Automatic Data Distribution for CFD Applications on Structured Grids

Michael Frumkin* and Jerry Yan

Numerical Aerospace Simulation Systems Division
NASA Ames Research Center

Abstract

Data distribution is an important step in implementation of any parallel algorithm. The data distribution determines data traffic, utilization of the interconnection network and affects the overall code efficiency. In recent years a number data distribution methods have been developed and used in real programs for improving data traffic. We use some of the methods for translating data dependence and affinity relations into data distribution directives. We describe an automatic data alignment and placement tool (ADAPT) which implements these methods and show it results for some CFD codes (NPB and ARC3D). Algorithms for program analysis and derivation of data distribution implemented in ADAPT are efficient three pass algorithms. Most algorithms have linear complexity with the exception of some graph algorithms having complexity $O(n^4)$ in the worst case.

1. Introduction

Well organized data layout improves performance of a parallel program. Data location and access patterns affect the amount of communications in the program, effectiveness of the cache, memory channel and communication network. Data traffic planning was important for gaining performance of vector and MIMD machines; it is still important for ccNUMA machines and will be more critical for machines with deeper memory hierarchy. Processing dependent data items requires the items to be loaded into the same processor at the same time. Data dependence and data layout determines the data traffic at the execution time. The volume and speed of data traffic can be optimized by appropriate alignment and distribution of the data.

In many applications (including Computational Fluid Dynamics (CFD) on structured grids) the data dependences are well structured. Such dependences can be expressed by structured affinity relations between arrays and can be translated into HPF (High Performance Fortran) data alignment and data distribution statements. Derived in this way, data distributions convert the program into a data parallel form with well organized data traffic.

*MRJ Technology Solutions, Inc. M/S T27A-2, NASA Ames Research Center, Moffett Field, CA 94035-1000; e-mail: {frumkin,yan}@nas.nasa.gov

In this paper we present an Automatic Data Alignment and Placement Tool (ADAPT) designed and implemented to facilitate conversion CFD codes to HPF. The tool was successfully applied to a number of CFD codes, including NAS Parallel Benchmarks (NPB), [4], and ARC3D, [8]. ADAPT implements a number of known data distribution techniques, see Section 9, involving memory traffic reduction via data distribution optimization. We start from the program model and an internal representation of the program. Then we analyze the array affinity on the loop level, nest level and routine level and develop translation of the affinity relation into HPF data mapping directives and discuss interprocedural data distributions. Then we compare ADAPT generated versions of NPB and ARC3D with hand written versions. We conclude the paper with a survey of existing data distribution methods, some observations, and plans for future work.

2. Program Model

In this context we plan to annotate Fortran programs with data traffic optimizing data alignment and distribution directives (we will use the word "mapping" to refer to both directives) and with HPF interface blocks. These directives will be inserted immediately before loop nests or before the first executable statement in a subroutine. These directives do not affect control flow of the program on single processor. The directives, however, can have side effects on parallel machines if processors are not synchronized before and after the execution of the REALIGN and REDISTRIBUTE (see [14]) directives. We will assume that the program is compiled with an HPF compliant compiler and that the processors involved in the execution are synchronized before and after REALIGN and REDISTRIBUTE directives.

For analysis and transformation purposes, a Fortran program is represented by control a flow graph [1]. The nodes of the graph are program basic blocks (BB) and arcs are possible transitions between BB. The control flow graph is augmented with parse trees for each statement in the block. ADAPT transforms the control flow graph into another graph which is used for generation of the annotated (and modified) program.

The data dependence and data affinity play crucial role in our development. We say "data item x depends on data item y if value of x depends on value of y" (see [1] p. 284,). If the data items are defined by variables X and Y respectively, we say that "X depends on Y". If X and Y are array variables, we call the set of pairs of dependent array elements "*affinity relation between* X and Y". The affinity relation can be translated into array alignment. The appropriate distribution of aligned arrays reduces the data movement performed by the program.

3. Loop Data Alignment

Loop level alignment is derived from the *affinity relation* between arrays referenced in the loop, and the loop *Data Transfer Graph* (DTG).

