F^{--}: A Parallel Extension to Cray Fortran

ROBERT W. NUMRICH

Cray Research, 655 Lone Oak Drive, Eagan, MN 55121, USA; e-mail: rwn@cray.com

ABSTRACT

F^{--} is a parallel extension to Cray Fortran 77 for distributed memory computers. It adds exactly one new symbol to the language, a vertical line that separates two sets of indices. The first set contains the coordinates for data in a local data grid. The second set contains the coordinates for processors in a global processor grid. A statement such as $x(i, j) = y(i, j|p, q)$ generates a load from remote address $y(i, j)$ in the data grid on processor (p, q) followed by a store to local address $x(i, j)$ in the local data grid. F^{--} syntax requires an explicit statement of the relationship between data layout and processor layout. It assumes that good performance on a distributed memory computer requires the programmer to understand and to exploit data locality. Programmers use the F^{--} syntax only when it is needed. Otherwise all data are local and all code is local. Compiler and library developers concentrate on generating well-optimized local code.

1 INTRODUCTION

F^{--} is a parallel syntax added to Cray Fortran 77. Its simple design is based on the assumption that the Fortran 77 language already contains most of what it needs to support parallel programming. Fortran 77 lacks two important features. One is a mechanism for pointing to data in another processor's memory. F^{--} provides a syntax to solve this problem. The other is a mechanism for processor synchronization and control. F^{--} provides library functions to solve this problem.

As its name is meant to imply, the F^{--} philosophy is quite opposite from the philosophy of other parallel extensions to Fortran [1–4] that, for the most part, adopt some variation of a data parallel model. F^{--} is an alternative to the message-passing style of programming but it is not message-passing. No handshaking between processors is necessary such as the send/receive mechanism required by message-passing. It assumes an underlying global address mechanism, supported either by hardware or by software, that allows the programmer to read or write any address in any processor's memory.

In relation to other Fortran dialects, F^{--} lies somewhere between Fortran 77 and Fortran 90. It is something less than Fortran 90 although something more than Fortran 77. The name F^{++} was discarded so as not to imply some flavor of an object-oriented language. On the contrary, it expresses a reluctance to embrace these more complicated programming models.

F^{--} is a performance-oriented extension to Fortran 77. It intentionally sacrifices some of the ease-of-programming features of other extensions to Fortran in favor of performance. Since Cray Fortran 77 already incorporates some of the features of Fortran 90, such as array syntax, F^{--} includes them. It excludes other Fortran 90 features such as passing array sections and the use of intrinsic functions that accept arbitrary data structures. F^{--} considers special operations to be part of a library supporting the language rather than a part of the language itself.

F^{--} supports the single-program-multiple-data (SPMD) style of programming [5]. Each processor runs independently with its own copy of the code with its own data. It also supports the multiple-program-multiple-data (MIMD) style of programming through explicit use of IF

Received March 1995
Revised January 1996

ISSN 1058-9244/97/$8
Scientific Programming, Vol. 6, pp. 275–284 (1997)

statements for program control and explicit calls to synchronization functions for processor control.

The most important advantage of the F^{--} syntax is that it represents processors in a coordinate grid in the same way that it represents data in a coordinate grid. Its syntax is a minimum addition to sequential Fortran that requires the programmer to perform explicit mappings between local and global addresses. The benefit to the programmer is that it exposes the relationship between data and processors and allows the programmer to assume control of the underlying global address space using a familiar syntax. F^{--} coerces local data to global data. All data references are assumed local unless proven remote by the explicit syntax. F^{--} syntax points to objects that are not *here* but *there* in another processor's memory. The syntax is a flag to the programmer as well as to the compiler that remote references are taking place.

2 FORTRAN ADDRESSES

The address of a Fortran matrix element $a(i, j)$ is defined by its dimension statement

```
real a(m,n)
```

and by a linear convention for computing its address in memory

$$\mathrm{loc}\big(a(i, j)\big) = \mathrm{base}(a) + (j - 1) \cdot m + (i - 1) \quad (1)$$

relative to a base address

$$\mathrm{loc}\big(a(1, 1)\big) = \mathrm{base}(a). \quad (2)$$

The combination of the dimension statement and the linear rule Equation (1) completely specifies the address of $a(i, j)$.

The dimension statement and the linear rule are interpreted within a local program environment. Across a subroutine boundary, the Fortran language passes a base address, which may be different from Equation (2), that becomes the new base address in the called subroutine. The dimension statement may also change from one subroutine to another without ambiguity. The compiler does not care if the programmer passes an invalid address or runs off the end of an array. The programmer is responsible for maintaining consistency across subroutine boundaries.

