NAS Experiences of Porting CM Fortran Codes to HPF on IBM SP2 and SGI Power Challenge

Subhash Saini
NASA Ames Research Center, Moffett Field, CA 94035

Abstract

Current Connection Machine (CM) Fortran codes developed for the CM-2 and the CM-5 represent an important class of parallel applications. Several users have employed CM Fortran codes in production mode on the CM-2 and the CM-5 for the last five to six years, constituting a heavy investment in terms of cost and time. With Thinking Machines Corporation’s decision to withdraw from the hardware business and with the decommissioning of many CM-2 and CM-5 machines, the best way to protect the substantial investment in CM Fortran codes is to port the codes to High Performance Fortran (HPF) on highly parallel systems. HPF is very similar to CM Fortran and thus represents a natural transition. Conversion issues involved in porting CM Fortran codes on the CM-5 to HPF are presented. Several CM Fortran codes have been ported to Subset HPF on the IBM SP2 and the SGI Power Challenge. Speedup ratios versus number of processors for the Linear solver and DSMC code are presented.

1: Introduction

The Numerical Aerodynamic Simulation (NAS) Program, located at NASA Ames Research Center, is a pathfinder in high performance computing for NASA and is dedicated to advancing the science of computational aerodynamics. One key goal of the NAS organization is to demonstrate by the year 2000 an operational computing system capable of simulating an entire aerospace vehicle system in one to several hours. It is currently projected that the solution of this Grand Challenge problem will require a computer system that can perform scientific computations at a sustained rate approximately 1000 times faster than 1990 generation supercomputers. Most likely such a computer system will employ hundreds or even thousands of powerful RISC processors operating in parallel.

High performance Fortran (HPF) [1-4] addresses the need of a large number of parallel computers users. NAS has a large base of CM-5 users and a significant number of production codes with good performance being run on it.

2: Taxonomy of Programming Models

Currently, the following programming models are in common use on various highly parallel systems.

- Message passing is the most common programming model available on highly parallel systems. One of the oldest message-passing libraries is NX on the Intel iPSC/860 [5] and the Intel Paragon [6]. The syntax requires the programmer to explicitly decompose arrays and communicate data. Parallel Virtual Machine (PVM) [7] and Message Passing Interface (MPI) [8] are considered two de facto standards. MPI is expected to become a standard. However, native MPI is yet not supported on the IBM SP2 [27] or the CRAY T3D.

- In data parallel model, the same operation is performed on many data elements by many processors simultaneously. Global data structures are distributed uniformly and processing power follows data. Conceptually, each processor computes only one element of each dataset. Data parallelism is a simpler programming paradigm that scales naturally as data grows. Some commonly used data parallel languages are as follows:

  - CM Fortran [9] is a proprietary language of Thinking Machines Corporation. It is implemented as a data parallel language with Fortran 90 [10] arrays as its parallel data structures. All Fortran 77 operations and intrinsic functions are extended to act point-wise on arrays. New intrinsic functions are provided for constructing and transforming arrays. In addition, CM Fortran also has a FORALL statement and a FORALL construct. The application programmer sees only a single thread of control and views data globally. Distribution of data and computation, communication, and synchronization are managed automatically. The programmer can modify data distribution or invoke special communication patterns. The programmer has to be very careful about the pattern of interprocessor commu-
cation and to limit it to the near-neighbor, if possible. The interprocessor communication is typically coded using intrinsic functions such as CSHIFT and EOSHIFT. In many CM Fortran codes, data parallel syntax can be difficult to decipher, especially when the code makes extensive use of CSHIFT and EOSHIFT. Array elements not explicitly distributed are mapped according to an implementation-dependent default distribution called the canonical distribution.

HPF, the high-level language for parallel programming is based on a number of existing languages such as CM Fortran [9], Fortran 90 [10], Fortran D [11], and Vienna Fortran [12]. HPF allows users to write a single parallel program that can execute on both distributed-memory and shared-memory machines. HPF eliminates the complex, error-prone task of explicitly telling the compiler how, where, and when to pass messages between processors on distributed-memory machines, or when to synchronize processors on shared-memory machines. HPF codes can execute a single program on a distributed-memory cluster, a single Symmetric Multiprocessing Processor (SMP), or a cluster of SMPs.

