To appear in Proceedings of the Third Workshop on Languages, Compilers and Runtime Systems for Scalable Computers, Troy, NY, May 1995

HPF Array Statements: Communication Generation and Optimization

Ashwath Thirumalai J. Ramanujam*
Department of Electrical and Computer Engineering
Louisiana State University, Baton Rouge, LA 70803
{ash,jxr}@gate.ee.lsu.edu

1 Introduction

Programming massively parallel distributed memory machines involves partitioning data and computation across processors. Several compilers like Fortran D [4] and High Performance Fortran [3] include directives such as alignment and distribution that describe how data and computation are mapped onto the processors in a distributed-memory multiprocessor. A compiler for HPF that generates code for each processor has to compute the sequence of local memory addresses accessed by each processor and the sequence of sends and receives for a given processor to access non-local data. In this paper, we present a novel approach for code generation based on integer lattices. Using the fact that the set of elements referenced can be generated by integer linear combinations of basis vectors, we also present algorithms for deriving the communication sets of each processor. In both Fortran D and HPF arrays are aligned to an abstract Cartesian grid called a *template*. The template is then distributed across the various processors. The distribution of computation in most compilers follows the *owner-computes* rule. That is, the processor performs only those computations (or assignments) for which it owns the left hand side variable. Access to non-local right hand side variables is achieved by inserting sends and receives. In order to generate node code we need to know the sequence of local memory addresses accessed by each processor and the sequence of sends and receives for a given processor to access non-local data.

2 Identical Alignment of Arrays to Templates

We consider an array A identically aligned to the template T; this means that if, $A(i)$ is aligned with $T(ai + b)$, then the alignment stride $a = 1$ and the alignment offset $b = 0$. Further let this template be distributed in a block-cyclic fashion with a block size of k across p processors. This is also known as a CYCLIC(k) distribution [3]. This mapping of the elements of A to the processor memories is shown in the Figure 1(a). Though the elements of A are stored in the processor memories in a linear fashion as shown in Figure 1(b), we intend to treat the storage allocated as a two dimensional grid shown in Figure 1(a) in the following analysis. A typical HPF assignment

*Supported in part by an NSF Young Investigator Award CCR–9457768, and NSF grant CCR–9210422, and by the Louisiana Board of Regents through contract LEQSF (1991-94)-RD-A-09.

Processor 0			Processor 1			$\cdots$	Processor p-1		
0	$\cdots$	k-1	k	$\cdots$	2k-1	$\cdots$	(p-1)k	$\cdots$	pk-1
pk	$\cdots$	(p+1)k-1	(p+1)k	$\cdots$	(p+2)k-1	$\cdots$	(2p-1)k	$\cdots$	2pk-1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Figure 1(a): Mapping of array identically aligned to template T with `CYCLIC`(k) distribution on p processors.

Global index (Processor 0)	0	$\cdots$	k-1	pk	$\cdots$	(p+1)k-1	$\cdots$
Local index (Processor 0)	0	$\cdots$	k-1	k	$\cdots$	2k-1	$\cdots$

Figure 1(b): Storage pattern of the array shown in Figure 1(a) in the local memory of Processor 0.

Figure 1: Global and local addresses for data mappings

statement is of the form,

$$A(l : h : s) = \cdots\cdots \tag{1}$$

where s is the access stride. Given an array statement with HPF-style data mappings, it is our aim to generate SPMD code along with communication for the different processors.
Consider the case where p=3, k=4 and s=5. Figure 2 shows the allocation of the array elements along with the corresponding global and local addresses. Each row of the global address space is of length pk = 12. As mentioned earlier we shall now treat the global address space as a two dimensional space, and every element of A has an address of the form $(x, y)^T$ in this space. Here x gives the number of the row to which this element belongs and y gives the offset of the element in that row.

2.1 Code generation for identical alignment for the array section (0:N:s)

We have assumed here that A is identically aligned to the template T. So as is evident in the Figure 1(a), we assign a block of k cells of the template to each of the p processors and then wrap around and assign the rest of the cells in a similar fashion. It should be obvious that the sequence of the array elements accessed in any of the processors can be obtained by strip mining the the array assignment statement in (1) with a strip length of pk and appropriately restricting the inner loop limits. The dependence vectors that arise on strip mining the array expression with a strip length of pk are, $(1, -\ pk \bmod s)^T$ and $(0, s)^T$. Every element of A that is accessed can be spanned by the sum of some positive integer multiple of these dependence vectors. We call such vectors the basis vectors of the two dimensional grid. So if B is the basis matrix of the two dimensional space shown in Figure 1(a), then,

$$B = \begin{bmatrix} 1 & 0 \\ -(pk \bmod s) & s \end{bmatrix}$$

The determinant of B is the access stride s itself. Using a transformation $T = B^{-1}$ and Fourier-Motzkin elimination, we derive the following code. For details refer to [10].
Figure 3 shows the code generated for processor $m (0 \leq m < p)$. Figure 3(a) contains the unopti-