The ability to refer to a matrix element as $a(i, j)$ is a notational convenience that hides the address computation from the programmer. The programmer, however, risks loss in performance by ignoring the fact that the address space has been linearized. On a single processor with interleaved memory, bank conflicts occur if code runs through data with a bad stride. On shared memory multiprocessors, interprocessor memory contention may lower performance, or worse yet race conditions among processors trying to write to the same area of memory may cause unpredictable results. On distributed memory machines, careful attention to locality of data is often critical for good performance. Vectorized blocked algorithms often provide solutions to these problems. The programmer allocates data such that each processor owns a slice or a block and then writes vector code on the first index of the block. F^{--} syntax is a method for explicitly adding support for this programming style to Fortran array syntax.

3 THE F^{--} EXTENSION

F^{--} allows the addition of a second set of indices separated from the usual set by a vertical line

```
real a(data_grid|processor_grid)
```

The first set of indices contains the normal Fortran data coordinates and the second set of indices contains processor coordinates. Any number of dimensions can be specified for processor coordinates according to the same rules that apply to the data coordinates. If no processor grid is specified, the data are local.

To take an example that is not too simple yet not too complicated, consider the dimension statement

```
real a(m,n|p,q)
```

describing a matrix of size $(m \times n)$ on each processor in a $(p \times q)$ grid of processors. F^{--} extends the linear address convention (Equation 1) to the processor grid such that matrix element $a(i, j|r, s)$ resides in the memory of processor

$$\mathrm{pe}(r, s) = \mathrm{base}(\mathrm{pe}) + (s - 1) \cdot p + (r - 1) \quad (3)$$

relative to a base processor

$$\mathrm{pe}(1, 1) = \mathrm{base}(\mathrm{pe}). \quad (4)$$

In most cases, the base processor number is zero, but it need not be.

F^{--} requires an important additional condition: the base address must be the same *virtual* address for all processors

$$\mathrm{loc}\big(a(1, 1)\big) = \mathrm{base}(a) \quad \text{for all pe's.} \quad (5)$$

If the virtual to physical translation is different on different processors, the hardware or the software must be able to resolve the translation so that each processor points to

the correct remote address. An array declaration containing F^{--} syntax must be loaded by the operating system at the same virtual address in each processor. Automatic arrays containing F^{--} syntax, as described in Section 6, imply implicit synchronization so that all processors allocate the same array at the same virtual address.

By default, F^{--} follows the normal Fortran style, numbering the processors starting with one. If the programmer wishes to number the processors starting from zero, the dimension statement becomes

```
real a(m,n|0:p-1,0:q-1)
```

and Equation (3) becomes

$$\mathrm{pe}(r, s) = \mathrm{base}(\mathrm{pe}) + s \cdot p + r. \qquad (6)$$

In general, the programmer may dimension the array as

```
real a(m_1:m_2,n_1:n_2|p_1:p_2,q_1:q_2)
```

F^{--} syntax replicates a local data structure across a set of processors. It has no concept of a global array shared by all processors. The programmer uses F^{--} syntax as needed to coerce local data to distributed data. The processor coordinates following the vertical line are logical processor numbers not necessarily related to the physical processor numbers in the hardware. There is no requirement that the product of processor coordinates in a dimension statement be equal to the number of processors actually running. Just as the compiler normally does no bounds checking for local array indices, it does no bounds checking for processor indices. Generation of an invalid processor number produces unpredictable behavior, most likely a run-time error that terminates the program.

4 DATA TRANSFER BETWEEN PROCESSORS

Transfer of data from one processor to another is a simple example that illustrates the F^{--} programming style.

```
real a(n|npes),b(n|npes)
my_pal = some_rule(me)
do i=1,n
    b(i|my_pal) = a(i)
enddo
```

Notice that the reference to the array $a(i)$ on the right side of the do loop has no F^{--} syntax attached to it even though its dimension statement contains the syntax. Such usage is legal and means that the local address of the array is intended. Since the data are written to another processor's memory, a memory quiet function, similar to the CMR (complete memory reference) instruction on Cray's shared memory machines, must be generated at the end of the do loop to guarantee completion of the transfer. If cache coherence on the remote processor is a problem, it must be handled either by the hardware or by the compiler or by the programmer as it is done now, for example, on the Cray-T3D [6].

Since Cray Fortran 77 includes array syntax, the programmer may choose to replace the do loop in this example with Fortran 90 array syntax

```
b(:|my_pal) = a(:)
```

Array syntax within F^{--} refers to purely local operations and has no connotation of data parallel operations. Any processor executing this statement writes data from its own memory to the memory of the processor whose number is my_pal. If a processor executes the reverse statement

```
a(:) = b(:|my_pal)
```

it reads data from another processor's memory. In either case, the programmer is responsible for any required synchronization between processors that makes reading or writing data safe. Section 9 considers synchronization questions in more detail.

