Release Notes: Southampton *High Performance Fortran* Translation System, Beta release 0.1

D.B. Carpenter and J.H. Merlin

June 15, 1995

1 Overview of the release

The Southampton HPF translation system is realised as a translator from HPF [1] to standard Fortran. The generated Fortran contains calls to an HPF run-time system, which in turn makes calls to an underlying message-passing library.

Concurrency and data parallel execution are obtained from the use of array syntax, e.g. array assignments, array external and intrinsic functions and the `WHERE` statement and construct. `DO`-loops are not parallelised. The HPF `FORALL` statement is supported but is sequentialised, so the user is encouraged to use array assignments rather than `FORALL` for this version. We intend to provide data-parallel execution of `FORALL` in a later release of the system.

Programs compiled with the beta release are slow. Our development efforts so far have been directed towards initial implementation of Subset HPF, rather than optimisation. For example, some of the code generated by the translator is obviously inefficient and could be easily optimised, but we have not had time to look at these refinements yet. The run-time library supplied is implemented quite naively—common operations can probably be speeded by an order of magnitude.

These obvious optimisations will be incorporated in the next release of the system. The current release is made available simply as an HPF development and testing environment.

Please report bugs to `bug-hpf@par.soton.ac.uk`, or use the automatic bug reporting form at `http://hpcc.soton.ac.uk/RandD/hpf/bug_report.html`

2 Using the hpf script

HPF programs are compiled by invoking a script called `hpf`, which calls the translator, then the Fortran compiler. Finally it calls the linker, automatically linking in the necessary run-time libraries.

All of the source code for an HPF program (including any library procedures it uses) must be translated, even parts of the code containing no HPF directives. This is because the translation process modifies procedure names and their argument lists, even in the absence of HPF data mapping directives.

The result of compilation is an executable which is run in a manner appropriate to the underlying message-passing platform. An HPF program can be run on P processors if and only if the size (the product of extents) of every processor arrangement defined explicitly or implicitly in the program is either P or 1.

The syntax of the `hpf` command is

```
hpf [-help] [-dryrun] [-T|-c] [-o OUT_FILE]
    [-C=CHECK] [-stack STACK_SIZE] [-procRANK EXTENT ...]
    [-temp=TEMP_DIR] [-v] [-w]
    [-compOpts COMP_OPTS] [-linkOpts LINK_OPTS]
    FILE ... [-lLIB_NAME] ...
```

Source files

HPF source files have one of the suffixes `.hpf`, `.hpf90` or `.h90`. In the first case the source format is assumed to be fixed (the old Fortran 77 style); in the latter two cases the source format is "free" (the format new to Fortran 90).

`hpf` also accepts files with suffix `.f`. These are assumed to be Fortran files which have been generated previously by translation of HPF—for example by using `hpf -T`. They are passed straight to the compiler and linked with the appropriate run-time libraries.

`hpf` will accept files with suffix `.o` or `.a`. These are object files, passed straight to the linker. Again, they would normally have been created by an earlier invocation of `hpf`.

Options to hpf

-help

Display a list of options.

-dryrun

Show but do not execute commands constructed by the `hpf` script.

-T

Translate only. Output the intermediate Fortran. Note that unless the `-o` option is specified, existing `.f` files with the same base name as the HPF source will be overwritten.

-c

Translate and compile only (do not link). This produces `.o` files.

-l *LIB_NAME*

Use the library `lib`*LIB_NAME*`.a`. This option is passed to the linker.

-o *OUT_FILE*

Name the final output file *OUT_FILE*. If no `-o` option is specified, the generated executable is called `a.out`. This option can also be used to name files created with the `-c` or `-T` options.

-C=*CHECK*

Compile in run-time checks specified by *CHECK*. Recognised values for *CHECK* are

`stack` Check that temporary arrays do not exhaust the amount of memory reserved for them. This amount can be increased (or reduced) using the `-stack` option.

-stack *STACK_SIZE*

Reserve *STACK_SIZE* words of memory for temporary arrays. The default is 250000.

-proc*RANK EXTENT* ...

In HPF `DISTRIBUTE` directives, the `ONTO` clause specifies a processor arrangement over which the array is to be distributed. The `ONTO` clause is optional. If it is omitted the compiler selects a default processor arrangement. The `-proc`*RANK* options allow the shape of the default processor arrangements to be specified. At most one default processor arrangement shape can be specified for each possible *RANK* between 1 and 7. The `-proc`*RANK* flag should be followed by precisely *RANK* numeric extent fields, defining the shape. For example

```
-proc2 5 6
```

specifies that default rank-2 processor array has shape (5,6).

-temp=*TEMP_DIR*

Set the directory used for temporary files to *TEMP_DIR*. The default is the current working directory.

-v

Verbose. Print the commands constructed by the `hpf` script as they are executed.

-w

Suppress warning messages.

-compOpts *COMP_OPTS*

Append the string *COMP_OPTS* to to the list of options passed to the target compiler.

-linkOpts *LINK_OPTS*

Append the string *LINK_OPTS* to to the list of options passed to the linker.

3 The example suite

The release comes with a set of example codes in the `examples` subirectory.

In this section we assume the provision of a script `hpfrun`. This runs an executable generated by the `hpf` command on a specified number of processes. An HPF program suitable for execution on 3 processors might be compiled and executed by

```
hpf test.hpf
hpfrun 3 a.out
```

The example codes are divided into three subdirectories

3.1 tests directory

Simple codes designed primarily to exercise the HPF run-time library. The codes

```
test0.h90
test1.h90
test2.h90
...
test6.h90
```

should be executed on 4 processes, eg

```
hpf -o test1 test1.h90
hpfrun 4 test1
```

The codes

```
test7.h90
test8.h90
test9.h90
...
```

should be executed on 6 processes, eg

```
hpf -o test7 test7.h90
hpfrun 6 test7
```

A script `runtests` will compile and execute all the programs in the directory.