Processor 0				Processor 1				Processor 2			
$\boxed{0}^{0}$	1	2	3	4	$\boxed{5}^{1}$	6	7	8	9	$\boxed{10}^{2}$	11
12	13	14	$\boxed{15}^{7}$	16	17	18	19	$\boxed{20}^{4}$	21	22	23
24	$\boxed{25}^{9}$	26	27	28	29	$\boxed{30}^{10}$	31	32	33	34	$\boxed{35}^{11}$
36	37	38	39	$\boxed{40}^{12}$	41	42	43	44	$\boxed{45}^{13}$	46	47
48	49	$\boxed{50}^{18}$	51	52	53	54	$\boxed{55}^{19}$	56	57	58	59
$\boxed{60}^{20}$	61	62	63	64	$\boxed{65}^{21}$	66	67	68	69	$\boxed{70}^{22}$	71
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

(a): Memory layout

do u = 0, N'
  $t = 2u + 4$
  do v = $\lceil \frac{t}{5} \rceil$, $\lfloor \frac{3+t}{5} \rfloor$
    $A[4u + 5v - t] = ...$
  end do
end do

(b): Code

Figure 2: Memory layout of array A (with global and local addresses of elements accessed) and code for p=3, k=4 and s=5.

mized code for A(0:N:s). It should be evident that the access pattern for the array A repeats after every $\frac{s}{\gcd(s,pk)}$ blocks of elements. So all we need to do is, store the access pattern of the elements in the first $\frac{s}{\gcd(s,pk)}$ blocks and use this information to determine the remaining elements accessed. Figure 3(b) incorporates this optimization. Figure 3(c) contains the code for A(l:h:s).
The SPMD code generated for processor 1 (using code in Figure 3(a)) for the case p=3, k=4 and s=5 (in Figure 2(a)) is shown in Figure 2(b).

3 Communication Generation

Communication involves sending those data items that a given processor owns and is required by other processors, and receiving data sent by other processors. A typical HPF-style assignment statement is,

$$A(l_1 : h_1 : s_1) = B(l_2 : h_2 : s_2) \tag{2}$$

3.1 Receive Set Generation

Given the global address of a LHS element, we can easily determine the processor that owns the RHS element and the local address of the RHS element on that processor using the following expressions,

$$m = (i \text{ div } k) \bmod p \qquad\qquad L = k * (i \text{ div } pk) + i \bmod k$$

$\alpha = pk \bmod s$
$k' = k - 1$
$\beta = mk$
do u = 0,N'
  $t = \alpha u + \beta$
  do v = $\lceil \frac{t}{s} \rceil$, $\left\lfloor \frac{k'+t}{s} \right\rfloor$
    $A[uk + vs - t] = \cdots\cdots$
  end do
end do

(a)Code for A(0:N:s)

$\alpha = pk \bmod s$; $\beta = mk$; $i = 0$;
$s' = \frac{s}{\gcd(s,pk)}$;$N' = \left\lceil \frac{N}{s'} \right\rceil$;
do $u = 0, s'$
  $t = \alpha u + \beta$
  do $v = \lceil \frac{t}{s} \rceil$, $\left\lfloor \frac{k'+t}{s} \right\rfloor$
    $U[i] = uk + vs - t$; $i{+}{+}$
  end do
end do
do $w = 0, N'$
  $i = i + s'$
  do $u = 0, k - 1$
    $A[i * k + U(u)] = \cdots$
  end do
end do

(b)Optimized code for A(0:N:s)

$\alpha = pk \bmod s$; $k' = k - 1$;
$\beta = mk - l$; $u = \mathrm{lb} = \left\lfloor \frac{l}{pk} \right\rfloor$
if $(l \bmod (pk)) > (mk + k - 1)$ **then** $++lb$;
if $\left\lfloor \frac{l}{k} \right\rfloor \bmod p == m$
  do $v = (l \bmod k), k - 1, s$
    $A[uk + v] = \cdots$
  enddo
  ++lb
endif
do $u = lb, \frac{h}{pk}$
  do $v = \left\lceil \frac{\alpha u+\beta}{s} \right\rceil$, $\left\lfloor \frac{k'+\alpha u+\beta}{s} \right\rfloor$
    $A[uk + vs - \alpha u + \beta] = \cdots$
  end do
end do