Gather or scatter operations are also possible

```
a(:) = b(index(:)|my_pal)
a(index(:)) = b(i|jndex(:))
```

A broadcast from processor zero to all other processors becomes

```
if(me.eq.0) b(:|:) = a(:)
```

And it is possible to do arithmetic using the syntax

```
a(:) = a(:) + s*x(:|pe)
```

Figure 1 shows transfer velocities measured using F^{--} syntax with the current implementation on the Cray-T3D as described in Section 12. The results are compared with the same measurements obtained using the library function **put**() written in assembler [7]. The asymptotic velocity is the same for both cases. The F^{--} syntax is slightly faster for short lengths because it saves the overhead of the subroutine call to the **put**() function.

A more complicated example, commonly found in computational fluid dynamics, weather and ocean codes that use domain decomposition methods, illustrates a procedure for updating ghost cells [8]. Each plane of physical data has one ghost cell on each of its four sides. For simplicity, this example assumes periodic boundary conditions. The East update is independent of the West update and the North update is independent of the South update. A barrier is required between the East-West and the North-South updates because the East-West transfer

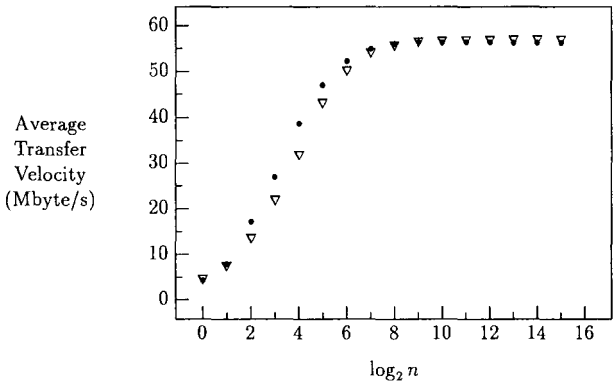

FIGURE 1 Average transfer velocity as function of transfer length on the Cray-T3D with machine frequency $\nu = 150$ MHz. Each processor writes n 64-bit words to its partner with processor number one greater than its own. The upper curve (•) represents measurements obtained with F^{--} syntax. The lower curve (▽) represents measurements obtained with the assembler **put**() function.

includes data in the corners of the ghost cells and must be completed before performing the North-South update.

F^{--} syntax points to a processor's North-South-East-West neighbors in a natural way. If my processor coordinates are (my_row, my_col), then my North neighbor is (my_row − 1, my_col) and my East neighbor is (my_row, my_col + 1). If the compiler is smart enough, it may recognize that each processor may safely generate traffic to two different processors at the same time. Using more wires on the interconnection network at the same time may lead to increased performance. The library functions **get**() and **put**() do not support concurrent communication patterns to more than one processor at a time. Rather than requiring specialized functions for every kind of communication pattern, F^{--} syntax allows the programmer to write arbitrary communication patterns as needed.

```
subroutine update(a,m,n,my_row,my_col,
                  p_row,p_col)
integer p_row,p_col,north,south,east,
        west
real a(0:m+1,0:n+1|0:p_row-1,0:p_col-1)
east = mod(my_col+1,p_col)
west = mod(my_col-1,p_col)
north= mod(my_row-1,p_row)
south= mod(my_row+1,p_row)
call barrier()
a(1:m,  0|my_row,east) = a(1:m,n)
a(1:m,n+1|my_row,west) = a(1:m,1)
call barrier()
a(m+1,0:n+1|north,my_col) = a(1,0:n+1)
a(0,  0:n+1|south,my_col) = a(m,0:n+1)
call barrier()
return
end
```

5 SUBROUTINE INTERFACE

F^{--} follows the same rules as Fortran 77 when it passes variables from one subroutine to another. Variables are passed by address. No global information is passed, only the local address. A call to a subroutine such as

```
call subx(x(1|p))
```

passes the local address of array x and the processor number is ignored. By constraint Equation (5) placed on the base address of an array, the address of the array x is the same on each processor. Processor information, if needed, is passed as a separate argument:

```
call subx(x,p)
```

There is no need to pass global information, for example, in the form of dope vectors, because global information has only local scope within a subroutine. An array dimension or shape may change, with or without vertical line notation, across a subroutine boundary. Indices following the vertical line are interpreted within their local context just as normal indices are interpreted within their local context in Fortran 77. For example, consider passing an array from the main program, where it is used only locally, to a subroutine, where it is used globally:

```
real x(n)
call subx(x,n,npes)

subroutine subx(x,m,np)
real x(m|np)
```

There is no ambiguity in the meaning as long as the array $x(n)$ declared in the main program is allocated space at the same memory location in each processor. At the call to routine **subx**(), the local address of x is passed across the interface where it is interpreted as the local address of x on the other side. Since the array x has F^{--} syntax in its declaration on the other side, it can be used to point to another processor's memory where the processors are numbered from 1 to np as a linear array of processors.