3.2 parkbench directory

Kernels taken from the Parkbench benchmark suite. These are intermediate sized codes. The codes

```
fl.hpf
tm.hpf
aa.hpf
sh.hpf
ir.hpf
as.hpf
```

are suitable for execution on 4 processors, eg

```
hpf -o fl fl.hpf
hpfrun 4 fl
```

The codes

```
st.hpf
rd.hpf
```

are suitable for execution on 8 processors, eg

```
hpf -o st st.hpf
hpfrun 8 st
```

A script `runtests` will compile and execute all the programs in the directory.

3.3 genesis directory

Codes taken from the Genesis benchmark suite. These are small to medium-size application codes. Each program is split over several files in its own subdirectory. The subdirectories are

```
pde1
pde2
trans1
fft1
md1
```

`make` files are provided in the subdirectories. All codes are suitable for execution on 4 processors, eg

```
cd pde1
make
hpfrun 4 pde1
```

The HPF translator doesn't yet support the Fortran 90 intrinsic `DATA_AND_TIME`. A timer routine, `ctimer`, is provided. `ctimer` is written in C. In principle, this makes the programs non-portable, but the versions provided will probably work on most UNIX platforms. `ctimer` returns the time local to each process—it's result is not a globally consistent HPF value. The translator has no way of knowing this, so careless use of the returned value (for example, in control constructs) could cause run-time errors.

4 Language restrictions

The translator accepts the official 'Subset HPF' language (as defined in Section 8 of the 'HPF Language Specification') with some restrictions. It recognises *full* HPF (including full Fortran 90) and gives error messages upon encountering unsupported features. This section lists those features in Subset HPF that are *not* supported in this version (V0.1).

Incidentally, some Fortran 90 features that are not in strict Subset HPF, e.g. the `SELECT CASE` construct, *are* supported.

4.1 Fortran 77 features

Sequence and storage association

The `EQUIVALENCE` statement is not supported; partitioning of common blocks differently in different program units is not supported.

Associating an array *element* actual argument with an array dummy argument is not supported; reshaping of arrays across procedure interfaces is not supported.

DATA statements and BLOCK DATA

Neither are supported.

I/O

Implied `DO` loops in `READ`, `WRITE` or `PRINT` statements are not supported.

The "`ERR =`", "`END =`" and "`EOR =`" specifiers in I/O statements, are not supported (if the program is executed on more than one abstract processor.)

CHARACTER variables

Variables, named constants and functions of `CHARACTER` type are not supported (though character literal constants and expressions *are* supported).

Specification of `CHARACTER` type in an `IMPLICIT` statement is not supported.

Assumed size arrays

Fortran 77 *assumed size* arrays are not supported. These are dummy arguments declared with an asterisk in their extent field. (Fortran 90 assumed *shape* arrays *are* supported.)

Specification expressions

A *specification expression* is an expression used in a declaration, for example, as an array extent. The HPF implementation does not support references to arrays or references to distributed data in specification expressions.

Termination of a DO loops

Termination of a `DO` loop at a statement other than `ENDDO` or `CONTINUE` is not supported.

Statement functions

Statement functions—"in-line" definitions of functions within other procedures—are not supported.

ENTRY statement

Multiple entry points into a single procedure are not supported.

4.2 Fortran 90 features in Subset HPF

Array constructors

Array constructors—lists of values between the brackets `(/ /)`, used to represent one-dimensional arrays—are not supported.

ALLOCATABLE arrays

`ALLOCATABLE` arrays and the `ALLOCATE` and `DEALLOCATE` statements are not supported.

Initialisation of variables in type declarations

This is not supported.

4.3 HPF features in the Subset

SEQUENCE directive

The `SEQUENCE` directive is not supported.

Distribution of PARAMETERs

Explicit distribution of `PARAMETER` values is not supported.

Inherited and transcriptive mappings

The HPF `INHERIT` attribute or directive is not supported.

Transcriptive distribution format and transcriptive distribution target are not supported. These are formats and targets consisting of a single asterisk.

FORALL statement

In a `FORALL` statement, references to the assignment variable or any part of it in a subscript expression, the right-hand-side expression, or the mask expression, are not supported.

4.4 Miscellaneous intrinsic procedures

The following intrinsic functions and subroutines are not supported.

Fortran 77 string-handling and inquiry functions

`REPEAT`, `TRIM` and `LEN`

Array inquiry functions

`ALLOCATED`, `LBOUND`, `SHAPE`, `SIZE` and `UBOUND`

Array construction functions

`PACK` and `UNPACK`

Intrinsic subroutines

DATE_AND_TIME, MVBITS, RANDOM_NUMBER, RANDOM_SEED and SYSTEM_CLOCK

HPF system inquiry intrinsic functions

NUMBER_OF_PROCESSORS and PROCESSORS_SHAPE

Other usage restrictions

The following intrinsic functions can only appear as primaries on the right-hand-side of an assignment statement (and not in any other context):

MATMUL, SPREAD, RESHAPE, TRANSPOSE

5 Known bugs

1. The hpf script executes very slowly if the release is configured to use GNU g++ as the linker.
2. The code examples/fft1 hangs under the EUI-based MPICH implementation, running on the SP2. This is believed to be due to a problem with MPI_Waitany on that platform. This problem is under investigation.
3. The code examples/fft1 hangs intermittently under the P4-based MPICH implementation, running on an RS6000 workstation. This problem is under investigation.

References

[1] *High Performance Fortran Language Specification.* High Performance Fortran Forum.