Affinity relation. For a pair of arrays used in the same loop statement, we define the affinity relation as a correspondence between array elements referred with the same value of the loop index. The affinity relation can be represented as a list of dependent pairs:

```
      do i=1,n
        a(idxa(i))=b(idxb(i))
      end do
c     Aff(a,b)={(idxa(i);idxb(i)), i=1,...,n}
```

An affinity relation exists for each pair of arrays referred in a statement. Similarly, a control dependence results in affinity relations between the arrays involved in the control statement and all arrays in each BB immediately dominated by the statement.

Data Transfer Graph[*]. Each variable used in the loop is represented as a node in DTG. Two nodes are connected by an arc if the value of the first variable is used for computation of the second, Figure 1. An arc connecting two arrays we annotate with an affinity relation between the arrays. For array A and any of its ancestor B an affinity relation between A and B can be inferred by applying the *affinity chain rule* along each directed path from B to A.

[*] The term Data Transfer Graph is used to avoid any confusion with Data Flow Graph where nodes are program statements, with statements A and B connected by an arc if a variable assigned in A is used in B.

```
do j = 1, m
  x1 = y(j,i11+k)
  x2 = y(j,i12+k)
  x(j,i21+k) = x1 + x2
  x(j,i22+k) = u*(x1 - x2)
end do
```

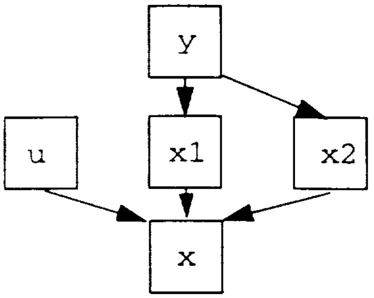


FIGURE 1. Data Transfer Graph of internal loop of vectorized autosorting FFT from FT benchmark.

The Affinity Chain Rule. Consider two statements in a loop

```
do i=1,n
  b(idxb1(i))=c(idxc(i))     !Aff(b,c)={(idxb1(i);idxc(i))}
  a(idxa(i))=b(idxb2(i))     !Aff(a,b)={(idxa(i);idxb2(i))}
end do
```

The chain rule allows to track affinity relation for indirectly dependent arrays:

```
Aff(a,c)={(idxa(i);idxc(j)),
                j = max{J: J<=i,idxb1(J))=idxb2(i)}}
```

Iterative application of affinity chain rule allows to find the closure of affinity relation on a loop in DTG. For an array assigned in the loop, it expresses the affinity relation of an array with each array it depends on. In general, due to the max operation involved in the chain rule, the affinity relation can not be expressed explicitly[*]. In practice, the affinity relation often can be expressed explicitly or can be approximated by a simple explicit relation. For example, if array indices are linear functions of the loop index, then the affinity relation can be represented by a linear mapping between array indices. The majority of affinity relations in CGD codes fall into three classes.

One-to-one affinity relations. In this relations each element of *a* depends on a single element of *b* and a single element of *c*,Figure 2. This affinity relation can be translated into an alignment with communication-free computations. The HPF alignment directive asserts that the corresponding elements are aligned. As result, no communications are necessary (regardless of the distribution). For the translation to be possible, array subscripts

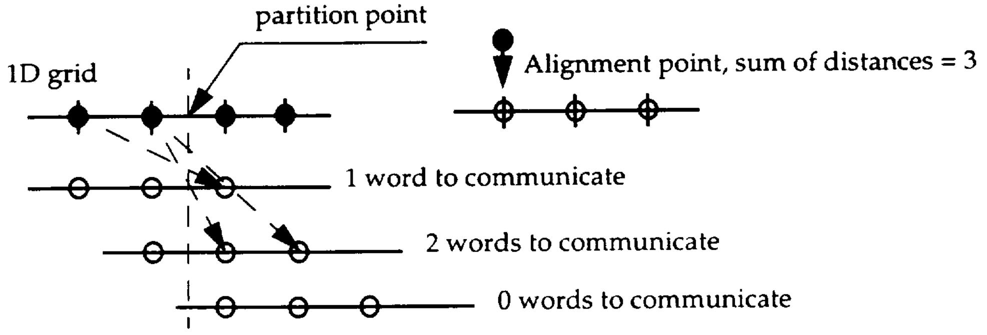
[*] In Section 4. we show that the problem of checking that an element of multidimensional array is affine with an element of another array is NP-complete.

