HPF with Parallel I/O Extensions [*]

Rajesh Bordawekar Alok Choudhary
Dept. of Electrical and Computer Engineering & NPAC,
3-228 CST, Syracuse Univ., Syracuse, NY 13244

SCCS Technical Report

[*]This work was sponsored by ARPA under contract # DABT63-91-C-0028 and by an NSF Young Investigator Award CCR-9357840 (Alok Choudhary) with matching support from Intel SSD. The content of the information does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

1 Introduction

High Performance Fortran (HPF) has been designed to be an informal standard programming language for a variety of high-performance computers, such as vector machines and massively parallel MIMD and SIMD multiprocessors [10, 8]. HPF uses data distribution and alignment for mapping and decomposing the computational domain on processors. HPF provides data distribution directives which can be used by the user to effectively declare and map distributed arrays. Important directives include PROCESSORS, TEMPLATE, DISTRIBUTE and ALIGN. Arrays can be distributed in either in BLOCK or CYCLIC form in each dimension.

A large number of scientific problems including many grand challenge problems are I/O intensive [1]. Therefore, in order to achieve good scalability in speed and problem size, support for high performance I/O to perform reads/writes and out-of-core computations is necessary. Currently, HPF does not provide any support for explicit parallel I/O, although some proposals were made [10, 2].

This paper proposes some directives for parallel I/O that can be used in conjunction with other HPF directives. These directives have proven to be useful in the initial implementation of runtime and compiler support for parallel I/O in our HPF compiler [9].

2 Parallel I/O Directives

Currently, we are addressing two parallel I/O problems; 1) parallel reads/writes from files, and 2) support for out-of-core computations. A brief description of the directives to support these problems is as follows.

• **DISKS**: This directive is used for describing the logical mapping of disks over which one or more files may be distributed and/or which are used to distribute scratch files for out-of-core computations. The syntax for this directive is similar to the **PROCESSORS** directive in HPF.

For example,

DISKS D(8,8)

indicates that disks are logically arranged as a two-dimensional logical grid of size 8×8. This directive aids a compiler associate a disk (or a set of disks) with processors for file distributions and out-of-core computations. Many processors are allowed to be associated with one disk and many disks are allowed to be associated with one processor. For example, if processor grid size id 16×16, each disk can be associated to maintain scratch files of a 2×2 processor sub-array.

• **FILEPROC**: This directive is also similar to the **PROCESSORS** directive in HPF except that it specifies the processors which really participate in performing I/O. From our earlier studies [4, 7], we observed that the best performance need not necessarily be obtained when all processors performing computations also perform I/O. Thus, this provides the user the flexibility to specify a set of processors to perform I/O. This directive is optional, and if not specified, the default is the number of processors specified in the **PROCESSORS** directives.

For example,

FILEPROC FP(2,2)

specifies that a 2×2 array of processors participates in I/O.

• **FILEDISTR**: This directive declares a file-template and distributes it over the specified number of disks declared in the **DISK** directive. It also uses the optional FILEPROC parameter. This directive uses names declared in **DISKS** and **FILEPRC** as pointers to the corresponding topologies. For example,

FILEDISTR F(D,[FP])

declares a file-template F which is distributed over D disks, and it associates this template with the processors declared in FP. Thus a file distributed over a set of disks can be associated with different sets of processors by using this directive. For example, when declared together,

FILEDISTR F(D, FP1)
FILEDISTR F(D, FP2)

permit two different processor configurations to access files on the same set of disks.

• **FILEALIGN**: This directive is similar to the **ALIGN** directive of HPF. **FILEALIGN** aligns the list of associated files to the template declared using **FILEDISTR** directive. However, there is a fundamental difference between **ALIGN** and **FILEALIGN**. File may not have a size at declaration time. Thus the same file may be aligned to more than one file-templates as illustrated above. This is quite logical since a file can be opened by two different processor grids. Following example illustrates the **FILEALIGN** directive.

FILEALIGN F :: F1, F2, F3

• **ASSOCIATE**: This directive describes the relationships between an array's and the corresponding file's mapping. That is, the **ASSOCIATE** directive `associates` a file-template with the corresponding array template. **ASSOCIATE** directive has the following form

ASSOCIATE :: (file-template, array-template)

For example,

ASSOCIATE :: (F,A),(,),...

associates the file-template F with the array-template A.

Thus, this directive provides an HPF compiler a list of files to be used for I/O for a set of arrays aligned to the corresponding array template.

• **OUT_OF_CORE**: This directive declares an array as an out-of-core array. Following example declares array A as an out-of-core array.

