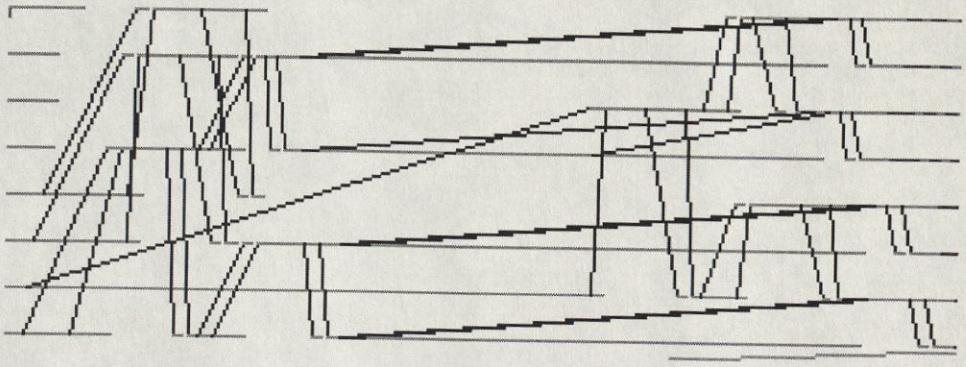


Figure 7: Timing diagram for the execution of the FFT1 butterfly phase on 8 processors, compiled with Adaptor

- Effective strategies for calculation placement;
- Efficient schemes for managing communication, and minimising associated data copying.

It is absolutely necessary to give the user feedback about the generated parallel program from his code. He should be informed which statements will require communication and whether this communication is efficient or not. As HPF systems will not implement all possible optimizations in an early stage, the documentation should demonstrate how to implement efficient programs.

## 5.2 Compiler vs. run-time system

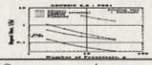
All of the early HPF systems have their own run-time system that performs operations on distributed arrays. The more complex the operations of the run-time system are, the less will be the potential of optimizations for future versions.

```
      integer IND(N)
      real A(N), B(N)
!HPF$ distributed (BLOCK) :: A, B, IND
      ...
      A =  B(IND)
      ...
```

If the above assignment with indirect addressing is translated in one call, it will not be possible to reuse the communication pattern. A better solution is to compute a schedule for the implied communication pattern and to use it for sending and receiving values. This will result in three calls, and the schedule can be reused. When more communications patterns are used, the run-time system should support the combination of more schedules to one schedule to minimize start-up times.

## 5.3 Portability issues

Portability of a program is a property that is most important for users. A first requirement is that the code runs on all machines of interest. But especially in the context of parallel machines it is also important that the application runs efficiently on all target machines. It may be necessary to tune the code where appropriate in order to achieve that.



| HPF system / MP env. | $p = 1$ | $p = 2$ | $p = 4$ | $p = 8$ | $p = 16$ | $p = 32$ |
|---|---|---|---|---|---|---|
| ADAPTOR | [a] | 1.11 | 0.61 | 0.35 | 0.22 | 0.15 |
| ADAPTOR(overlap) | [a] | 0.53 | 0.32 | 0.22 | 0.17 | 0.14 |
| xHPF90 | [a] | [a] | [a] | 3.5 | 2.3 | 1.7 |
| PARMACS | – | 0.153 | 0.087 | 0.056 | 0.041 | 0.032 |

[a] No results, because the problem size is too large to fit into the local memories.

Figure 8: Elapsed time in seconds for the execution of PDE1 on iPSC/860, compiled with Adaptor, xHPF90 and PARMACS. The problem size is defined by N = 6.

While the portability of sequential Fortran 77 programs is no longer a real problem on sequential machines and also on vector machines, the portability of High Performance Fortran programs will still be a problem for some years. There are many reasons for that, but the most important from our point of view are listed below:

- HPF is based on Fortran 90. Therefore recoding of existing Fortran 77 applications is required.
- First experiences have shown that Fortran 90 itself requires very good compilers as Fortran 90 programs are very often slower than their Fortran 77 counterpart when compiled with the existing early Fortran 90 compilers.
- HPF is a new language and the early HPF systems do not support all features of subset HPF and hence require some canonical representations. For example, there are several different sorts of syntax for the distribution directives which are not fully implemented in the early

HPF systems under evaluation.

- The task of an HPF compiler is also to translate implicit communication to explicit communication. Minimization of communication is a central issue for optimizations. However, different systems have different strategies for generating message passing directives that makes efficiency and portability very difficult issues.

### 5.4 The Fortran 90 problem

HPF is intended to use the inherent parallelism of array operations and of parallel loops. In most cases, this requires recoding of existing applications. Users will be reluctant to recoding in Fortran 90 even if Fortran 90 is as efficient as Fortran 77.