should satisfy one condition: at most one subscript the coefficient different from {+1,-1}*.

```
!HPF$ ALIGN A(i) WITH B(i+3)
!HPF$ ALIGN B(j) WITH C(5*(n-j)-4)
!HPF$ DISTRIBUTE C(BLOCK)
      do i=1,n
        b(n-i)=c(5*i-4)
        a(i-1)=b(i+2)
      end do
c     Aff(a,c)={(i-1;5*j-4), j=max{J: J<=i,n-J=i+2}}
c or  Aff(a,c)={(i-1;5*j-4), j=max{J: n-2<=2*i,J=n-i-2}}
c or  Aff(a,c)={(i-1;5*n-5*i-14), n-2<=2*i}
```

FIGURE 2. Translation of one-to-one affinity relations into an alignment providing communication free computations. Note that the alignment statement for *c* is stronger then required by *Aff(a,c)*.

Stencil Affinity Relations. It is a relation where elements of array *b* affine with each element of array *a* (excluding some boundary elements) comprise the same stencil.Explicit difference operators for structured grids, for example, give rise to such relations. To optimize alignment for a stencil relation we note that for block distribution the message size per partition point is the sum of distances of the alignment point from the other stencil points, Figure 3. To minimize the message size we use *bisectors (*points splitting stencil points into two sets of equal size) as alignment points and we generate alignment as bisectors of the affinity points in the alignee.



3 words to communicate per partition point

*The syntax of the ALIGN directive ([14], p. 116) allows to use only align dummies in align subscript list and integer expressions of align dummies in align subscript.

FIGURE 3. The message size per partition point is sum of distances of the alignment point from the other stencil points.

True dependence. If there is a directed path in DTG which starts and ends in the same node, then there is a dependence carried by the loop. Iterations of the chain rule along the path expose an affinity of each array element with multiple elements of the same array. There are two important cases: the dependence is "true" when previously computed array elements are used and "anti" when a used array element is overwritten, [12]. The anti dependence can be eliminated by making an extra copy of the array. If the true dependence has a constant step d then a cyclic distribution onto d processors would create communication free computations.

4. Loop Nest Data Alignment and Distribution

If arrays referenced in a loop nest have more than one dimension (3 and 4 dimensional arrays are most common for structured grid CFD) then we must consider all loops surrounding computations with the arrays to obtain complete data affinity. In CFD applications, loop index bounds are usually linear functions of the surrounding loop indices, hence the nest index domain can be described as a set of integer points in a polyhedron*.

The chain rule for loop nests is similar to one for loops:

```
        do 10 I from PI
          b(idxb1(I))=c(idxc(I))
          a(idxa(I))=b(idxb2(I))
 10     continue
 c   Aff(a,c)={(idxa(I);idxc(J)),
```

*We will use multidimensional indices, functions and domains in this section. It will make presentation more compact and the analogy between loops and nests more transparent. For example, instead of

```
        do i=1,nx
          do j=1,ny(i)
            do k=1,nz(i,j)
              a(idxa1(i,j,k),idxa2(i,j,k),idxa3(i,j,k))=
                     b(idxb1(i,j,k),idxb2(i,j,k),idxb3(i,j,k))
            end do
          end do
        end do
```

we will write

```
        do 10 I from PI
          a(idxa(I))=b(idxb(I))
 10     continue
```

```
c                         J=max{j: j<=I,idxb1(j)=idxb2(I)}}    (*)
```

where the max operation and inequality *(j<=I)* are performed in the lexicographical order.

For statements with different nesting the chain rule is similar:

```
         do 10 I from PI
           do 20 J from PJ
             b(idxb1(I,J))=c(idxc(I,J))
20         continue
           do 30 K from PK
             a(idxa(I,K))=b(idxb2(I,K))
30         continue
10       continue
c      Aff(a,c)={(idxa(I,K);idxc(L,J)),
c              (L,J)=max{(l,j):l<=I,idxb1(l,j)=idxb2(I,K)}}
```

Example. *The problem of checking that an element of array is affine with an element of another array is NP-complete.* Let nest loop bounds are 0 and 1, meaning that *PI* is a boolean cube. Let *I=(1,...,1)* which means that the inequality *(j<=I)* is true for all *j* from *PI*. In this case if *idxb1(j)* is an arbitrary linear form of *j* then calculating the value of *J* in (*) is a special case of the *Boolean Knapsack Problem.* In practice, however, the number of array dimensions is fixed (it does not exceed 7 for CFD codes) and index functions are linear functions. In this case the affinity relation can be calculated in polynomial time by Lenstra's algorithm, see [22].