The processor dimension in subroutine **subx**() could just as well be dimensioned with a dummy dimension, for example,

```
real x(m|*)
```

if the programmer just wants to think of processors in a linear grid. The programmer may, however, want to think of the processors in a two-dimensional grid in subroutine **subx**(), for example,

```
subroutine subx(x,m,nx,ny)
real x(m|nx,ny)
```

The incoming address for x still points to the same local address but the programmer has a different picture of the relationship of the processors to each other. There is no ambiguity in switching views in this way.

6 DYNAMIC MEMORY ALLOCATION

Dynamically allocated arrays that will be used for communication between processors must be located at the same virtual address on each processor in order for the F^{--} syntax to work. The programmer must append processor information in the dimension statement to tell the compiler to allocate the new data in a common, shared area.

```
subroutine subx(n,p,q)
real x(n|p,q)
```

If the shape of the processor grid is unimportant, then the declaration

```
subroutine subx(n)
real x(n|*)
```

is sufficient. Synchronization is required at the point of allocation so that all processors arrive at the same point. The programmer is responsible for making sure they all get there and the compiler is responsible for the implicit synchronization.

7 GLOBAL POINTERS

F^{--} syntax effectively defines a global pointer to a data structure that exists on each processor at the same memory location. The existing Cray Fortran 77 pointer can be combined with F^{--} syntax to point to irregular data structures that are located at different addresses or don't exist at all on other processors. For example, let

```
pointer (ptr,remote_x(n|npes))
```

define a data structure that may live on another processor. Suppose each processor maintains a table containing a list of addresses that contain data to be shared with other processors. These addresses may point to variables that the other processors know nothing about. If the table is allocated at the same address in each processor, then any processor can use F^{--} syntax to read addresses from or write addresses to the table. A global pointer can then be established in the following way:

```
integer pe,table(npes|npes)
pointer (ptr,remote_x(n|npes))
pe = some_rule(me)
ptr = table(me|pe)
y(:) = remote_x(:|pe)
```

Combined with dynamic memory allocation, this example allows the programmer to construct arrays of structures where each processor has a different data structure.

A similar programming style is possible in the C language using a syntax appropriate to that language. A name for such a language might be C^{--}. Fry [9] has implemented one method. Carlson and Draper [10] have created AC. Culler et al. [11] have proposed Split-C. Rose and Steele [12] describe C*.

8 LIBRARY SUPPORT

A simple library supports the F^{--} extension. Although some of the members of this library might be implemented as compiler directives, the F^{--} philosophy limits extensions to the basic Fortran language to a minimum set. F^{--} has no compiler directives. Fortran uses functions and subroutines to perform common operations. Some of these functions and subroutines may need to be intrinsic functions for performance reasons.

Two special variables are resolved at either compile-time or load-time: `n$pes` and `log$pes`. If the number of processors equals a power of two they are related by `n$pes` $= 2^{\texttt{log\$pes}}$. If the number of processors is not a power of two, then `log$pes` $= -1$ so that it can be used as a flag to test that the number of processors matches the assumptions of the program.

These two variables are useful for designing scalable programs that work for any size problem and any number of processors equal to a power of two. For example, the following parameter statements divide the processors into a two-dimensional grid (`pe_row` × `pe_col`):

```
parameter(n_pes=2**log$pes)
parameter(half_log_pe=log$pes/2)
parameter(pe_row=
    2**(log$pes-half_log_pe))
parameter(pe_col=2**(half_log_pe))
```

A matrix `a(mmax,nmax)`, for example, divides into blocks corresponding to the processor grid,

```
parameter(mmax=128,nmax=128)
parameter(idim = mmax/pe_row,
          jdim = nmax/pe_col)
real a(idim,jdim|0:pe_row-1,0:pe_col-1)
```

If log$pe = 0, the code runs on a single processor without change. With care the programmer can write code using the F^{--} syntax in such a way that by simply changing vertical lines to commas the code becomes normal Fortran 77, portable to other machines.

At run-time, each processor computes its own position within the processor grid,

```
my_pe = mype()
my_row = and(my_pe,pe_row-1)
my_col =
    shiftr(my_pe,log$pes-half_log_pe)
```

where the grid is zero based in both directions.

