VFC 2.0
Compilation System for HPF+
Installation Notes – NEC SX-5

Co-operation NEC CCRLE/Uni-Vienna (ADVANCE Project)
Deliverable 1

Siegfried Benkner, Erwin Laure, Viera Sipkova

Institute for Software Technology and Parallel Systems
University of Vienna
Liechtensteinstrasse 22
A-1090 Vienna, AUSTRIA
sigi@ieee.org

February 2000

Contents

1 Overview

This document describes the current release of the HPF+ compilation system VFC on the NEC SX-5. We give an overview of the main features of VFC and describe its use on the NEC SX-5 at NLR (`super.nlr.nl`), Netherlands.

VFC is a command-line compiler, that translates Fortran 90/HPF/HPF+ programs to Fortran 90/MPI message passing programs. The generated SPMD message passing codes are compiled with the SX/Fortran 90 compiler `sf90` and linked with the VFC runtime system and the SX-5 MPI libraries to produce an executable parallel program.

2 Using VFC on the NEC SX-5

HPF/HPF+ programs are compiled using the command `vfc`, which invokes a script that calls the VFC compiler, then the F90/SX compiler, and finally the linker, automatically linking in the necessary runtime libraries. The result of the compilation, if successful, is an executable file which can be run by means of `mpirun`.

In order to use VFC on the front-end of the SX-5 at NLR (katayama) the following alias has to be defined:

```
alias vfc /shared/home/nec/vsipkova/VFC/bin/vfcgo
```

The VFC command-line syntax is

vfc [*options*] *filename* ...

Where:

options is zero or more command-line options (see below).

filename is the name of a source file. More than one filename may be specified.

Options is zero or more of:

`-f` free source form

`-i` *path* Add *path* to the list of directories in which to search for `INCLUDE`-files or `MODULE`s.

`-o` *out* Name of output files or output file directory (depending on `-s` option).
If `-o` is specified but not `-s` the final executable file is named *out*
and the intermediate Fortran 90 message passing program *out*`_vfc.f90`
instead of `a.out` and `a_vfc.f90`.
If `-o` is specified together with `-s`, the intermediate message passing programs
are generated in the directory *out* (see below).

(*Options* continued) ...

`-s` Stop after generation of the intermediate message passing Fortran 90 code.
For each input file *name.ext* an output file with name *name*`_vfc.f90` is generated in the current directory, or in the directory *out* if `-o` *out* has been specified.

`-b` *n* Specify buffer allocation ratio. *n* is a positive integer number.
It should be set according to allocation ratio, if a program stops with the message: "DVIEW: not enough space in buffer - allocation ration *n*".

`-d` Dynamic buffer management. (May be used alternatively to "-b ", when a program stops with the message: "DVIEW: not enough space in buffer")

`-h` Enable the use of HPF+ halo features.

3 Executing programs parallelized with VFC on the NEC SX-5

On the NEC SX-5 both the VFC compiler as well as the runtime system is available. The VFC compiler, however, can be executed only on the SUN front-end `katayama`.

In the following we show an example of how HPF programs parallelized by VFC can be executed on the SX-5. Please consult the corresponding web-pages or related documentation for more information.

Compilation:

```
katayama> vfc -o test_prog test_prog.hpf
```

Execution:

```
katayama> qsub test_script
```

For example, What follows is an example for shell-script to run the `test_prog` located in the directory `/shared/home/nec/someuser` on 4 processors of the SX-5 in batch-mode.

```
#!/bin/ksh
#@$-lT 0:15:00
#@$-lu 4
#@$-lM 300mb

F_PROGINF=DETAIL
export F_PROGINF
cd /shared/home/nec/someuser

mpirun -np 4 test_prog
```

4 Language Features

VFC recognizes full Fortran 90, full HPF-2, many of the HPF-2 Approved Extensions, and most of the HPF+ language features. Fortran 90 program constructs for which no HPF/HPF+ directives have been specified and that do not involve *mapped objects*[1] are fully supported. On program constructs that involve *mapped objects* the system imposes some restrictions which are indicated to the user in form of *restriction messages*. The major restrictions are described in Section 4.14.