The chain rule allows to construct an affinity relation for each directed path coming to *a* from *b* and passing only through privatizable arrays. The union of these relations over all directed paths to *a* from *b* yields the final affinity relation between *a* and *b*. The relation lists all elements of *b* used for computation of the element of *a* and can be considered as one-to-many mapping. In most practical cases, this is a one-to-few stencil relation; a bisector[*] of the stencil gives an optimal alignment. Many CFD stencils have a center of symmetry which can be used as the alignment point.

Alignment with Systems of Linear Forms. Each array reference in the loop nest defines a mapping of the nest domain into array index space. Here we consider linear mappings:

$$d = AI + b$$

[*] Bisector of a finite set in n-dimensional space is a point such that each coordinate hyperplane passing through it bisects the set. 3-point LU stencil shows that not every set has a bisector.

where d ba an array and I be the nest indices, A is a matrix with constant elements and b is a vector with constant elements.

For linear index functions, the affinity relation can be expressed in the form:

$$\mathtt{Aff}(x_1,x_2)=\{(A_1*I+b_1;A_2*I+b_2),\ \mathtt{I\ from\ PI}\}$$

We want to translate this relation into alignment directives of the form

```
!HPF$ ALIGN Y1(i1,...,in) WITH Y2(m1*j1+c1,...,mn*jn+cn)
```

where $m_1,...,m_n$ are integer multipliers, $c_1,...,c_n$ are integer shifts, $(i_1,...,i_n)$->$(j_1,...,j_n)$ is a dimension permutation, Y_1 and Y_2 are x_1, x_2 or auxiliary templates.

In a common case, when one matrix (say $A1$) is nonsingular, the relation may be written explicitly:

$$\mathtt{Aff}(x_1,x_2)=\{(d_1;A_2A_1^{-1}d_1-A_2A_1^{-1}b_1+b_2),\ d_1\ \mathtt{from}\ A_1PI+b_1\} \qquad (1)$$

If the matrix $A_2A_1^{-1}$ can be transformed to an integer diagonal matrix diag$(m_1,...,m_n)$ by a permutation of the columns, then $-A_2A_1^{-1}b_1+b_2=(c_1,...,c_n)$ is an integer vector and the relation can be translated to an alignment directive:

```
!HPF$ ALIGN x1(i1,...,in) WITH x2(m1*j1+c1,...,mn*jn+cn)   (2)
```

where $(i_1,...,i_n)$->$(j_1,...,j_n)$ is the column permutation.

If the matrix $A_2A_1^{-1}$ cannot be reduced to a diagonal form by permutation of columns, then (1) would require presence of general linear forms in the align subscript list which is not permitted in HPF, see [14], p. 116. The relation (1) cannot be expressed by HPF ALIGN directive also if both $A_2A_1^{-1}$ and $A_1A_2^{-1}$ have noninteger elements. In such a case we can look for a submatrix of A_1 and A_2 having the property. If such submatrix exists, the alignment is performed on the corresponding set of indices.

The generation of alignment directives uses the alignment graph derived from DTG. The nodes of the alignment graph are non privatizable arrays of DTG. Two nodes of the alignment graph are connected if in DTG they are connected by a directed path passing through privatizable variables only. We annotate each arc of the alignment graph with a list of closures of affinity relations along each simple path connecting the arrays in DTG*.

* This is the most expensive operation of the method. It involves a few matrix multiplications for computation of the set of directed paths and has complexity $O(n^4)$, where n is the number DTG nodes.

For each arc in the alignment graph, we analyze the affinity relations attached to it. If all relations are expressed in the form (1) and have the same dimension permutation, then we will generate directive (2). In the directive, each multiplier is the greatest common divisor of multipliers of the relations, the shift is a bisector of the relations shifts and $(i_1,\dots,i_n)$->$(j_1,\dots,j_n)$ is the common dimension permutation.