A partial list of library functions is the following:

```
mype()
npes()
log_pes()
get()
put()
global_sum()
global_min()
global_max()
global_collect()
atomic_update()
atomic_swap()
broadcast()
```

This library is very similar to the library developed for the Cray-T3D [6]. The function **mype**() returns to each processor a unique number in the interval [0, n$pes − 1]. The function **npes**() returns a value equal to the number of processors assigned at run-time. It can be used to check consistency with the value n$pes set at compile or load-time:

```
if(npes() .ne. n$pes) stop "error"
```

The function **log_pes**() returns the base two logarithm of the number of processors at run-time and can be used in the same way to check consistency.

The functions **get**() and **put**() represent a set of communication primitives that the programmer may prefer to use instead of F^{--} syntax. The functions **global_sum**(), **global_min**(), **global_max**(), and **global_collect**() perform important global reduction operations. For example, the **global_sum**() function written in F^{--} syntax has the following form:

```
subroutine global_sum(x,n,work)
real x(n|0:n$pes-1),work(n)
integer bit,dim
dim = log$pes
if(dim .eq. 0) return
me = mype()
bit = 1
do i=1,dim
    mypal=xor(me,bit)
    bit=shiftl(bit,1)
    call barrier()
    work(:) = x(:|mypal)
    call barrier()
    x(:)=x(:)+work(:)
enddo
return
end
```

The **atomic_update**() function allows protected addition into a specified address and the **atomic_swap**() function allows protected interchange of data at a specified address. The **broadcast**() function sends data to a specified address on all processors or to a subset of processors.

Other functions such as matrix transpose, which have become part of the Fortran 90 language, are included as part of libraries to support F^{--} but are not part of the language extension itself.

9 SYNCHRONIZATION AND CONTROL

Synchronization and control are the responsibility of the programmer. The programmer must prevent race conditions where more than one processor writes to the same location in memory at the same time.

Two kinds of synchronization are useful. Full synchronization of all processors,

```
call barrier()
```

and partial synchronization of a subset of processors,

```
call sync(list)
```

The array `list` contains a list of processors that participate in the barrier. The first word in the array tells how many processors participate in the barrier. It is up to the programmer to make sure all of the appropriate processors reach the correct barriers.

F^{--} supports the SPMD programming model in a natural way. But it also supports the MIMD programming model. Each processor may branch, based on its processor number, into separate code within a program. For example,

```
if(mype() .eq. 0) then
      call subx()
else
      call suby()
endif
```

Critical regions can be programmed explicitly using the **atomic_swap**() function. For example,

```
1 continue
  if(atomic_swap(-1) .ne. -1) then
     call subx()
     if(atomic_swap(+1) .ne. -1)
         stop "error"
  else
     go to 1
  endif
```

The **atomic_swap(x)** function swaps the control variable x into a preset control word in global memory. If it returns with the same value of x, then another processor is already in the critical region.

10 EXTENSION TO SHARED MEMORY AND CLUSTERED ARCHITECTURES

F^{--} syntax was designed with a distributed memory machine in mind but it can be applied to other machines as well. It could become a standard, portable addition to the Fortran language if we establish conventions for the meaning of F^{--} syntax for shared memory machines and for clustered machines and for networks of machines. The programmer is responsible for writing code that makes sense on the chosen machine. With careful attention to detail, it is possible to write code that runs on different kinds of machines by adjusting parameters to fit the target architecture.

On a shared memory multiprocessor, for example, F^{--} syntax refers to a coordinate representation of shared memory. The notation

```
real a(m,n|p,q)
```

describes a matrix of size $(m \times n)$ replicated $(p \times q)$ times in different parts of shared memory similar to the idea of TASK COMMON.

For clustered architectures or for networks of workstations, the syntax extends to multiple hierarchies of memory. For example, the dimension statement

```
real a(m,n|p,q|j,k)
```

might describe a matrix of size $(m \times n)$ in local memory distributed first across a local cluster of processors in a $(p \times q)$ grid and then across remote clusters in a $(j \times k)$ grid.

11 F^{--} SYNTAX AND TENSOR NOTATION

F^{--} syntax has a natural relationship to tensor notation. In fact, the minus-minus notation used as a superscript is meant to evoke memories of tensor notation in the reader's mind. The authors of references [13] and [14] use tensor notation for parallel algorithm design somewhat differently from the approach presented here.