HPF uses compiler directives PROCESSORS, ALIGN, LAYOUT, and TEMPLATE to map array elements to abstract processors in a two-level mapping: (i) The array elements are first aligned with an abstract index space called a template [22]; (ii) The template is then distributed onto a rectilinear arrangement of abstract processors.

Cray Research Adaptive Fortran (CRAFT) [13-15] derives its features from Fortran 77, Fortran 90, Fortran D, and HPF. CRAFT has the following three directives: (a) Data Sharing Directives; (b) Work Sharing Functions and Directives; (c) Synchronization Functions and Directives. CRAFT emphasizes on its data distribution mechanisms whereas CRAFT emphasizes on work distribution mechanism. The drawback of CRAFT is that codes written using CRAFT-specific directives are not portable to another machine.

4: Porting CM Fortran Codes to HPF

In this section, we discuss the various conversion issues involved in porting CM Fortran codes to HPF [21]. CM-5’s Run Time System (RTS) distributes array elements among the processors available to the program. The particular distribution of an array is called its layout [23]. The layout chosen by the RTS without intervention from the user is the canonical layout or canonical distribution. The main reason to use layout directives is to improve performance over what a user cannot obtain using the canonical layout. The CM Fortran data distribution directives are much simpler than ones available in HPF. For the CM Fortran on the CM-5, there are two layout directives: LAYOUT and ALIGN. For CM Fortran running on the CM-2, in addition to these two there is a third directive called COMMON

In CM Fortran, an arbitrary number of arrays of an arbitrary number of dimensions can be specified in the same CMFS LAYOUT directive as shown in Figure 1. Note "weight" 2 in 2:NEWS in layout directive. There is no concept of weight in HPF. HPF does not allow specifying more than one array in the same line of CHPFS DISTRIBUTE due to the stringent attributed form of the distribution directive as shown in Figure 2. In summary, SERIAL translates into "s" and NEWS translates into BLOCK.

3: Characteristics of CM Fortran Codes

A quick look at any CM Fortran code reveals that it is comprised mostly of the following statements, structures, and directives: (a) Array syntax, which is a part of Fortran 90 language; (b) WHERE statement and WHERE construct, which are parts of Fortran 90 language; (c) Intrinsic shift functions – CSHIFT and EOSHIFT which are parts of Fortran 90 language; (d) FORALL statement and FORALL construct, which are parts of HPF specification; (e) CMFS LAYOUT directives for data distribution-like HPF compiler directives; (f) CMFS ALIGN directive for data alignment - like HPF directive ALIGN; (g) CM Fortran utility routines with equivalent routines in HPF Library; (h) Parallel I/O that have no counterpart in HPF, as there is no parallel I/O in HPF. It is clear that, except for parallel I/O, porting CM Fortran codes to HPF is a natural transition [21].

Example: 1 (a)

```
INTEGER, PARAMETER :: N=256
REAL*8, ARRAY(n,n,n) :: a
CMFS LAYOUT a(:,SERIAL,2:NEWS,:) :: x
```

Example 2 (a)

```
INTEGER, PARAMETER :: m=128
REAL*8, ARRAY(m,m) :: x
REAL*8, ARRAY(m) :: y
CMFS LAYOUT(:,SERIAL,2:NEWS), y(:,NEWS)
```

Figure 1: Use of layout directive in CM Fortran.

Example: 1 (b)

```
INTEGER, PARAMETER :: N=256
REAL*8, DIMENSION(n,n,n) :: a
CHPF$ DISTRIBUTE a(*,BLOCK,BLOCK)
```

Example 2 (b)

```
INTEGER, PARAMETER :: m=128
REAL*8, DIMENSION(m,m) :: x
REAL*8, DIMENSION(m) :: y
CHPF$ DISTRIBUTE x(*, BLOCK)
CHPF$ DISTRIBUTE y(BLOCK)
```

Figure 2: Use of distribute directive in HPF.