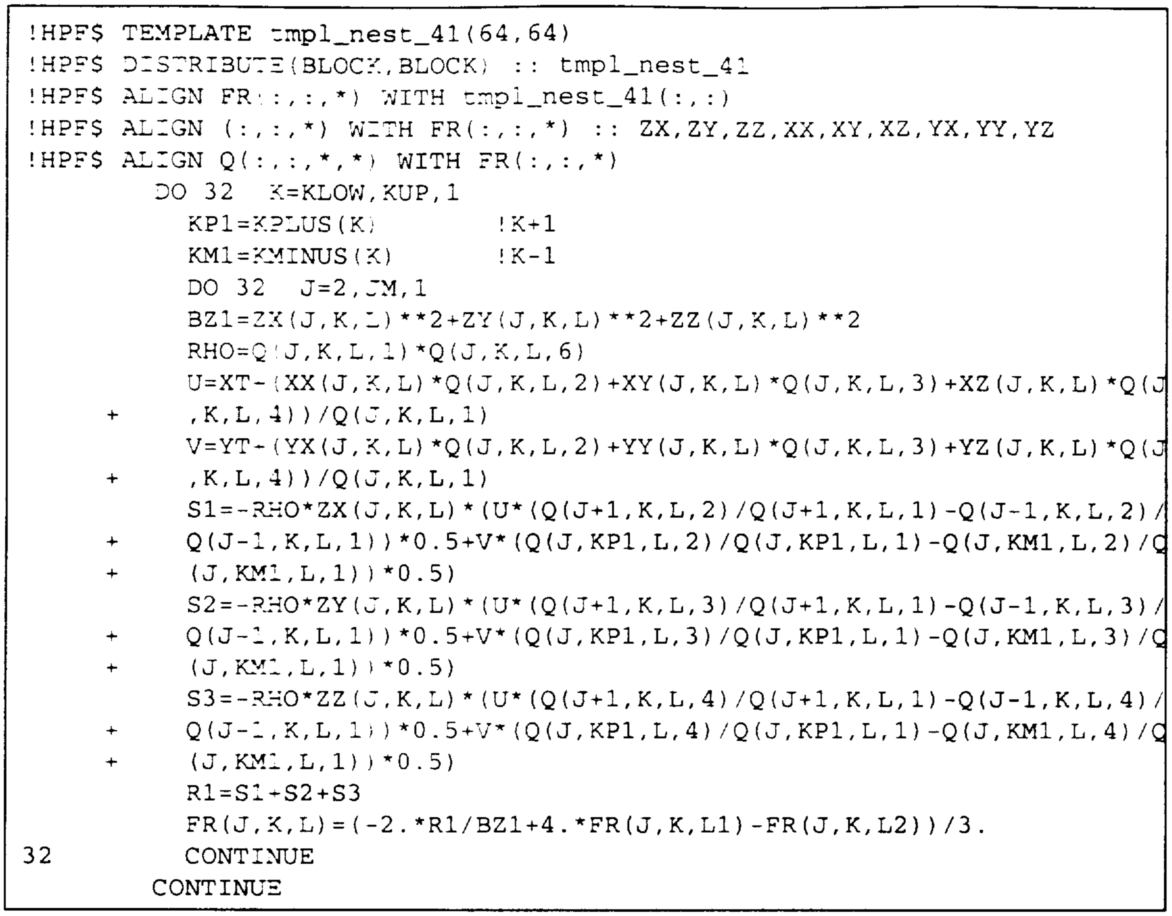
A maximal equivalence class of graph nodes having directed paths in both directions is called "strongly connected component" of the graph. The set of strongly connected components forms a directed acyclic graph. and we attach a template to each node of the graph. The affinity relation for loop nest then expressed as alignment of each array of the strongly connected component with the template, as shown in Figure 4. The generation of alignment statements for each connected component of a directed graph with affinity relations attached to every arc is performed in three steps.

- 1. A common template is generated for all leafs of the component. Each leaf is connected to the template with arc and appropriate affinity relation attached to the arcs.
- 2. A rooted spanning tree is constructed for each component with the template as the root.
- 3.For each non root node of the spanning tree, the alignment directive is generated for the arc leading from the node to the root (darker arcs in the Figure 5).

5. Subroutine Level Data Distribution

For data distribution on the subroutine level, we build a *phase control flow graph* (PCFG) [15]. The graph nodes are loop nests having at least one nonprivatizable array (following [15] we skip the loops (usually iteration loops) with the index not used as an array index). Two loop nests are connected by an arc if there is a possible transition from the last BB of one loop nest to the first BB of another loop nest. Given an alignment graph and the ALIGN and DISTRIBUTE directives (mapping directives) for each nest of PCFG; is it possible to combine the directives for a pair of adjacent nests in PCFG?

