UDC 681.325.3

Code Generator for HPF Library on Fujitsu VPP5000

●Matthijs van Waveren ●Cliff Addison ●Peter Harrison ●Dave Orange
●Norman Brown

(Manuscript received October 23, 1999)

The Fujitsu VPP5000 supports the data parallel language High Performance Fortran (HPF). The HPF Library gives a user access to intrinsic functions that are particularly useful in a data parallel environment. The implementation of HPF Library presents the challenge that all data types, all data kinds, all array ranks and all input distributions need to be supported. The number of specific functions runs into the billions so it is not feasible to code each individually. This paper presents a method to solve this problem. We have developed a library generator, which consists of templates and a template processor along with an interface to the HPF compiler. We show that instead of implementing *billions* of specific functions, we only need to implement *five* templates.

1. Introduction

The VPP5000 is the latest generation of Fujitsu VPP supercomputers. The basic architecture of the system is very similar to the VX/VPP300/VPP700 discussed in Refs. 1), 2), and 3) in that it comprises a group of vector processors linked via a crossbar switch. There are, however, several important differences, which are listed in Chapter 2.

The VPP5000 supports the data parallel language High Performance Fortran (HPF), version 2.0, which is based on Fortran 95. The language specification of HPF 2.0 is described in Ref. 4), and of Fortran 95 in Ref. 5). The data parallel programming model of HPF is single-threaded, with a global name space, and loosely synchronous parallel computation. HPF 2.0 consists of a core language and approved extensions. One of the notable approved extensions supported by the Fujitsu HPF compiler is task parallelism, as described in Section 9 of Ref. 4). The HPF/JA extensions (Ref. 6) for optimisation of communication and for enlargement of description capability are also supported.

One of the language features of the core language of HPF 2.0 is a library of intrinsic functions, called HPF Library. These intrinsic functions allow a user to develop portable data parallel implementations of highly irregular problems, as described by Hu et al. (Ref. 7) HPF Library consists of 55 generic functions. The implementation of this library presents the challenge that all data types, data kinds, array ranks and input distributions need to be supported. For instance, more than 2 billion separate functions are required to support COPY_SCATTER when the full range of data types, data kinds and array ranks is considered. The efficient support of these billions of specific functions is one of the outstanding problems of High-Performance Fortran, as mentioned by Professor Ken Kennedy at a recent HPF User Group meeting.

This paper presents a method to solve the problem of the astronomical number of specific functions. We have developed a library generator, which uses templates. The library generator consists of a template processor and templates, and has an interface to the HPF compiler. We show that instead of implementing *billions* of specific functions, we only need to implement *five* templates.

This paper consists of the following chapters: Chapter 2 describes the Fujitsu VPP5000 Vector-Parallel Processor; Chapter 3 describes the functions in the HPF Library; Chapter 4 describes the design of the Library Generator; Chapter 5 gives some examples of code generation; and Chapter 6 is the conclusion.

2. Fujitsu VPP5000 Vector-Parallel Processor

The differences between the VPP5000 system and the VX/VPP300/VPP700, discussed in Ref. 1) and 2) are:

- The clock pulse has been reduced to 3.3 ns (300 MHz) and the width of the vector pipes increased from 8 to 16, giving a single-node performance of over 9.6 Gflops.
- To accommodate the increased processor performance, the interconnect bandwidth has been increased to over 1.6 Gbyte/s bi-directional.
- The design of the vector processor has changed with the number of vector pipes reduced from seven to four: a single load/store pipe, a multiply and/or add pipe, a divide and/or square root pipe and a mask pipe.
- The scalar processor has been substantially enhanced, having a peak performance of 1.2 Gops and now including a 2 Mbytes 4-way set associative secondary cache.
- The memory system has been improved to provide some degree of memory caching and also includes special hardware to handle efficiently certain data access patterns. Each processor can be configured with up to 16 Gbytes of 45 ns SDRAM. The memory to CPU bandwidth has increased to 72.8 Gbyte/s.
- To allow for the larger per-node memory size, the operating system has full 64 bit addressing capability.

3. HPF Library functions

The HPF Library is described in Section 7 of the High Performance Fortran Language Specification (Ref. 4). It consists of five groups of procedures, as listed below.