A LAYOUT directive can align a vector only with the
first row or column of array. However, an ALIGN directive can match the vector with any row or column of array. In CM Fortran, for statically allocated arrays: (a) conformable arrays with canonical layouts are aligned and (b) conformable arrays with identical noncanonical layouts are aligned. The ALIGN directive can enhance the performance of an application by eliminating unnecessary data communication when operations are performed on multiple arrays that would not normally be aligned. The use of ALIGN directive in CM Fortran and in HPF is shown in Figures 3 and 4, respectively. Many CM Fortran codes rely on the canonical alignment and do not use any ALIGN directives. However, such codes, when ported to HPF must use the ALIGN directive for better performance as there is no canonical alignment in the current implementation of HPF compilers.

In HPF, a template is an abstract space of indexed positions [22] - an “array of nothings” as opposed to, say, an “array of reals.” One can use a template as an abstract align-target that can then be distributed. There is no concept of TEMPLATE in CM Fortran. In some HPF compilers (e.g., in pg/hpf), if arrays are not aligned to the same template, the compiler does not recognize efficient communication patterns. However, when arrays are aligned to a common template the compiler generates efficient code. An example illustrating the use of template is shown in Figure 8.

CM Fortran utility routines provide convenient access from CM Fortran to the capabilities of lower-level CM software [24]. The purpose of this was to provide a functionality and performance beyond what was available from the compiler. With maturing of the compiler most of these functionalities were included in the compiler. Nevertheless, many CM Fortran codes even today rely significantly on these utility routines. Most of these routines have equivalent functionality in HPF routines.

Many CM Fortran codes use subroutines from CM Fortran Utility Library to perform prefix operations, or scans on one axis of an array using CALL CMF_SCAN_com-

```
INTEGER, PARAMETER :: N=256
REAL*8, ARRAY(n,n) :: a
REAL*8, ARRAY(n) :: b
CMFS ALIGN b(j) WITH a(1,j)
```

Figure 3: Use of ALIGN directive in CM Fortran.

```
INTEGER, PARAMETER :: N=256
REAL*8, DIMENSION(n,n) :: a
REAL*8, DIMENSION(n) :: b
CHPF$ ALIGN b(j) WITH a(1,j)
```

Figure 4: Use of ALIGN directive in HPF.

In HPF, ALIGN directive can enhance the performance of an application by eliminating unnecessary data communication when operations are performed on multiple arrays that would not normally be aligned. The use of ALIGN directive in CM Fortran and in HPF is shown in Figures 3 and 4, respectively. Many CM Fortran codes rely on the canonical alignment and do not use any ALIGN directives. However, such codes, when ported to HPF must use the ALIGN directive for better performance as there is no canonical alignment in the current implementation of HPF compilers.

On CM-5, the usual Fortran OPEN, WRITE, and READ statements do very fast parallel I/O for large parallel arrays using the Scalable Disk Array (SDA). There is no provision for parallel I/O in HPF. Parallelism in I/O is necessary for performance of I/O dominated codes. The I/O dominating codes such as out-of-core solvers written in CM Fortran will not perform well in HPF unless parallel I/O is included in HPF.

Many CM Fortran codes use the FORALL construct [26]. Currently, Subset HPF compilers don’t provide this construct. Therefore the FORALL construct in CM Fortran codes needs to be converted to FORALL statements. Current Subset HPF compilers have only FORALL statement.

In CM Fortran, by default, for every CM fortran array the CM Run-Time System (CMRTS) uses a canonical data distribution. It is not necessary to use layout directives for CM Fortran arrays. The CMRTS uses block layout of elements as opposed to cyclical layout. In HPF, compilers don’t distribute arrays by default and, in fact, they are replicated. For porting CMF codes to HPF, it is necessary to insert HPF mapping directives to distribute the data. Code of Figure 7 relies on canonical distribution feature of CM Fortran and therefore does not use layout directive. HPF version of the code is shown in Figure 8 and uses TEMPLATE, ALIGN, and DISTRIBUTE directives of HPF. It is recommended to vendors to include canonical alignment and canonical distribution as default in HPF compilers.