In simple cases (if alignment is the same in both nests) the answer can be obtained by comparing distributed dimensions of arrays in each nest. In general, however, the answer has to be obtained by attaching the second alignment graph to the first graph and getting



```
!HPF$ TEMPLATE tmpl_nest_41(64,64)
!HPF$ DISTRIBUTE(BLOCK,BLOCK) :: tmpl_nest_41
!HPF$ ALIGN FR(:,:,*) WITH tmpl_nest_41(:,:)
!HPF$ ALIGN (:,:,*) WITH FR(:,:,*) :: ZX,ZY,ZZ,XX,XY,XZ,YX,YY,YZ
!HPF$ ALIGN Q(:,:,*,*) WITH FR(:,:,*)
         DO 32  K=KLOW,KUP,1
           KP1=KPLUS(K)        !K+1
           KM1=KMINUS(K)       !K-1
           DO 32  J=2,JM,1
           BZ1=ZX(J,K,L)**2+ZY(J,K,L)**2+ZZ(J,K,L)**2
           RHO=Q(J,K,L,1)*Q(J,K,L,6)
           U=XT-(XX(J,K,L)*Q(J,K,L,2)+XY(J,K,L)*Q(J,K,L,3)+XZ(J,K,L)*Q(J
     +     ,K,L,4))/Q(J,K,L,1)
           V=YT-(YX(J,K,L)*Q(J,K,L,2)+YY(J,K,L)*Q(J,K,L,3)+YZ(J,K,L)*Q(J
     +     ,K,L,4))/Q(J,K,L,1)
           S1=-RHO*ZX(J,K,L)*(U*(Q(J+1,K,L,2)/Q(J+1,K,L,1)-Q(J-1,K,L,2)/
     +     Q(J-1,K,L,1))*0.5+V*(Q(J,KP1,L,2)/Q(J,KP1,L,1)-Q(J,KM1,L,2)/Q
     +     (J,KM1,L,1))*0.5)
           S2=-RHO*ZY(J,K,L)*(U*(Q(J+1,K,L,3)/Q(J+1,K,L,1)-Q(J-1,K,L,3)/
     +     Q(J-1,K,L,1))*0.5+V*(Q(J,KP1,L,3)/Q(J,KP1,L,1)-Q(J,KM1,L,3)/Q
     +     (J,KM1,L,1))*0.5)
           S3=-RHO*ZZ(J,K,L)*(U*(Q(J+1,K,L,4)/Q(J+1,K,L,1)-Q(J-1,K,L,4)/
     +     Q(J-1,K,L,1))*0.5+V*(Q(J,KP1,L,4)/Q(J,KP1,L,1)-Q(J,KM1,L,4)/Q
     +     (J,KM1,L,1))*0.5)
           R1=S1+S2+S3
           FR(J,K,L)=(-2.*R1/BZ1+4.*FR(J,K,L1)-FR(J,K,L2))/3.
32         CONTINUE
         CONTINUE
```

FIGURE 4. Example of the generated directives for one of the nests of ARC3D.

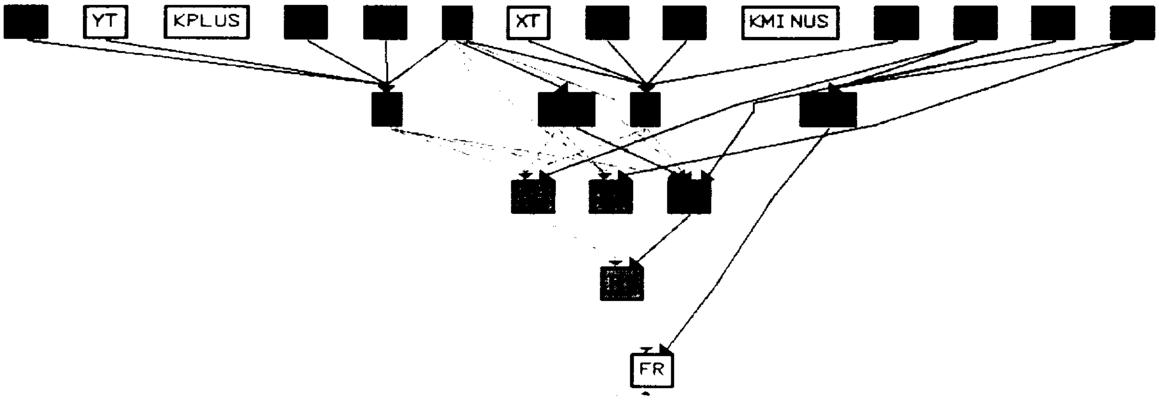


FIGURE 5. The data transfer graph of the nest of the Figure 4. Dark arcs show a spanning tree.