Recall what the coordinate representation of an operator means [15]. Let V_n and V_m be vector spaces of dimension n and m and let

$$A : V_n \to V_m \tag{7}$$

be an abstract linear operator from V_n and V_m. The abstract operator A does not care how we represent it, but to perform computations, we must pick a coordinate representation for it. For example, let

$$\mathbf{e}^n = \{e_1^n, \ldots, e_n^n\} \tag{8}$$

be a basis for V_n and let

$$\mathbf{e}^m = \{e_1^m, \ldots, e_m^m\} \tag{9}$$

be a basis for V_m. Then

$$a_i^j = (e_i^m | A | e_j^n). \tag{10}$$

is the component of the operator A in the ith row and jth column for these basis sets. In Fortran syntax, this component is written $a(i, j)$. Equation 1 is the rule for placing the component in memory.

We are free to reorder or regroup the basis vectors anyway we like. For a $(p \times q)$ processor grid, we may choose to think of the basis sets as tensor products

$$\mathbf{e}^{n/q} \otimes \mathbf{e}^q, \qquad \mathbf{e}^{m/p} \otimes \mathbf{e}^p. \tag{11}$$

In the tensor product basis, the operator A has the coordinate representation

$$a_{ir}^{js} = (e_i^{m/p} \cdot e_r^p | A | e_j^{n/q} \cdot e_s^q). \tag{12}$$

We may regroup the basis vectors such that

$$a_{ir}^{js} = (e_r^p | A_i^j | e_s^q) \tag{13}$$

where A_i^j is the $(m/p \times n/q)$ block of data owned by processor (r, s). In F^{--} syntax, this component is written $a(i, j \mid r, s)$. Equation 3 is the rule for assigning the blocks to processors.

Matrix multiplication has a natural representation in tensor notation that translates easily into F^{--} syntax. In the original basis,

$$c_i^j = a_i^k b_k^j \tag{14}$$

where repeated indices imply summation. In the tensor product basis,

$$c_{ip}^{jq} = a_{ip}^{kR} b_{kR}^{jq} \tag{15}$$

where repeated indices k and R imply summation over both. Since the uppercase symbol R is a processor index, the summation over R implies global communication. For square matrices with n$pes an even power of 2, this block-based algorithm translates into F^{--} syntax as follows:

```
real a(m,m|p,q), b(m,m|p,q), c(m,m)
do ipe=0,n$pes/2-1
  pe = mod(my_col+ipe,n$pes/2) + 1
  do j = 1,m
    do k = 1,m
      do i = 1,m
      c(i,j)=c(i,j)+
        a(i,k|my_row,pe)
          *b(k,j|pe,my_col)
      enddo
    enddo
  enddo
enddo
```

For matrices distributed by columns rather than by blocks, only the basis set $\mathbf{e}^n$ is split into a product so that

$$c_i^{jp} = a_i^{hR} b_h^{jp} \tag{16}$$

where $h = R \cdot (n/p) + k$. Summation over h implies summation over both k and R. This column-based algorithm translates into F^{--} syntax as follows:

```
parameter(m=n/n$pes)
real a(n,m|0:n$pes-1),b(n,m),c(n,m)
do pe=0,n$pes-1
   remote_pe = mod(mype()+pe,n$pes)
   kk = remote_pe*m
   do j = 1,m
      do k = 1,m
        do i = 1,n
         c(i,j)=c(i,j)+
           a(i,kk+k|remote_pe)
             *b(kk+k,j)
        enddo
      enddo
   enddo
enddo
```

Matrix transpose assumes a simple form in F^{--} syntax. To avoid end cases, assume that *npes* processors are arranged in a logical square grid of size $p\_row = p\_col = \sqrt{npes}$ and that each processor knows its own coordinates my_row and my_col within the grid. A square matrix of size n is distributed among the processors such that each processor owns a square block of size $n/\sqrt{npes}$. Then the transpose just requires the programmer to interchange the data indices and the processor indices:

$$a_{ip}^{jq} = b_{jq}^{ip}. \tag{17}$$

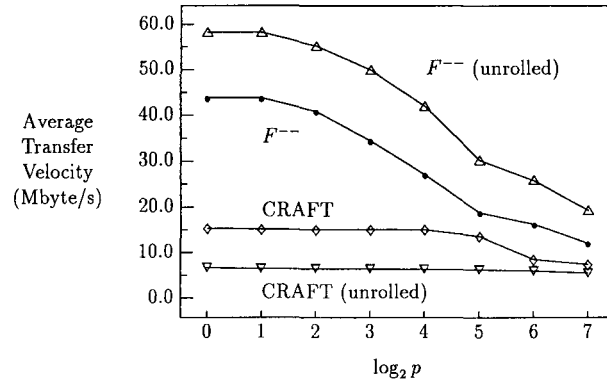

FIGURE 2 Average transfer velocity as function of the number of processors for matrix transpose measured on the Cray-T3D with machine frequency $\nu = 150$ MHz. The size of the matrix increases with the number of processors such that each has a local block of size (256 × 1024) 64-bit words. The four curves represent CRAFT (◇), F^{--} (•), unrolled CRAFT (▽), and unrolled F^{--} (△). Do Loops were unrolled by four to take advantage of the four word cache line.