Most of production codes running on CM-2 or CM-2 use routines from the CMSSL [25]. Commonly used CMSSL routines on CM-5 at NAS are FFTs, block and scalar pentadiagonal solvers, and block tridiagonal solvers. None of these routines is available in HPF on any machine. At NAS, an effort is underway to develop these routines in HPF [39]. It looks as though the big obstacle in porting CM Fortran codes of CM-5 to HPF is the lack of CMSSL equivalent routines in HPF. The 3-D FFT (FT) CM Fortran code from the NAS Parallel Benchmarks could not be ported to HPF because of the absence of parallel 3-D FFT that can be used from HPF. This obstacle must be overcome soon to facilitate the easy porting of NAS CM Fortran codes to HPF.
5: Programming Tools

A tool called PRISM is available for developing and optimizing CM Fortran codes [28]. The PRISM programming environment is integrated and graphical within which users can perform editing, debugging, data visualization, and performance analysis. It provides an easy-to-use, flexible, and comprehensive set of tools for performing all aspects of CM-5 programming for CM Fortran programs. No such tool is available for HPF programs. At this point, we discuss the functionality of PRISM and give a work around for HPF programs until a tool is available for HPF programs.

One can edit and compile source code by invoking the appropriate utilities from PRISM. Currently, no such feature is available in any HPF compiler. The D editor developed by Ken Kennedy’s group at Rice University may be used as a starting point to develop the one for HPF [37].

Currently, no HPF debugger is available. However, for the pghpf compiler [29-30] the user can debug the Single Program Multiple Data (SPMD) Fortran 77 program on a node using multiple X windows. Although very inconvenient, this work around for debugging is very useful to obtain a trace back of a program that is crashing unexpectedly. This type of debugging can be done either on a processor or on multiple processors. The advantage of debugging HPF code on a single processor is that one can use standard debugger on the executable. We recommend that vendors provide the HPF debugger as soon as possible. This debugger need not be very sophisticated in the beginning.

Currently, no profiling tool is available for HPF codes. However, PGI has developed a graphical HPF profiling tool that does subroutine-level and line-level profiling of HPF codes. This tool is undergoing beta testing and may be available soon on some machines. At this time, profiling of HPF codes can be done manually by inserting a Fortran 90 intrinsic function as one example, COM Fortran / HPF:

\[ y = \text{CSHIFT}(x, \text{DIM}=1, \text{SHIFT}=-1) \]

**Figure 5:** Use of CSHIFT and EOSHIFT in CM Fortran.

\[ b = \text{EOSHIFT}(a, \text{DIM}=1, \text{SHIFT}=-1, \text{BOUNDARY}=[1:4]) \]

**Figure 6:** Use of CSHIFT and EOSHIFT in HPF.

6: HPF Compilers

Subset HPF [22] is intended by HPF Forum to include a minimum set of features from Fortran 90 and HPF to encourage and facilitate an early release of production quality compilers. Vendors may include more features than listed in Subset HPF. Although not required by Subset HPF, PGI [29-30] has included KIND, Namelist I/O, etc. in their pghpf compiler. Currently, two Subset HPF compilers one from PG [29] and other from Applied Parallel Research (APR) [32] are available on NAS IBM SP2 and the HPCCP workstation cluster. The pghpf compiler has limited CM Fortran compatibility as follows:

- It supports both the CM Fortran and HPF names. For example, CM Fortran / HPF: DSHAPE/SHAPE, DOT-PRODUCT/DOT_PRODUCT,DLBOUND/LBOUND, DUBOUND/UBOUND.

- It supports both the CM Fortran and HPF versions of array constructors. It accepts the CM Fortran method of square brackets in the definition of array constructors, i.e., a = \([1,2,3,4,5,6,7,8]\). The corresponding syntax in HPF is a = \((1,2,3,4,5,6,7,8)\).

- It supports CM Fortran ARRAY keyword in place of the Fortran 90 standard DIMENSION keyword as in:

  **CM Fortran:** INTEGER, ARRAY 
  **HPF:** INTEGER, DIMENSION

The three CM Fortran intrinsic functions CSHIFT, EOSHIFT and RESHAPE have names identical to corresponding functions in Fortran 90, but different calling sequences. Porting of these functions from CMF to HPF is shown in Figures 5 and 6. In HPF, boundary argument in EOSHIFT is not yet implemented.

The following intrinsics of CM Fortran: DIAGONAL, FIRSTLOC, LASTLOC, PROJECT, RANK and REPLICATE have no equivalent in Fortran 90 or HPF.