a mapping for the combined graph. If merging the mapping directives does not decrease the number of distributed dimensions the scope of the mapping can be expanded to include both nests. If the common number of distributed dimensions is 0, then either a redistribution between the nests or pipeline the computations with data having dependence between sections. Currently we use a simple trade-off model between redistribution and pipelineing to choose between these alternatives. The redistribution of an array of size N on p processors requires communication of N-N/p elements. The pipelined computation requires to communicate $f*N*p/d$ elements, where d is the array size in the pipelined direction and f is the number of the dependence vectors. The cost of pipeline startup is a factor of $1+pd/N$ of the execution time. A similar model is used to trade-off redistribution and serial execution of a nest.

The propagation of the mapping directives along arcs of PCFG results in annotation of each source and sink of the graph with the directives. These annotations, together with information on in/out/through routine arguments and on the common blocks used in the routine, the final redistributions are made and HPF routine interface are generated. The HPF standard requires a routine to preserve the distribution of arrays visible to other subroutines. To comply with this requirement we perform adjustment of the annotations:

- For each array argument, we choose the mapping of the array at one of the source node of PCFG; include this mapping (as prescriptive mapping) at the subroutine interface; and remove the mapping directives from the source node.
- For each leaf node of PCFG, we compare the final mapping of each subroutine argument with the mapping on the leaf and restore the mapping if necessary.

6. Interprocedural Data Distribution

So far we assumed that statements do not include routine or function calls. To handle routine calls we can either inline the routine or use HPF routine interface to express the data mapping in the routine. The inlining requires the same analysis at each call statement and may not result in any useful distribution if there is no single data distribution inside the routine. The use of HPF routine interface limits the expression of the data mapping through the routine interface. The data mappings in routine must comply with the HPF requirement of preserving data distributions by a callee.

ADAPT uses the interprocedural data distribution methods developed at [9], [13],

[15] and [19]. At each call site, the interprocedural analysis provides the mapping of dummy arguments onto actual arguments. This mapping is used to attach the routine alignment graph to the nest alignment graph and transform the mapping declared at routine interface into the nest mapping. If the mappings are compatible, meaning that the number of distributed dimensions for combined graphs is larger than 0, then the scope of both mappings can be combined, otherwise data redistribution at each call site will be necessary.

7. ADAPT Implementation

ADAPT is written in C++ and is based on a few standard C++ classes such as List, Point, Vector and Matrix. Some advanced classes Polynomial (symbolic polynomial), SLForm (system of linear forms) and DGraph (NAS Directed Graph class) have been implemented and widely used in the tool. The burden of Fortran program parsing, analysis and code generation is placed on CAPTools [11].

CAPTools (Computer Aided Parallelization Tools) have been developed in University of Greenwich, UK [11]. CAPTools demonstrated the ability to parse, analyze and parallelize a number of CFD applications, including NPB and ARC3D. As a result of an agreement between the University of Greenwich and NASA Ames Research Center, the CAPTools group provided NAS Parallel Tools Group with an API. This includes a description of internal data structures used by CAPTools, internal program representation (application data base), a number of utilities and a code generator.

ADAPT uses the CAPTools generated database to perform a single pass through the source program. It builds a PCFG for the whole application and a data transfer graph (DTG) for each loop nest. It annotates each arc with the affinity relation between arrays representing the arc ends. The complexity of processing a nest with n arrays is $O(n^4)$ and is dominated by computing the closure of affinity relation. As a result, data alignment directives are generated for each nest. The directives are then lifted bottom up along the arcs of PCFG by creating subroutine interfaces and either merging directives or placing redistribution directives.