The following example of `MD1` shows the typical problem of a Fortran 90 compiler:

```
      NATMCN (:, :, :) =
1            CSHIFT (CSHIFT (CSHIFT (NATMCL (:, :, :),
2                    SX, DIM = 1), SY, DIM = 2), SZ, DIM = 3)
```

When using serial Fortran 77 this would be coded in such a way that one READ and one WRITE for every element of the arrays is required. But straightforward Fortran 90 compilers will implement this array operation by using temporary arrays:

```
TMP1    (:, :, :) = CSHIFT (NATMCL (:, :, :), SX, DIM = 1)
TMP1    (:, :, :) = CSHIFT (TMP1   (:, :, :), SY, DIM = 2)
NATMCN  (:, :, :) = CSHIFT (TMP1   (:, :, :), SZ, DIM = 3)
```

This needs at least three READ and three WRITE operations.

Another problem of the array syntax is that due to the blocking of operations the cache of a processor may not be utilized well. Therefore it is very important for an HPF system to generate efficient serial code and to avoid temporary arrays and local data movements. All optimization issues of Fortran 90 compilers are issues of HPF compilers.

## Acknowledgments

Amongst those who have helped by discussion and criticism with the preparation of this paper we would like to mention Dave Watson (NA Software) and John Merlin (University of Southampton) for their useful and detailed comments. John Levesque, the President of Applied Parallel Research. Inc. has provided us with the information about their xHPF90 compiler and also ran the GENESIS HPF benchmarks with it. The Fortran 90D compiler was obtained from the University of Syracuse with the help of Zeki Bozkus. We have also received very generous assistance from Dirk Roose (Leuven University) with the access to their iPSC/860 installation and the Express communication libraries.

## References

[1] C.A. Addison, V.S. Getov, A.J.G. Hey, R.W. Hockney, I.C. Wolton. The GENESIS Distributed-Memory Benchmarks. In: J. Dongarra and W. Gentzsch (Eds.), *Computer Benchmarks*, North-Holland 1993, pp. 257-271.

[2] I. Ahmad, R. Bordawekar, Z. Bozkus, A. Choudhary, G. Fox, K. Parasuram, R. Ponnusamy, S. Ranka, R. Thakur. *Implementation and Scalability of Fortran 90D Intrinsic Functions on Distributed Memory Machines*. Technical Report SCCS-256, NPAC, University of Syracuse, 1993.

[3] Applied Parallel Research Inc. *APR's FORGE 90 Parallelization Tools for High Performance Fortran (HPF)*. June 1993.

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

[5] Z. Bozkus, A. Choudhary, G. Fox, T. Haupt, S. Ranka, M.-Y. Wu. *Compiling Fortran 90D/HPF for Distributed Memory MIMD Computers*. Technical Report SCCS-444, NPAC, University of Syracuse, 1993.

[6] T. Brandes. Adaptor: A Compilation System for Data Parallel Fortran Programs. *Proc. of AP'93*. Saarbrücken, March 1993.

[7] T. Brandes. Compiling Data Parallel Programs to Message Passing Programs for Massively Parallel MIMD Systems. *Proc. of Working Conference on Massively Parallel Programming Models*. Berlin, September 1993.

[8] G. Fox, S. Hiranandani, K. Kennedy, C. Koelbel, U. Kremer, C. Tseng, M. Wu. *Fortran D Language Specification*. Technical Report COMP TR90-141, Rice University, April 1991.

[9] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam. *PVM 3 User's Guide and Reference Manual*. Technical Report ORNL/TM-12187, Oak Ridge National Laboratory, May 1993.

[10] V. Getov. 1-Dimensional Parallel FFT Benchmark on SUPRENUM. In: D. Etiemble and J.-C. Syre (Eds.) PARLE'92, Parallel Architectures and Languages Europe. *Lecture Notes in Computer Science*, **605**, 1992, pp 163-174.

[11] V. Getov, T. Hey, R. Hockney, I. Wolton. The GENESIS Benchmark Suite: Current State and Results. *Proceedings of 1st Workshop on Performance Evaluation of Parallel Systems - PEPS'93*. Coventry, November (1993) pp 182-190.

[12] R. Hempel. *The ANL/GMD Macros (PARMACS) in FORTRAN for Portable Parallel Programming using the Message Passing Programming Model*. User's Guide and Reference Manual, Version 5.1. GMD, November 1991.

[13] A. J. G. Hey. The GENESIS Distributed-Memory Benchmarks. *Parallel Computing*, **17**(10-11), 1991, pp 1275-1283.

[14] High Performance Fortran Forum. *High Performance Fortran Language Specification, Version 1.0*. Technical report CRPC-TR-92225, Rice University, Houston, May 1993.

[15] J. Levesque. *FORGE 90 and High Performance Fortran (HPF)*. Applied Parallel Research, Inc., 1993.

[16] R. Lovely, A. Marshall, D. Watson. *Design Specification of HPF Mapper*. PPPE Deliverable D4.1.a, February 1993.

[17] M. Metcalf and J. Reid. *Fortran 90 Explained*. Oxford Science Publications/OUP, Oxford and New York, 1990.

[18] ParaSoft Corporation. *Express Fortran Reference Guide*. Version 3.0, 1990.

[19] H. Zima, B. Chapman. Compiling for Distributed-Memory Systems. *Proc. of the IEEE*. Special Section on Languages and Compilers for Parallel Machines, February 1993.