The DEC Fortran 90 V1.3 compiler the first to support the HPF compiler.
both the ANSI/ISO Fortran 90 standards and the full de-facto HPF specification as claimed by DEC in a recent press release [33]. It includes program development tools like a debugger and profiler, as well as underpinnings such as optimized message passing and parallel system management tools. However, both are only available for DEC Alpha platforms running Digital OSF-1 and not on any other machines [33].

7: Results

In this section, we will discuss the porting of CM Fortran codes from the CM-5 to HPF.

7.1: Parallel Matrix Multiplication

The algorithm we have implemented is based on Cannon algorithm [34]. This algorithm was earlier implemented by the author on CM-5 using CM Fortran and on CRAY T3D using the CRAFT model [35]. It is implemented using only the Fortran 90 intrinsic function CSHIFT. CM Fortran implementation of this algorithm is shown in Figure 7, whereas HPF implementation is shown in Figure 8. In CM Fortran version, no layout directives are used to distribute the data as default data distribution is canonical distribution. However, in HPF version three directives (TEMPLATE, DISTRIBUTE, and ALIGN) are used to distribute and align the data. Performance and scalability of the HPF version of Cannon algorithm is compared with other parallel implementations and will be published separately [36].

```fortran
PROGRAM nasmult
PARAMETER (n = 1024)
DOUBLE PRECISION, DIMENSION(n,n)::x,y,z,x1,y1
INTEGER iskew(n)
!HPF$ TEMPLATE T(n,n)
!HPF$ DISTRIBUTE T(BLOCK,BLOCK)
!HPF$ ALIGN (:,:) WITH T :: Z
!HPF$ ALIGN (:,* ) WITH T(:,*) :: x
!HPF$ ALIGN (*,:) WITH T(*,:) :: y
x = 2.0
y = 3.0
z = 0.0
FORALL (i =1:n) iskew(i) = i - 1
x1 = CSHIFT( x, 2, iskew )
y1 = CSHIFT( y, 1, iskew )
z = 0.0d0
DO ( n ) TIMES
  z = z + x1 * y1
  x1 = CSHIFT( x1, 2, 1 )
y1 = CSHIFT( y1, 1, 1 )
ENDDO
PRINT *, z(1, 1), Z(n, n)
z = 0.0d0
z = MATMUL(x, y)
PRINT *, z(1, 1), z(n, n)
END
```

Figure: 7 Parallel matrix multiplication in CM Fortran using Cannon algorithm.

7.2: Gaussian Elimination

The other code ported from CM Fortran to HPF solves systems of linear equations, \( Ax = b \), using Gaussian elimination followed by backward substitution. Fortran 90 functions used in this program are MAXLOC, SYSTEM_CLOCK, SELECTED_REAL_KIND, and RANDOM_NUMBER. HPF functions or directives used are PROCESSORS, NUMBER_OF_PROCESSORS(), TEMPLATE, ALIGN, and DISTRIBUTE. We also used FORALL statement. We used various distribution schemes such (block,block), (cyclic,cyclic), and (block,*) to investigate the communication behavior and performance results will be published separately [36]. Speedup of this linear solver for \( A = 1000 \times 1000 \) with blocked distribution as a function of processors for the IBM SP2 and SGI Power Challenge, and is given in Figure 9. Note that on both the IBM SP2 and SGI Power Challenge, performance first increases and then decreases. Results of these figures were obtained using the pgHPF compiler.

7.3: Laplace Equation Solver

We solve Laplace’s equation on a unit square by point Jacobi relaxation method, with the boundary condition that \( f = 1 \) on \( y = 1 \) and \( 2 \) elsewhere on the boundary, and the initial condition that \( f = 0 \) in the interior. The port from CM
Fortran to HPF was straightforward. Various distribution schemes were tried and performance results (as a function of number of processors and different grid sizes) will be published separately [36].

7.4: Convolution

CM Fortran code performs a one-dimensional convolution over the first dimension of a two-dimensional dataset. This code tests the performance of the Fortran intrinsic function EOSHIFT. The port of this code to HPF was straightforward. Once again, various distribution schemes were tried and performance results will be published elsewhere [36].