4.1 Processors Statements

- `PROCESSORS` directive
- `NUMBER_OF_PROCESSORS()` intrinsic function

The HPF-2 `PROCESSORS` directive is fully supported. However, VFC requires that an input program contains at least one processor array declaration.

The intrinsic function `NUMBER_OF_PROCESSORS()` may be used to specify processor arrays whose sizes are determined upon program start. The intrinsic function `RESHAPE` may be used to establish a relationship of processor arrays of different rank according to the Fortran *array element order*.

4.2 Distribute Directives

For arrays of type `INTEGER, REAL, DOUBLE PRECISION, COMPLEX(KIND=4), COMPLEX(KIND=8), LOGICAL`, `CHARACTER`, and for arrays of *derived type*, arbitrary, multi-dimensional distributions may be specified by means of the `DISTRIBUTE` directive using any combination of the following distribution formats:

- `BLOCK` (HPF-2)
- `CYCLIC` (HPF-2)
- `GEN_BLOCK` (HPF+, HPF-2 Appr. Ext.)
- `INDIRECT` (HPF+, HPF-2 Appr. Ext.)
- `*` (non-distributed dimensions)

Note that the number of distributed dimensions of an array (i.e. those dimensions for which a distribution format other than '*' has been specified) must match the rank of the processor array to which the distribution refers. All expressions appearing as arguments of the above distribution formats must be *specification expressions* which must not reference mapped objects. If a `DISTRIBUTE` directive does not contain an `ONTO` clause the system attempts to select one of the available processor arrays. If no suitable processor array is found, an appropriate error message is issued.

[1] A mapped object is an object that appears in a `DISTRIBUTE` directive or `ALIGN` directive, or that has the `INHERIT` or `DYNAMIC` attribute.

Example 1: HPF+ Distribution Directives

```
      INTEGER A2(100,100), A3(100,100,100)

!HPF$ PROCESSORS R(4)
!HPF$ PROCESSORS R2(2,2) = RESHAPE(R)

!     multidimensional distributions
!
!HPF$ DISTRIBUTE(BLOCK,*,CYCLIC) ONTO R2 :: A3


!     general block distribution
!
      INTEGER, PARAMETER :: BS(4) = (/ 30, 20, 20, 30 /)
!HPF$ DISTRIBUTE(GEN_BLOCK(BS), *) ONTO R1 :: A2


!     indirect distributions
!
      INTEGER MAP(100)          ! must be an array constant or dummy argument
!HPF$ DISTRIBUTE(INDIRECT(MAP),*) :: A2
```

■

The above data distribution features may be applied also to *dynamically allocated arrays*, i.e. arrays that have the ALLOCATABLE or POINTER attribute or to *automatic arrays*. Moreover the distribution of components of derived types is supported as described in Section refderived.

The distribution of an array may also be specified by means of the ALIGN directive as described next.

4.3 Align Directives

Alignment may be used to specify that an array or an array dimension is distributed in the same way as another array or array dimension. An alignment relationship between two arrays is fixed and cannot be destroyed at runtime and thus may be used to specify groups of arrays that are always distributed in the same way, even in the context of dynamic data distribution.

The alignment mechanisms available in VFC support perfect alignments, either of whole arrays or of individual array dimensions, and collapsing of individual dimensions of the *alignee*. Note that VFC supports neither alignment offsets nor alignment strides, and it does not support the use of colons in alignment specifications.

Alignments as currently provided by VFC are shown in Example 2.