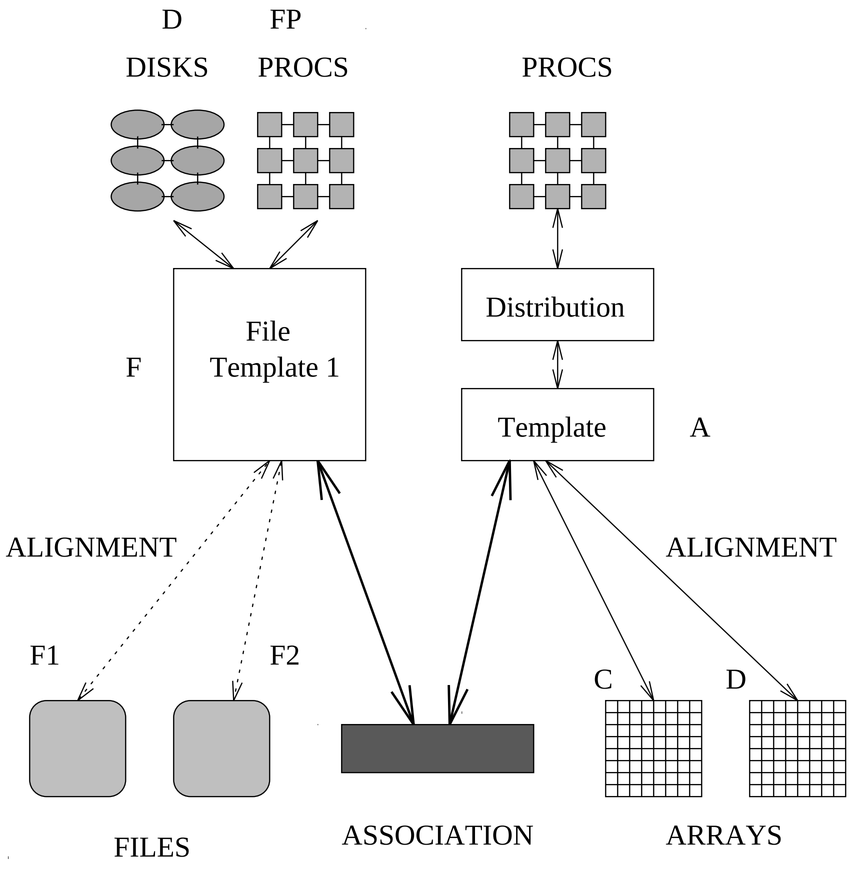

Figure 1: File Distribution and Association in HPF

OUT_OF_CORE :: C

Figure 1 illustrates the relationships between these directives. File-template declares an abstract template (F), which is distributed over (logical) disks (D) and processors (FP). Note that FP specifies a set of processors which may be a subset of processors declared in the processor directive or a may be a different set (e.g., I/O nodes). Implementation, therefore, may vary from one architecture to another. Files to be accessed in the HPF program are aligned to this template F. Array template, to which arrays C and D are aligned, is distributed over a set of processors (PROCS). As illustrated, the array template A and file template F are associated using the associate directive. Thus a set of files mapped to the a file-template will have a set of arrays (may have different sizes) associated with them. As a result, a compiler can optimize the parallel accesses (e.g., read/write) of distributed arrays from/to the associated files using various strategies (e.g., [7]).

3 Compiler and Runtime Support for Parallel I/O

Information provided by the compiler directives is used to extract parameters about array and file distributions, which in turn are used in the runtime primitives. In the following we briefly discuss the primitives and how they are used by the compiler.

Input and output operations include reading/writing arrays from/to files. The cost of reading/writing files in parallel may vary tremendously as a function of the distribution of data on compute nodes [4, 7].

We have developed a set of runtime primitives to perform parallel read and write operations [3, 5]. These primitives provide consistent (and high) performance independent of the type of distribution [5]. The parallel I/O primitives include parallel read (pread), parallel write operation (pwrite) and several supporting primitives including popen, pclose, array_map and proc_map.

We have developed compiler support in our HPF compiler [9] to automatically embed the runtime primitives in the compiler code (F77+MP+I/O) using the directives and constructs specified in the source HPF program. Figure 2 illustrates a set of directives and HPF code fragment as well as the corresponding compiled code in F77+MP+I/O. We only show the pertinent code for the sake of brevity. Note that in the source HPF code, the user uses very simple constructs called pread and pwrite (figure 2(I)) which are automatically converted into the appropriate calls to the runtime routines. Also, the compiler performs all the necessary transformations [1].

4 Summary

The main goal of this paper was to describe directives which, based on our experience, are useful to perform parallel I/O from HPF programs. We also described how these directives are used to embed parallel I/O runtime primitives in the generated code.