7.5: Direct Simulation Monte Carlo

Particle simulation using direct simulation Monte Carlo (DSMC) is a technique for analyzing low density flows and is routinely used for performing engineering analysis of aerospace vehicles. In the CM Fortran implementation of the code, there is fine-grain parallelism associated with the particles in the simulation and coarse-grain parallelism associated with the cells used in the simulation [38]. NAS, in collaboration with Portland Group has ported this CM Fortran code to HPF. Speedup of HPF code as a function of number of processors for the SGI Power Challenge is shown in Figure 10. Up to 12 processors, the code scales well. Beyond 12 processors, performance is almost flat. This is due to inadequate bus bandwidth of 1.2 GB/s for the SGI Power Challenge.

8: Conclusions

a. In most cases porting of NAS CM Fortran codes to HPF on the IBM SP2 and SGI Power Challenge machines proved to be straightforward. Much of the effort was devoted to translating CM Fortran layout and ALIGN directives to equivalent HPF distribute and align directives. Some time was also spent in replacing CM Fortran Utility routines with HPF equivalent functions.

b. It is recommended to vendors that they make canonical data distribution and canonical align as defaults in their HPF compilers.

c. Inclusion of FORALL construct will facilitate the porting of large numbers of CM Fortran codes to HPF with ease. Basic profiler and debugger tools should also be provided for HPF programs.

d. One of strong feature of CM Fortran is the ease with which one can perform parallel I/O. It is recommended to the HPF Forum to include parallel I/O consistent with CM Fortran. The capability of a communication compiler should also be considered for inclusion in the next HPF revision.

e. CM Fortran codes relying on CMSSL library could not be ported to HPF and this led to a lot of frustration. One of the biggest hurdle the NAS users are facing in porting CM Fortran codes to HPF on the IBM SP2 is lack of parallel mathematical and scientific libraries, which can be used from HPF codes. It has been argued in some quarters that the use of CMSSL library points to the poor quality of the CM Fortran compiler. On the CM-2 and CM-5, a CM Fortran Utility Library was available to work around for CMF compiler deficiencies. Eventually, most routines (including routines for parallel I/O) of this utility library became part of the matured CM Fortran compiler. Just as BLAS 2, BLAS 3, LIBSCI, and LAPACK are essential for successful parallel-vector programming, parallel scientific libraries will also be needed for the success of HPF on highly parallel computers. An immediate need for NAS HPF users is parallel Fast Fourier Transform, block tridiagonal, and block pentadigonal solvers. We recommend to vendors that CMSSL is a logic library, to emulate based on the fact that CMSSL is an industry leader and the production CM Fortran codes far exceeds that of any other vendor. One of most powerful feature, of CMSSL is “multiple instances.” This feature permit users to perform multiple independent operations on different datasets concurrently.

f. It is our belief that vendors should pay special attention for inclusion of an efficient data parallel programming model as defined by the HPF Forum as a paradigm characterized by single-thread execution, global name space, and loose synchronously. Although NAS recognizes that HPF may not be a ultimate solution for perfect parallel programming, it seems to offer the best hope for maintaining a single portable version of codes on all computing platforms. The success of HPF hinges on the development of compilers that can provide reasonable performance satisfactory to users. The first and foremost task for HPF compiler developers should be to develop a HPF compiler that will generate code equal to or better than CM Fortran with consistently high performance across all computing platforms.

The success of HPF hinges on the development of compilers that can provide reasonable performance satisfactory to users. The first and foremost task for HPF compiler developers should be to develop a HPF compiler that will generate code equal to or better than CM Fortran with consistently high performance across all computing platforms.

As the vendors move closer to that goal, they should consider taking a more active role in the development of programming tools, as well.

h. Computations such as FFT, banded tridiagonal solvers, finite element, n-body problems possess regular but specific data access patterns that HPF compilers need to recognize and support.
Figure 9: Speedup versus number of processors for solving $Ax = b$ by Gaussian elimination.

Figure 10: Speedup versus number of processors for DSMC code.
References

[38] D. H. Bailey, S. Saini, and R. S. Schreiber, et al. - A group has been formed to identify and develop scientific libraries needed by NAS users in HPF codes.