Example 2: HPF+ Alignment Directives

```
REAL, DIMENSION(N)   :: A1, B1
REAL, DIMENSION(N,M) :: A2, B2

!HPF$ PROCESSORS R(NUMBER_OF_PROCESSORS())
!HPF$ DISTRIBUTE(BLOCK)   :: B1
!HPF$ DISTRIBUTE(BLOCK,*) :: B2

! perfect alignment
!HPF$ ALIGN A2 WITH B2

! alignment of individual array dimensions
!HPF$ ALIGN A2(I,J) WITH B2(I,J)
!HPF$ ALIGN A2(I,*) WITH B2(I,*)

! collapsing a dimension of the alignee
!HPF$ ALIGN A2(I,*) WITH B1(I)

! replication over a dimension of the align-target
!HPF$ ALIGN A1(I) WITH B2(I,*)
```

■

4.4 Halos

The (super) set of non-local elements of a distributed array that may be accessed during the execution of the program may be specified by the user by defining *halo regions* by means of the `HALO` directive or attribute. The `UPDATE_HALO` directive is provided in order to update halo regions and the `REDUCE_HALO` directive to perform reduction operations on halos.

Example 3: The `HALO` Directive

```
!HPF$ PROCESSORS, DIMENSION(NUMBER_OF_PROCESSORS()) :: R
!HPF$ DISTRIBUTE (INDIRECT(map)) ONTO R :: a
!HPF$ HALO (halo_function(I)) ON R(I)   :: a
      ...
!HPF$ UPDATE_HALO :: A
      ...
!HPF$ REDUCE_HALO :: A
```

The set of non-local indices of array `a` *that may be accessed during execution of the program is obtained by evaluating the array-valued function* `halo_function` *on each processor with the corresponding processor index* `I`*.* ■

If an array is *perfectly aligned* with another array that has a halo, then the aligned array takes over the halo of the align target.

For arrays with the DYNAMIC attribute a halo may also be specified at the time of redistribution, by specifying the HALO attribute within the REDISTRIBUTE directive.

4.5 Dynamic Data Distribution

VFC provides the following language features for dynamic data distribution.

- DYNAMIC attribute and directive
- REDISTRIBUTE statement
- RANGE attribute

For non-formal arrays the DYNAMIC attribute may be combined with the DISTRIBUTE directive in order to provide an *initial distribution.* The REDISTRIBUTE statement supports the same distribution formats as outlined in section 4.2. The RANGE attribute may be used to provide the compiler with information on the set of *permissible distributions* an array may be associated with.

The REDISTRIBUTE statement may be combined with the HALO attribute in order to specify or change the *halo region* of a dynamically distributed array.

Note that an array that has the DYNAMIC attribute may not be accessed (or referenced in an ALLOCATE statement) unless it has been distributed (either by providing an initial distribution, by means of the REDISTRIBUTE directive, or as the result of a procedure call).

Example 4: Dynamic Data Distribution

```
!HPF$ PROCESSORS R(NUMBER_OF_PROCESSORS())

!HPF$ DYNAMIC           :: a, b
!hpf$ DISTRIBUTE(CYCLIC) :: b

INTEGER, POINTER, DIMENSION(:)   :: a,b

READ(*,*) map                            ! read map array

!HPF$ REDISTRIBUTE(INDIRECT(map)) :: a   ! redistribute dynamically

ALLOCATE (a(size(map)),b(n))
```

This program would not execute successfully if the order of the REDISTRIBUTE *statement and the* ALLOCATE *statement were reversed.* ∎

4.6 Distribution of Dummy Arrays

The distribution of dummy arrays may specified in one of the following ways:

1. With the `INHERIT` attribute, possibly accompanied by the `RANGE` attribute.
2. With the `DYNAMIC` attribute. In this case the corresponding actual array must also have the `DYNAMIC` attribute. This feature may be used to define or modify the distribution of an array as the result of a procedure call.
3. With the `DISTRIBUTE` directive. In this case the distribution of the corresponding actual array must not be dynamic and must always match the distribution of the dummy array. (This is checked by the system.)

Note that dummy arrays for which a distribution has been specified in one of the above ways must always be *assumed shape arrays*. As a consequence, the interface of any procedure that contains mapped formal arguments must be explicit. This can be achieved by either specifying an *interface block* or by using *internal procedures*.

The Fortran 90 intrinsic functions `SIZE`, `LBOUND`, `UBOUND`, and `SHAPE` are fully supported as long as the array argument is a whole array.