The F^{--} code is the following:

```
subroutine transpose(a,b,m,n,pe_row,
    pe_col,my_row,my_col)
real a(m,n|pe_row,pe_col), b(n,m)
do i=1,m
   do j=1,n
      a(i,j|my_col,my_row) = b(j,i)
   enddo
enddo
return
end
```

Figure 2 shows results for matrix transpose measured on the Cray-T3D. CRAFT is Cray's version of a data parallel language [2]. The CRAFT code is the following:

```
      subroutine transpose(a,b,m,n)
cdir$ shared a(:block,:block),
             b(:block,:block)
      real a(m,n), b(n,m)
      integer i,j
cdir$ do shared (i,j) on b(j,i)
      do i=1,m
         do j=1,n
            a(i,j) = b(j,i)
         enddo
      enddo
cdir$ nobarrier
      return
      end
```

In practice, the programmer would probably call a library subroutine to perform common matrix operations.

The purpose of these examples is to illustrate what happens should the programmer need to write communication patterns that are not contained in standard libraries. In such cases, as far as performance is concerned, the advantages of F^{--} over CRAFT are clear. For small numbers of processors, the transfer velocities approach the same high values as shown in Figure 1. Unrolling the do loops by four to take advantage of cache behaves as expected using F^{--} syntax while the CRAFT result is counterintuitive. As the number of processors increases, the velocity drops using F^{--} syntax because of contention on the interconnection network [7]. The CRAFT velocities are less sensitive to contention because the traffic on the network is lower.

Which programming style appeals to a given programmer is a matter of taste. With F^{--} syntax, the programmer maintains an explicit picture of the data layout among processors. With the CDIR$ SHARED directive in CRAFT or the !HPF$ DISTRIBUTE directive in high-performance Fortran (HPF), the compiler maintains an implicit picture of the data layout. The F^{--} style puts more burden on the programmer but yields better performance. The CRAFT or HPF style puts less burden on the programmer but, at least for now, yields poorer performance.

12 CURRENT IMPLEMENTATION

All of the examples in this article are running on the Cray-T3D. In the absence of compiler support for F^{--} syntax, a simple coding trick fools the Fortran compiler [6]. To satisfy condition (5), the programmer allocates static arrays to communicate between processors, for example, by placing the arrays in a common block somewhere in the program. The following lines of code:

```
real a(m,n), b(m,n)
pointer (ptr,remote_a(m,n))
ptr = mpp_annex(a,ipe,num,icode)
```

create an alias for the array a(m, n) called

```
remote_a(m,n).
```

The function **mpp_annex**() inserts the value *num* into an appropriate position of the address of the aliased array pointing to a hardware supported lookup table containing the remote processor number ipe and the function code icode. After a call to function **mpp_annex**(), a replacement statement such as

```
remote_a(:,:) = b(:,:)
```

moves every matrix element b(i, j) from local memory to a(i, j) in remote memory.

13 SUMMARY

Fortran syntax is an intuitive convention for translating mathematical formulas into machine instructions [16]. It defines a linear map of data across the computer's memory. F^{--} extends Fortran syntax to define a linear map across processors. Its syntax is analogous to tensor notation. Algorithms written in tensor notation translate easily into F^{--} code.

F^{--} explicitly emphasizes the difference between local and remote memory. It gives the programmer complete control over the placement and manipulation of data. It exposes the underlying reality that using data from remote memory is more costly than using data from local memory.

F^{--} syntax allows compiler developers to concentrate on local code optimization. The compiler always knows whether an address is local or remote. The vertical line syntax explicitly tells the compiler that the local address happens to be in memory on a remote processor. The compiler needs to recognize invariant code related to remote processor references and to schedule that code efficiently just as it schedules local code.

The programmer defines the processor grid in a convenient way, adding or deleting vertical lines with no ambiguity in meaning. The programmer changes shapes or dimensions of the processor grid with the same rules as Fortran 77. Incorrect indexing generates run-time errors not compile-time errors. F^{--} places no restrictions on array sizes and requires no compiler directives.

F^{--} is not a new language so there are no complicated language issues to resolve. It is a simple extension to Fortran 77 that is explained in one page of text in Section 3. The rest of the article contains examples for using the new syntax. All the rules that apply to data indices apply in the same way to processor indices. Just as data indices allow the programmer to think of data in a coordinate grid, F^{--} syntax allows the programmer to think of processors in a coordinate grid. Abstract operators are independent of the particular grid chosen. Hence the programmer may freely switch back and forth from representation to representation both in data coordinates and in processor coordinates. This covariance property is fundamental to physical theory. The rules of Fortran were designed to support it and F^{--} extends this support to processor coordinates.