The number of specific functions in the list is calculated on the basis of data types, data kinds and array ranks. The Fujitsu VPP5000 supports 4 kinds of INTEGER, 3 kinds of REAL, 3 kinds of COMPLEX, 4 kinds of LOGICAL, and one kind of CHARACTER. Fortran 95 specifies that the rank of an array can have a maximum value of seven.

- Array Combining Scatter Functions. This group consists of 12 generic functions. The number of specific functions is approx. 10 billion (9 680 449 961).
- Array Prefix Functions. This group consists of 12 generic functions. The number of specific functions is 378 875.
- Array Suffix Functions. This group consists of 12 generic functions. The number of specific functions is 378 875.
- Array Reduction Functions. If we include the Fortran 95 array reduction functions (Chapter 13 of Ref. 5) in this group, there are 11 generic functions. The number of specific functions is 15 050.
- Array Sorting Functions. This group consists of 4 generic functions. The number of specific functions is 1120.

The numerical algorithms used in the implementation of the HPF Library functions are described in Ref. 8). The selection criteria are:

- Speed and scalability with respect to problem size on a single VPP5000 processor;
- Conducive to running in parallel on several processors with minimal communications;
- Able to perform the operations on all required data types, data kinds, array ranks, and array distributions.

This paper focuses on the method used to solve the problem of the implementation of the billions of specific functions.

4. Design of library generator

The library generator was designed to minimise the number of templates, by making them as independent as possible of type, kind, rank and distribution. The consequence is that the development cost is significantly reduced and the product is easily maintainable.

4.1 Overview

The library generator consists of two main components, as shown in **Figure 1**:

- Templates
- Template processor

When the HPF compiler encounters a call to a HPF Library function in the user code, it calls the library generator and passes the following information in the interface:

- Name of the required function.
- Type, kind, rank, upper bound and lower bound of all dummy arguments.
- Arrangement of the processors.
- Distribution of the dummy arguments.

The template processor then reads in the template corresponding to the required function and generates the function code corresponding to the properties of the dummy arguments. The function code is passed back to the compiler through the interface.

4.2 Function templates

The function templates encapsulate the algorithms for the HPF Library functions. They contain the following language items:

- Fortran 95 code, as described in Ref. 5).
- HPF 2.0 directives, as described in Ref. 4).
- HPF/JA directives, as described in Ref. 6).
- CPP , C pre-processing, directives, as described in Ref. 9).
- Template parameters.
- Template macros.

Only five templates are necessary for the HPF Library, one for each of the five groups of procedures listed in Chapter 3.

4.2.1 Template parameters

The template parameters correspond to the properties of the dummy arguments. The template parameters listed in **Table 1** are implemented. They start and end with an at-sign, in order for

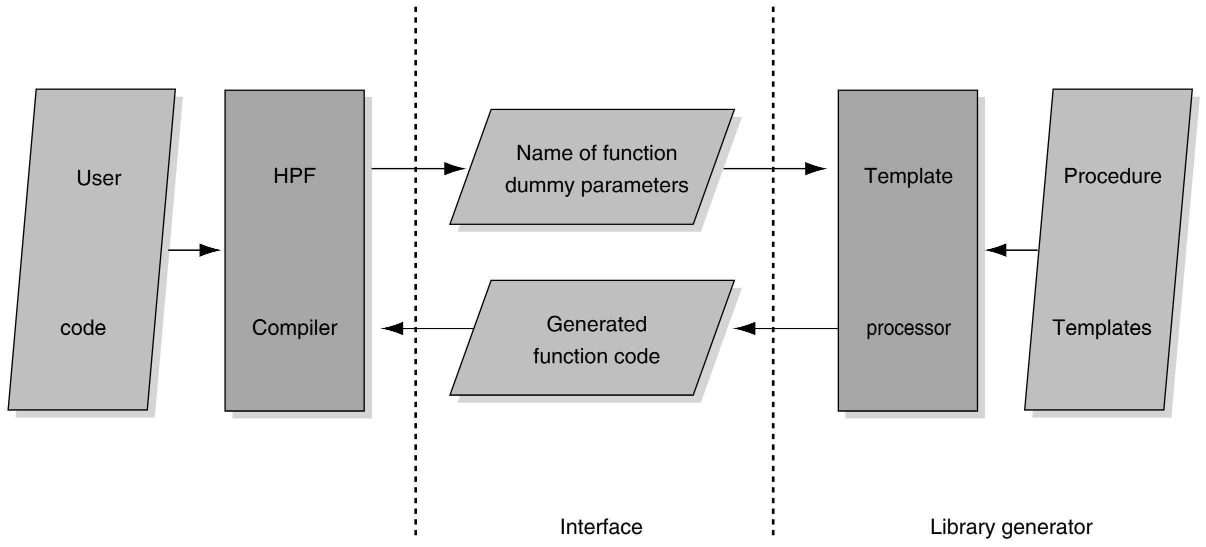


Figure 1
Overview of integrated HPF compiler-library generator system.

the template processor to distinguish them from Fortran 95 variables.