We have also developed runtime primitives for out-of-core computations. These include communication and the corresponding optimizations for out-of-core data as well as read and write routines to perform accesses to scratch files. One of the important features of these routines in the they use access pattern information (to be provided by the compiler) to enhance the I/O performance.

Acknowledgments

We would like to thank Ken Kennedy for keeping the parallel I/O problem in the limelight. The OUT-OF-CORE directive has been borrowed from his terminology. We also thank Chuck Koelbel for various helpful discussions. Finally, we would like to acknowledge the contributions of Marc Snir (and his group) and the Vienna Fortran Group (P. Mehrotra, H. Zima and B. Chapman) for parallel I/O in the first round of the HPF definition effort.

[1] If accepted, in the full paper we will provide details of the transformations and performance results on some codes

(I) HPF Program

```
real A(64,64),B(64,64),C(64,64)
CHPF$ processors p(1,4)
CHPF$ template R(64,64)
CHPF$ distribute R(block,block)
CHPF$ align (I,J) with R(I,J) :: A,B
CHPF$ disks d(8,8)
CHPF$ fileproc H(1,4)
CHPF$ filedistr F(H,d)
CHPF$ filealign F :: F1,F2
CHPF$ associate :: F,R
CHPF$ out_of_core :: A
call popen(3,'F1',SEQUENTIAL,UNFORMATTED,OLD)
call pread (A,3)
call pwrite(A,3)
```

(II) F77+MP+I/0 Program

```
REAL A (64, 16), B (64, 16), C (64, 64)
INTEGER array_map,map_info(1536)
INTEGER size_info(7),proc_info(7),distr_info(7),block_size(7)
COMMON /INFO/F_INFO, P_INFO, A_INFO
```

[Initialize the data structures]

```
CALL popen (3,'F1', 0, 0, 1, disks, procs)
TT3temp = array_map (A, size_info, distr_info, block_size, proc_info, proclist)
CALL pread (A, 64, 16, TT3temp, 3, TT2temp, 1024, 1024)
CALL pwrite (A, 64, 16, TT3temp, 3, TT4temp, 1024, 1024)
```

Figure 2: (I) shows a sample HPF^{io} program fragment which uses the proposed I/O directives. Disks are logically arranged as an 8×8 logical grid (D) and I/O processors are arranged as 1×4 grid (H). File-template F is distributed over H and D. Files F1 and F2 are aligned to the file-template F. Finally, file-template F is associated with array template A. Array A is declared out_of_core. HPF compiler automatically generates F77+MP+IO code (Shown in (II)). Various parameters in the routines are used for buffers and other information and will be explained in detail in the full paper.

References

[1] Juan Miguel del Rosario, and Alok Choudhary. High Performance I/O for Parallel Computers: Problems and Prospects. To appear in IEEE Computer.

[2] S. Benkner, B. Chapman, and H. Zima. Vienna Fortran 90. *Scalable High Performance Computing Conference*, April 1992.

[3] Rajesh Bordawekar. Issues in Software Support for Parallel I/O. Master's Thesis, ECE. Dept., Syracuse University, May 1993.

[4] Rajesh Bordawekar, Juan Miguel del Rosario, and Alok Choudhary. An Experimental Performance Evaluation of Touchstone Delta Concurrent File System. *ICS'93*, pages 367-377, July 1993.

[5] Rajesh Bordawekar, Juan Miguel del Rosario, and Alok Choudhary. Design and Evaluation of Primitives for Parallel I/O. To be presented in *Supercomputing'93*, November 1993.

[6] Rajesh Bordawekar and Alok Choudhary. Compiler and Language Support for Parallel I/O. To be presented in *Fourth Workshop on Compilers for Parallel Computers*, Delft, The Netherlands, December 1993.

[7] Juan Miguel del Rosario, Rajesh Bordawekar, and Alok Choudhary. Improved parallel I/O via a two-phase run-time access strategy. *The 1993 IPPS workshop on Input/Output in Parallel Computer Systems*, April 1993.

[8] Geoffrey Fox, Seema Hiranandani, Ken Kennedy, C. Koelbel, Uli Kremer, and Chau-Wen Tseng. Fortran D Language Specification. Technical Report Rice COMP TR90-141. Rice University, December 1990.

[9] Zeki Bozkus, Alok Choudhary, Geoffrey Fox, Tomasz Haupt, and Sanjay Ranka. Fortran 90D/HPF Compiler for Distributed Memory MIMD Computers: Design, Implementation, and Performance Results. *Supercomputing'93* (to appear), November 1993.

[10] High Performance Fortran Forum. High Performance Fortran Language Specification Version 1.0. Technical Report CRPC-TR92225. CRPC, Rice University, January 1993.