NOTE ADDED IN PROOF

I thank the anonymous referees for helpful comments and suggestions for improving the original version of this paper. I thank Dr. Geert Wenes, my occasional office mate at Cray Research now with IBM, for helping me pick F^{--} as the name for this extension to Fortran.

During the long delay between acceptance and publication of this paper, I have extended F^{--} to the Fortran 90 language to remove some of the restrictions forced on the extension by the limitations of the Fortran 77 language. The interested reader should see the following papers:

1. R. W. Numrich, J. L. Steidel, B. H. Johnson, B. D. de Dinechin, G. Elsesser, G. Fischer, and T. MacDonald, "Definition of the F^{--} extension to Fortran 90," in *Proceedings of the 10th International Workshop on Languages and Compilers for Parallel Computing*, August 1997, Springer-Verlag, in press.

2. R. W. Numrich and J. L. Steidel, "F^{--}: A simple parallel extension to Fortran 90," *SIAM News*, September 1997.

REFERENCES

[1] C. Koelbel, D. Loveman, R. Schreiber, G. Steele, and M. Zosel, *The High Performance Fortran Handbook.* Cambridge, MA: The MIT Press, 1994.

[2] D. M. Pase, T. MacDonald, and A. Meltzer, "The CRAFT Fortran programming model," *Sci. Prog.*, vol. 3, pp. 227–253, 1994.

[3] High Performance Fortran Forum, "High performance Fortran / Journal of development," *Sci. Prog.*, vol. 2, no. 1–2, 1993.

[4] B. Chapman, P. Mehrotra, and H. Zima, "Programming in Vienna Fortran," *Sci. Prog.*, vol. 1, pp. 31–50, 1992.

[5] R. W. Numrich, "An explicit node-oriented programming model," Cray Research, Inc., Eagan, MN, Tech. Rep., March 1991.

[6] R. W. Numrich, "The Cray T3D address space and how to use it," Cray Research, Inc., Eagan, MN, Tech. Rep., April 1994.

[7] R. W. Numrich, P. L. Springer, and J. C. Peterson, "Measurement of communication rates on the Cray T3D interprocessor network," in *Proc. High-Performance Computing and Networking, Vol. 2: Networking and Tools*, W. Gentzsch and U. Harms, Eds. New York: Springer-Verlag, 1994, pp. 150–157.

[8] A. Sawdey, M. O'Keefe, R. Bleck, and R. W. Numrich, "The design, implementation, and performance of a parallel ocean circulation model," in *Proc. Sixth ECMWF Workshop on the Use of Parallel Processors in Meteorology*, World Scientific Publishers, 1995, pp. 523–550.

[9] J. H. Fry, "MYC: A user-optimized C compiler," Cray T3D Applications Meeting, Munich, Tech. Rep., Oct. 1993.

[10] W. W. Carlson and J. M. Draper, "AC for the T3D," Supercomputer Research Center, Institute for Defense Analysis, 17100 Science Drive, Bowie, MD 20715-4300, Tech. Rep. SRC-TR-95-141, Feb. 1995.

[11] D. E. Culler, A. Dusseau, S. Copen Goldstein, A. Krishnamurthy, S. Lumetta, T. von Eicken, and K. Yelick, "Parallel programming in Split-C," in *Proc. Supercomputing '93*, 1993, pp. 262–273.

[12] J. R. Rose and G. L. Steele Jr., "C*: An extended C language for data parallel programming," in *Proc. 2nd Int. Conf. Supercomputing*, vol. 2, 1987, pp. 2–16.

[13] D. L. Dai, S. K. S. Gupta, S. D. Kaushik, J. H. Lu, R. V. Singh, C.-H. Huang, P. Sadayappan, and R. W. Johnson, "EXTENT: a portable programming environment for designing and implementing high-performance block recursive algorithms," in *Proc. Supercomputing '94*, 1994, pp. 49–58.

[14] S. D. Kaushik, C.-H. Huang, J. R. Johnson, R. W. Johnson, and P. Sadayappan, "Efficient transposition algorithms for large matrices," in *Proc. Supercomputing '93*, 1993, pp. 656–665.

[15] P. R. Halmos, *Finite Dimensional Vector Spaces*, 2nd ed. Princeton, NJ: D. Van Nostrand Company, Inc., 1958.

[16] M. J. Merchant, *Fortran 77 Language and Style.* Belmont, CA: Wadsworth Publishing, 1981.