4.2.2 Template macros

We have developed a set of template macros, in order to generate instances of character strings used to form the varying parts of the function code. The following template macros are implemented:

- Generation of a list of characters, separated by commas.
- Generation of a list of variable names:
  - with or without a colon;
  - between brackets;
  - possibly omitting one;
  - with a comma.
- Generation of a list of size specification indexes:
  - possibly omitting one;
  - with lower and upper bounds;
  - with a comma;
- Generation of a dimension attribute:
  - possibly omitting one rank;
  - with a list of colons.
- Generation of an index list for array syntax.
- Generation of the first line of a multi-dimensional FORALL construct.

4.2.3 Loops

The FDO construct has been introduced, in order to generate lines of pseudo-code and to be able to loop over the above-mentioned template macros.

The syntax of the FDO construct is as follows:

fdo-construct	is	FDO *loop-control*
		fdo-block
		END FDO
loop-control	is	*fdo-var = const, const, const*
const	is	*signed-int-literal-constant*
fdo-var	is	^ *scalar-int-variable* ^

The FDO construct is only allowed in templates. The *fdo-block* may contain any of the language items allowed in templates. Nested FDO constructs are allowed. The execution of the FDO construct follows the same execution rules as the Fortran 95 DO construct (Ref. 5).

Table 1
List of template parameters.

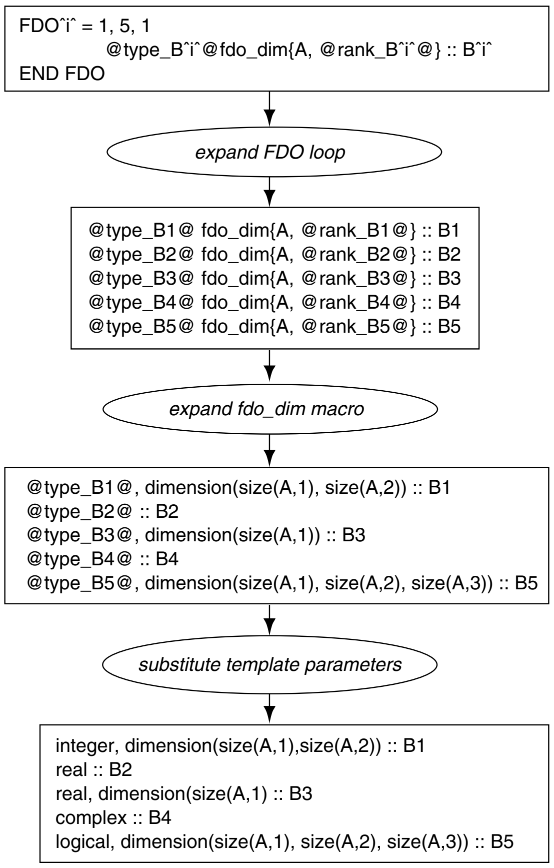
Template parameter	Description
@type_xxx@	Type of parameter xxx
@kind_xxx@	Kind of parameter xxx
@rank_xxx@	Rank of parameter xxx
@lbound_n_xxx@	Lower bound in dimension n of variable xxx
@ubound_n_xxx@	Upper bound in dimension n of variable xxx
@function_name@	Unique name of the function
@rank_processors@	Rank of the PROCESSOR array
@lbound_n_processors@	Lower bound in dimension n of the PROCESSOR array
@ubound_n_processors@	Upper bound in dimension n of the PROCESSOR array
@name_template_xxx@	Name of the template for parameter xxx
@rank_template_xxx@	Rank of the template for parameter xxx
@lbound_n_template_xxx@	Lower bound in dimension n of the template for parameter xxx
@ubound_n_template_xxx@	Upper bound in dimension n of the template for parameter xxx
@format_n_template_xxx@	Format of the distribution in dimension n of the template for parameter xxx
@size_n_template_xxx@	Distribution blocksize in dimension n of the template for parameter xxx
@format_n_xxx@	Format of the distribution in dimension n for parameter xxx
@size_n_xxx@	Distribution blocksize in dimension n of parameter xxx