(c)Code for A(l:h:s)

Figure 3: Code for array statement for processor m

$\alpha = pk \bmod (s_1)$
$k' = k - 1$
$\beta = m * k - l_1$
do u $= \frac{l_1}{pk}, s_1 - 1 + \frac{l_1}{pk}$
  $t = \alpha u + \beta$
  do v $= \left\lceil \frac{t}{s_1} \right\rceil, \left\lfloor \frac{k'+t}{s_1} \right\rfloor$
    $i = \frac{(upk+mk+vs_1-t-l_1)s_2}{s_1} + l_2$
    Receive $B(k(i \text{ div } pk) + i \bmod k)$
      from Processor $(i \text{ div } k) \bmod p$
  end do
end do

(a): Code for computing the receive set

$\alpha = pk \bmod (s_2)$
$k' = k - 1$
$\beta = m * k - l_2$
do u $= \frac{l_2}{pk}, s_2 - 1 + \frac{l_2}{pk}$
  $t = \alpha u + \beta$
  do v $= \left\lceil \frac{t}{s_2} \right\rceil, \left\lfloor \frac{k'+t}{s_2} \right\rfloor$
    $i = \frac{(upk+mk+vs_2-t-l_2)s_1}{s_2} + l_1$
    Send $B(k(i \text{ div } pk) + i \bmod k)$
      to Processor $(i \text{ div } k) \bmod p$
  end do
end do

(b): Code for computing the send set

Figure 4: Generation of receive and send sets for the array statement $A(l_1 : h_1 : s_1) = B(l_2 : h_2 : s_2)$

where i is a global address, m is the processor to which i belongs to and L is the local address of i on processor m.
It should be evident that the access pattern in the global address space repeats itself after pk elements are accessed, i.e. after every s_1^{th} row, where s_1 is the access stride of the LHS array. To generate the receive set, all we need to do is run the code generated in the previous sections to obtain the global address of the first pk elements accessed on the LHS. For each of these elements we calculate the processor that owns the corresponding RHS element and local address of that element on it. The code is shown in Figure 4(a).

3.2 Send Set Generation

Our approach to the Send set generation is very similar to the Receive set generation shown in the previous section. It should be obvious that there is a one to one correspondence between the LHS array element and the RHS array elements. So the Send set for a processor is the same as the Receive set calculated assuming that the RHS element is actually the LHS element. Here again we

- Search for the first row of the array section B assigned to the processor and run the Send loop as shown in Section 3.2.
- Store the access patterns of the right hand side array section (in this case accesses to array B) for processor m in $Spattern_m$. For processor m, we run the following code:

  do $i = 0, p-1$
      Initialize $SList_i$
  enddo
  do $i = 0, sizeof(Spattern_m) - 1$
      q = Owner of LHS array element that needs $Spattern_m[i]$
      Append $Spattern_m[i]$ to $SList_q$
  enddo
  do $i = 0, p-1$
      $\mathcal{L}_i = sizeof(SList_i)$
      Initialize $SendBuffer_i$
      do $j = 0, \mathcal{L}_i - 1$
          $SendBuffer_i(j : \mathcal{L}_i \frac{max_2}{s_2 k} : \mathcal{L}_i) = B(SList_i[j] : max_2 : s_2 k)$
      enddo
  enddo

  where max_2 is the local address of the last element of B accessed on the processor.
- Send data in buffers to respective processors.

Figure 5: Send phase generation algorithm for the array statement in Equation 2

notice that the Send set pattern should repeat after every s_2^{th} row. The code that generates the Send set is shown in Figure 4(b).

3.3 Communication Algorithm

Some of the assumptions we make here are,

- We allocate memory for a temporary array corresponding to each RHS array.(The temporary array elements have the same alignment and are accessed with the same stride as the LHS array).
- Each processor maintains a separate send and receive buffer for all the m processors.

The communication algorithm for an array assignment statement is shown in Figures 5 and 6. As we have assumed a single template model, closed form expressions for the communication set generation exist for the case where $k = 1$ (template is cyclically distributed) and $k = \left\lceil \frac{N}{p} \right\rceil$ (template is block distributed) [7]. The algorithm shown in Figures 5 and 6 simplifies for these cases as well as the case where, s_{lhs} divides s_{rhs} or vice-versa, as the send and receive set patterns repeat after $\frac{s}{\gcd(s,pk)}$ rows itself.
Below, we show the communication set generation for the array assignment statement, $A(0 : h_1 : 5) = B(0 : h_2 : 3)$ where A and B are aligned identically with the template T, and $p = 3$ and $k = 4$. The Send set for processor 1 is generated as follows:

- Receive data from the p processors

- Search for the first row of the array section A assigned to the processor and run the Receive loop as shown in Section 3.1. We assume that elements received in this loop are stored in $Rpattern_m$ and array elements received from processor i is stored in $ReceiveBuffer_i$ for each processor $i = 0, p-1$.