4.7 Distribution of Derived Type Components

VFC supports the distribution of derived type components. As in HPF-2, only one distribution level is allowed within derived types. More specifically, a derived type is said to be an *explicitly mapped type* if any of its components is explicitly mapped or an explicitly mapped type. Distribution of a derived type component is only allowed if it is not explicitly mapped.

For an explicitly mapped ultimate component (i.e. a component of intrinisc type) of a derived type VFC's standard distribution techniques are applied.

Example 5: Distributed Derived Type Components

```
!HPF$ PROCESSORS R(NUMBER_OF_PROCESSORS())

      TYPE foo
          REAL              :: R
          INTEGER, POINTER  :: P(:)
!HPF$     DISTRIBUTE(BLOCK) :: P
        END TYPE foo
```

■

4.8 Distribution of Arrays of Derived Type

VFC adopts an *unrolling* strategy for implementing the distribution of arrays of derived type. More specifically, a distributed derived type array is transformed to a set of intrinsic data type arrays, each of which conforms to an ultimate component of the derived type. These arrays are subsequently distributed using VFC's standard techniques.

Example 6: Transformations of Derived Type Arrays

```
      TYPE foo
         REAL :: bar1
         REAL :: bar2(N)
      END TYPE foo

      TYPE(foo) :: x(NN), y(NN)

!HPF$ DISTRIBUTE(BLOCK) :: x, y
```

■

4.9 Redistribution at Procedure Boundaries

If a mapped actual array is passed to a dummy array for which no mapping has been specified the system redistributes the actual argument such that it is *replicated* on each processor prior to the call and *restores* the original distribution upon exit from the procedure.

Passing a mapped actual argument to a formal argument with a different mapping is not yet supported and is indicated to the user via a corresponding restriction message.

4.10 Independent Do Loops

The `INDEPENDENT` directive may be used in order to indicate to the system that a `DO` loop does not contain loop carried dependencies, and thus the loop iterations may be executed in any order.

The following clauses may be specified within the `INDEPENDENT` directive:

- the `ON HOME` clause, to specify the distribution of loop iterations.
- the `REDUCTION` clause, for reduction operations.
- the `NEW` clause, to declare *private* variables.
- the `REUSE` clause with or without a logical condition, for *schedule reuse*.
- the `RESIDENT` clause with or without a list of array names, to avoid communication.

All `INDEPENDENT` loops are currently parallelized by applying the *inspector/executor strategy*. This version of the system imposes a number of restrictions on the use of distributed arrays within such loops (see below). Whenever one of these restrictions is violated an `INDEPENDENT` loop might be sequentialized. If this is the case, it is indicated to the user by means of a *diagnostic message*.

The parallelization of `INDEPENDENT` loops is currently subject to the following restrictions:

- The bounds-expressions of loops that are nested within `INDEPENDENT` loops must not reference outer loop variables or distributed objects.
- `INDEPENDENT` loops and any loops that are nested within an `INDEPENDENT` loop must not have a step-size other than 1.
- Procedures called from within an `INDEPENDENT` loop must be `PUREST`.
- The `SELECT`, `REDISTRIBUTE`, `ALLOCATE` and `DEALLOCATE` statements are not yet supported within `INDEPENDENT` loops.

4.11 Array Assignment Statements

Array assignment statements which do not reference mapped objects are supported in the full generality of Fortran 90.

Array assignment statements referencing mapped arrays are currently handled only if the following restriction (which is checked by the system) is satisfied:

- The left hand side is a *whole array* (e.g. A) or an array section where distributed dimensions are accessed as a whole and the right hand side expression is either
  - a scalar expression,
  - or an array expression where all referenced non-scalar objects are whole arrays or array sections where distributed dimensions are accessed as a whole and each distributed dimension on the right hand side is of the same size and distributed in exactly the same way as the corresponding dimension of A.