4.3 Template processor

The template processor has the following processing sequence:

1) Obtain name of required function, properties of the dummy arguments, and arrangement of the nodes from the interface.
2) Convert the properties of the dummy arguments into template parameters and/or CPP predefines.
3) Generate a specific name for the function.
4) Read in the corresponding function template.
5) Process the CPP directives.
6) Expand the FDO loops.
7) Expand the template macros.
8) Substitute the template parameters.
9) Pass the generated code through the interface to the compiler.

5. Examples of code generation



```
FDOˆiˆ = 1, 5, 1
          @type_Bˆiˆ@fdo_dim{A, @rank_Bˆiˆ@} :: Bˆiˆ
END FDO
```

```
@type_B1@ fdo_dim{A, @rank_B1@} :: B1
@type_B2@ fdo_dim{A, @rank_B2@} :: B2
@type_B3@ fdo_dim{A, @rank_B3@} :: B3
@type_B4@ fdo_dim{A, @rank_B4@} :: B4
@type_B5@ fdo_dim{A, @rank_B5@} :: B5
```

```
@type_B1@, dimension(size(A,1), size(A,2)) :: B1
@type_B2@ :: B2
@type_B3@, dimension(size(A,1)) :: B3
@type_B4@ :: B4
@type_B5@, dimension(size(A,1), size(A,2), size(A,3)) :: B5
```

```
integer, dimension(size(A,1),size(A,2)) :: B1
real :: B2
real, dimension(size(A,1) :: B3
complex :: B4
logical, dimension(size(A,1), size(A,2), size(A,3)) :: B5
```

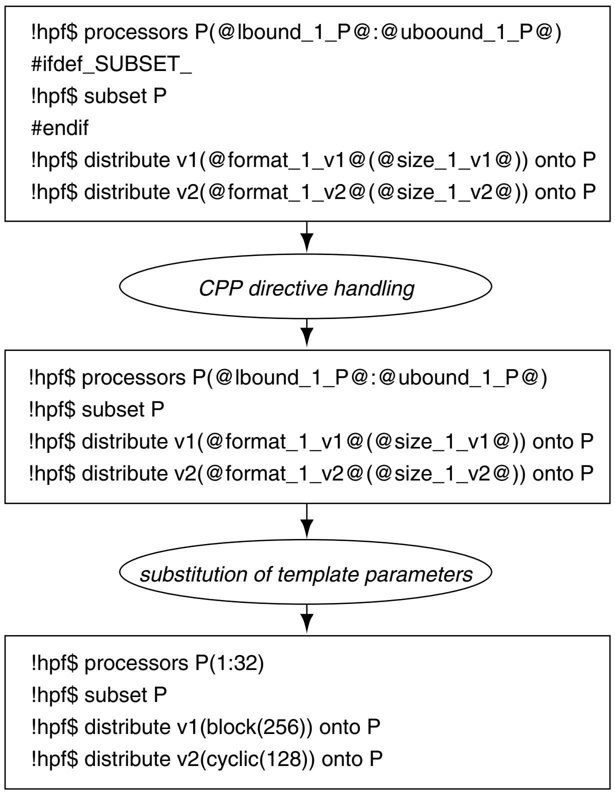
Figure 2
Generation of declaration statements.

6. Conclusion

One of the major difficulties with High Performance Fortran (HPF) is the efficient generation of HPF Library functions. We have solved this problem by developing a library generator so that we only need to implement *five* templates, one for each of the five groups of procedures of HPF Library.

This paper describes the design of this library generator. It consists of templates and a template processor. The new language items of template parameters, template macros, and the FDO construct have been designed and implemented. These language items lead to a significant reduction in the amount of code that needs to be written.