- For processor m, we run the following code:

  do $i = 0, p-1$
      Initialize $RList_i$
  enddo
  do $i = 0, sizeof(Rpattern_m) - 1$
      q = Owner of RHS array element $Rpattern_m[i]$
      Append $Rpattern_m[i]$ to $RList_q$
  enddo
  Initialize $BTemp$
  do $i = 0, p-1$
      $\mathcal{R}_i = sizeof(RList_i)$
      do $j = 0, \mathcal{R}_i - 1$
          BTemp$(RList_i[j] : max_1 : s_1 k) = ReceiveBuffer_i(j : \mathcal{R}_i \frac{max_1}{s_1 k} : \mathcal{R}_i)$
      enddo
  enddo

  where max_1 is the local address of the last element of A accessed on the processor.

Figure 6: Receive phase generation algorithm for the array statement in Equation 2

```
do u = 0,2
    t = 3u + 4
    do v = ⌈t/3⌉, ⌊(3+t)/3⌋
        i = ((12u+3v+4−t)*5)/3
        Send B(12u + 3v + 4 − t)
            to Processor (i div 4) mod 3
    end do
end do
```

u	$v_{lb} = \lceil \frac{3u+4}{5} \rceil$	$v_{ub} = \lfloor \frac{3u+7}{5} \rfloor$	Send global index B	To Processor
0	2	2	6	2
1	3	3	18	1
2	4	4	30	0

The Receive set for processor 1 is generated as follows:

```
do u = 0,4
    t = 2u + 4
    do v = ⌈t/5⌉, ⌊(3+t)/5⌋
        i = ((12u+5v+4−t)*3)/5
        Receive B(i) from
            Processor (i div 4) mod 3
    end do
end do
```

u	$v_{lb} = \lceil \frac{2u+4}{5} \rceil$	$v_{ub} = \lfloor \frac{2u+7}{5} \rfloor$	Receive global index B	From Processor
0	1	1	3	0
1	2	1	¶	¶
2	2	2	18	1
3	2	2	24	0
4	3	3	33	2

¶: No array element is received

Finally we use the send and receive set information in the communication algorithm shown in Figures 5 and 6.

References

[1] B. M. Chapman, P. Mehrotra, and H. P. Zima. Vienna Fortran – a Fortran language extension for distributed memory multiprocessors. In J. Saltz and P. Mehrotra, editors, *Language, Compilers and Runtime Environments for Distributed Memory Machines*, pages 39–62. 1992.

[2] S. Chatterjee, J. R. Gilbert, F. J. E. Long, R. Schreiber, and S.-H. Teng. Generating local addresses and communication sets for data parallel programs. In *Proc. of ACM Symposium on Principles and Practices of Parallel Programming*, pages 149–158, May 1993.

[3] High Performance Fortran Forum. High Performance Fortran language specification version 1.0. Technical Report CRPC-TR92225, Rice University, May 1993.

[4] G. Fox, S. Hiranandani, K. Kennedy, C Koelbel, U. Kremer, C.-W. Tseng, and M. Wu. Fortran D Language Specification. Technical Report TR-91-170, Dept. of Computer Science, Rice University, Dec. 1991.

[5] S. K. S. Gupta, S. D. Kaushik, C.-H. Huang, and P. Sadayappan. On compiling array expressions for efficient execution on distributed-memory machines. Technical Report OSU-CISRC-94-TR19, Department of Computer and Information Science, The Ohio State University, April 1994.

[6] S. Hiranandani, K. Kennedy, J. Mellor-Crummey, and A. Sethi. Advanced compilation techniques for Fortran D. Technical Report CRPC-TR-93-338, Center for Research on Parallel Computation, Rice University, Oct. 1993.

[7] C. Koelbel. Compile-time generation of communication for scientific programs. In *Supercomputing '91*, pages 101–110, Nov. 1991.

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

[9] J. M. Stichnoth. Efficient compilation of array statements for private memory multicomputers. Technical Report CMU-CS-93-109, School of Computer Science, Carnegie Mellon University, Feb. 1993.

[10] A. Thirumalai and J. Ramanujam. Code generation and optimization for array statements in HPF. Technical Report, Louisiana State University, Nov. 1994.