- The left hand side is a distributed array section and the right hand side is
  - a scalar expression,
  - or a distributed array expression, where all arrays are of the same rank as the corresponding left hand side array. Sections of not-mapped arrays are not allowed to be referenced in the right hand side expression.
    For handling such array assignments the `ADLIB` runtime library is employed by VFC. Since the `ADLIB` library is restricted to `BLOCK` and `CYCLIC` distribution, arrays with `INDIRECT` or `GEN_BLOCK` distributions cannot be handled in this context. Moreover, `ADLIB` requires the local shape of the array to be equal on all processors. If a mapped array occurs both in this kind of array assignment of this kind and in independent do loops, dynamic buffer handling is required (see VFC command-line option: '-d'). Additional overheads may be caused if an array is both used in array assignments of this kind and independent do loops.

4.12 The `PURE` Attribute and the `PUREST` Directive

The `PURE` attribute is recognized, however the constraints imposed by it are currently not checked by the system. The `PUREST` directive may be specified for procedures that exclusively operate on local data.

4.13 Extrinsic Procedures

The `EXTRINSIC(HPF_LOCAL)` and `EXTRINSIC(HPF_SERIAL)` prefix for procedures and the associated argument transfer mechanisms are supported by VFC.

4.14 Restrictions

This section lists those features of HPF-2 which are not yet supported in this version of VFC.

Note that VFC currently provides only **limited support for an efficient handling of regular codes** and, in particular, does not automatically parallelize loops that are not proceeded by an `INDEPENDENT` directive.

The main focus has been on the parallelization of codes which are characterized by independent loops exhibiting indirect, possibly dynamically changing, array accesses. All `INDEPENDENT` loops are currently parallelized based on runtime-compilation techniques (some special cases are recognized by the compiler and optimized). Consequently, a large overhead may be introduced for loops that do not contain indirect array references. The same is true for `INDEPENDENT` loops that are sequentialized because some restrictions are violated.

- Data distribution
  - Only the distribution of objects of type `INTEGER, REAL, DOUBLE PRECISION, COMPLEX(KIND=4), COMPLEX(KIND=8), LOGICAL`, `CHARACTER`, and for objects of *derived type*, is fully supported.
  - Recursive derived types may not be a component of an explicilty mapped derived type.
  - Arrays of recursive derived types may not be distributed.
  - `TEMPLATES` are not supported.
  - Only *basic alignments* are supported; i.e. each *align-source* must be an *align-dummy* or '*', each align-subscript must be a *align-dummy* or '*', and the *align-target* must not have the `ALIGN` attribute.
  - *Prescriptive*, *descriptive* and *transcriptive* mappings for dummy arguments are not yet supported.
  - A distributed array must not have the `SAVE` attribute.
  - The declared lower bound of all distributed arrays must be 1.

- Other HPF features
  - The HPF intrinsic and library procedures are not yet supported, with the exception of `NUMBER_OF_PROCESSORS()`.
  - The HPF extrinsic library is not yet supported.
  - The `FORALL` statement and `FORALL` construct is not yet supported.
  - The HPF `SEQUENCE` directive is not supported.

- Input/Output
  - Implied-DO list in not-parallel I/O statements for mapped objects is not supported.
  - Implied-DO list in parallel I/O statement is only partially supported (*do-object-list* may consist only of one object).

- Fortran 77 features
  - The `EQUIVALENCE` statement for mapped objects is not supported.
  - Statement functions are not supported.
  - Arithmetic `IF` statements are not supported.
  - Specifying different distributions in different program units for objects of `COMMON` blocks is not supported.

- Fortran 90 features
  - The `WHERE` statement and construct are not yet supported if they reference mapped objects.
  - The pointer assignment statement is not yet supported.
  - The array assignment statement where only one - the left hand side or the right hand side is distributed, is not yet supported.
  - Array construction functions (`PACK` and `UNPACK`) are not yet supported for mapped arrays.
  - The `ALL`, `ANY`, `COUNT`, `CSHIFT`, `DOT_PRODUCT`, `EOSHIFT`, `MATMUL`, `MAXLOC`, `MAXVAL`, `MINLOC`, `MINVAL`, `MERGE`, `RESHAPE`, `SPREAD`, `TRANSFER`, and `TRANSPOSE` intrinsic functions are not yet supported for mapped arrays.
  - Mapped objects must not be referenced in *specification expressions*.
  - The transfer of sections of mapped arrays (other than whole arrays) to procedures is not yet supported.