Our method leads to a significant reduction in development cost and to an easily maintainable product.



```
!hpf$ processors P(@lbound_1_P@:@uboound_1_P@)
#ifdef_SUBSET_
!hpf$ subset P
#endif
!hpf$ distribute v1(@format_1_v1@(@size_1_v1@)) onto P
!hpf$ distribute v2(@format_1_v2@(@size_1_v2@)) onto P
```

```
!hpf$ processors P(@lbound_1_P@:@ubound_1_P@)
!hpf$ subset P
!hpf$ distribute v1(@format_1_v1@(@size_1_v1@)) onto P
!hpf$ distribute v2(@format_1_v2@(@size_1_v2@)) onto P
```

```
!hpf$ processors P(1:32)
!hpf$ subset P
!hpf$ distribute v1(block(256)) onto P
!hpf$ distribute v2(cyclic(128)) onto P
```

Figure 3
Generation of HPF directives.

References

1) N. Uchida: Hardware of VX/VPP300/VPP700 series of vector-parallel supercomputer systems. *Fujitsu Sci. Tech. J.*, **33**,1, pp.6-14 (1997).
2) Y. Koeda: Operating system of the VX/VPP300/VPP700 series of vector-parallel supercomputer systems. *Fujitsu Sci. Tech. J.*, **33**, 1, pp.15-24 (1997).
3) E. Yamanaka and T. Shindo: Parallel Language Processing System for High-Performance Computing. *Fujitsu Sci. Tech. J.*, **33**, 1, pp.39-51 (1997).
4) High Performance Fortran Forum: High Performance Fortran Language Specification. Version 2.0, 31 January 1997.
5) ISO/IEC 1539-1: 1997 Information Technology – Programming Languages – Fortran.
6) Japanese Association for High Performance Fortran: HPF/JA Language Specification. Version 1.0, 1999.
7) Y.C. Hu, S.L. Johnsson, and S.-H. Teng: High Performance Fortran for Irregular Problems. in Proceedings of the 6th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Las Vegas, Nevada, June 1997.
8) M. van Waveren, C. Addison, P. Harrison, D. Orange, and N. Brown: HPF Library on the Fujitsu VPP5000. Proceedings of the HPF User Group 99, Redondo Beach, CA, USA, August 1999.
9) ISO/IEC 9899:1990 Information Technology – Programming Languages – C.

Matthijs van Waveren received the Ph.D. degree in Computational Physics from the University of Amsterdam, the Netherlands, in 1989. He worked at the SARA Computer Centre in Amsterdam as a consultant, and at Schlumberger/Geco-Prakla R&D in Delft and the University of Groningen as a researcher. He joined the Fujitsu European Centre for Information Technology, United Kingdom, in 1996 as a Senior Researcher, where he is currently engaged in the development of software libraries for Fujitsu supercomputers. He is member of the programming language standardization committees ISO WG5, NCITS J3, and NNI NC38122.

Cliff Addison was born in Canada and gained his Ph.D. at the University of Toronto in 1980. He was a post-doctoral fellow at Manchester, United Kingdom, and has worked at the University of Alberta, Canada, CMI Bergen, Norway, and the University of Liverpool, United Kingdom, primarily in the fields of programming environments and parallel numerical algorithms. He joined the Fujitsu European Centre for Information Technology, United Kingdom, in 1996 as a Research Manager, where he has managed several development projects on behalf of Fujitsu, including the HPF Library project.

Peter Harrison received a B.Sc. in Applied Physics from Hull University, United Kingdom, in 1994. He then received an M.Sc. (with distinction) in Advanced Scientific Computation from the University of Liverpool, United Kingdom, in 1996. He joined the Fujitsu European Centre for Information Technology, United Kingdom, in 1997 as a Researcher, and was involved in the design and production of numerical software libraries for Fujitsu supercomputers. He has recently taken up the role of System Administrator at the same company.

Dave Orange received the Ph.D. degree in Computer Science from the University of Liverpool, United Kingdom, in 1993. He worked at the University of Liverpool as a lecturer until 1995. In this year he joined NASoftware Ltd., Liverpool, where he has the position of Software Engineer, and is engaged in compiler design and network operating systems.

Norman Brown received the Ph.D. degree in Computational Mathematics from the University of Liverpool, United Kingdom, in 1980. He worked at Liverpool Polytechnic as a lecturer, at LDRA as a director, and at the University of Liverpool as a deputy director of the Centre of Mathematics Software Research. He joined NASoftware Ltd., Liverpool, in 1992, where he has the position of Managing Director.