8. Experiments with CFD Codes

ADAPT have been applied to NPB to an aerodynamic application ARC3D. A qualitative comparison with the data distributions used in handwritten HPF implementations of NPB [7] working on single structured grid: BT, SP, LU and FT and ARC3D [8] is given in Table 1. All applications except LU use redistribution of data. The redistributions and their locations in the code have been successfully determined by ADAPT (line1). In all applications except SP some distributed arrays are passed as subroutine arguments. ADAPT was able to use the interprocedural information generated by CAPTools to move the distributions across subroutine boundaries and generate HPF subroutine interfaces (lines 2 and 3). For simple dependences between distributed array sections (BT,SP,LU and ARC3D) ADAPT was able to ignore the redistributions and leave the pipelineing to the compiler, line 4. ADAPT was not able to generate redistributions of some boundary data necessary for efficient computations of boundary conditions (BC) in ARC3D, line 5 (note that BC was excluded from the plot on Figure 6). Based on the analysis of index expressions and loop nest indices ADAPT was able to detect and skip iteration loop, line 6, as well to perform qualification of privatizable arrays, line 7. Neither of the considered applications would benefit from cyclic distribution nor ADAPT was enabled to generate cyclic distributions, lines 8 and 9.

TABLE 1. ADAPT (A) versus Manual (M) HPF Data Distribution for Scientific Codes. [+ *uses the feature, - does not use the feature, * depends on compiler support, ✓ automatically generated*]

Benchmark	BT		SP		LU		FT		ARC3D	
DD Features	**M**	**A**	**M**	**A**	**M**	**A**	**M**	**A**	**M**	**A**
1. Redistribution	+	✓	+	✓	-	-	+	✓	+	✓
2. Interprocedural	+	✓	-	-	+	✓	+	✓	+	✓
3. Interfaces	+	✓	-	-	+	✓	+	✓	+	✓
4. Pipeline[a]	*	✓	*	✓	*	✓	-	-	*	✓
5. BC redistribution	-	-	-	-	-	-	-	-	+	-
6. Time loop invariant	+	✓	+	✓	+	✓	+	✓	+	✓
7. Privatization (new)	+	✓	+	✓	+	✓	-	✓	+	✓
8. Block distribution	+	✓	+	✓	+	✓	+	✓	+	✓
9. Cyclic distribution	-	-	-	-	-	-	-	-	-	-

[a] The feature can be used if the compiler is able to support pipelineing

The worst case complexity of $O(n^4)$ for computing the closure of the affinity relation (where n is the maximum number of nodes in the nest DTG) never have been reached, Table 2. The execution time (line 6) was dominated by other factors such as computing of the affinity relations from index expressions and the lifting of the directives along edges of PCFG. The complexity of these operations is proportional to the number of arcs in DTG (line 3) and in PCFG (line 4) respectively. Overall ADAPT execution time was significantly less than the CAPTools (line 5) analysis time.

TABLE 2. ADAPT Performance

Benchmark	BT	SP	LU	FT	ARC3D
1. Number of subroutines	48	33	34	31	33
2. Number of nests	165	51	43	17	82
3. Max size of DTG (nodes,arcs)	(30,381)	(29,148)	(39,480)	(12,16)	(48,201)
4. Size of PCFG (nodes,arcs)	(165,220)	(173,229)	(174,208)	(85,122)	(253,297)
5. CAPTools analysis time (min.)	72	67	26	30	23
6. ADAPT CPU time (sec.)[a]	3	3	14	1	6

[a] The execution time is on 150 MH SGI R5000 machine, including time for code generation and excluding time for creating CAPTools data base.

Finally we have applied some hand editing to the code generated by ADAPT for BT, SP and FT[*] and ARC3D. The compiler was able to compile code, but performance was very poor. Few hand editing steps were necessary to get good code performance. First, pghpf2.4 does not support the REDISTRIBUTION statement and all redistributions were implemented by hand. It included copying distributed arrays to arrays with an alternative distribution and substituting the arrays with alternative distribution instead of original arrays in the scope of the REDISTRIBUTION directive. Second, some linear algebra routines called in BT take array section as an argument. The compiler passed the section by value which was very slow. We performed inlining of the routines by hand significantly iproved performance. The performance of resulted code was comparable with the performance of the handwritten HPF code and with the MPI code, Figure 6.

[*] An inspection of the DTG and PCFG of the FT suggested that one of three 3D complex arrays is redundant. A removing of this array from the benchmark reduced the memory requirements by 30% and slightly improved performance.

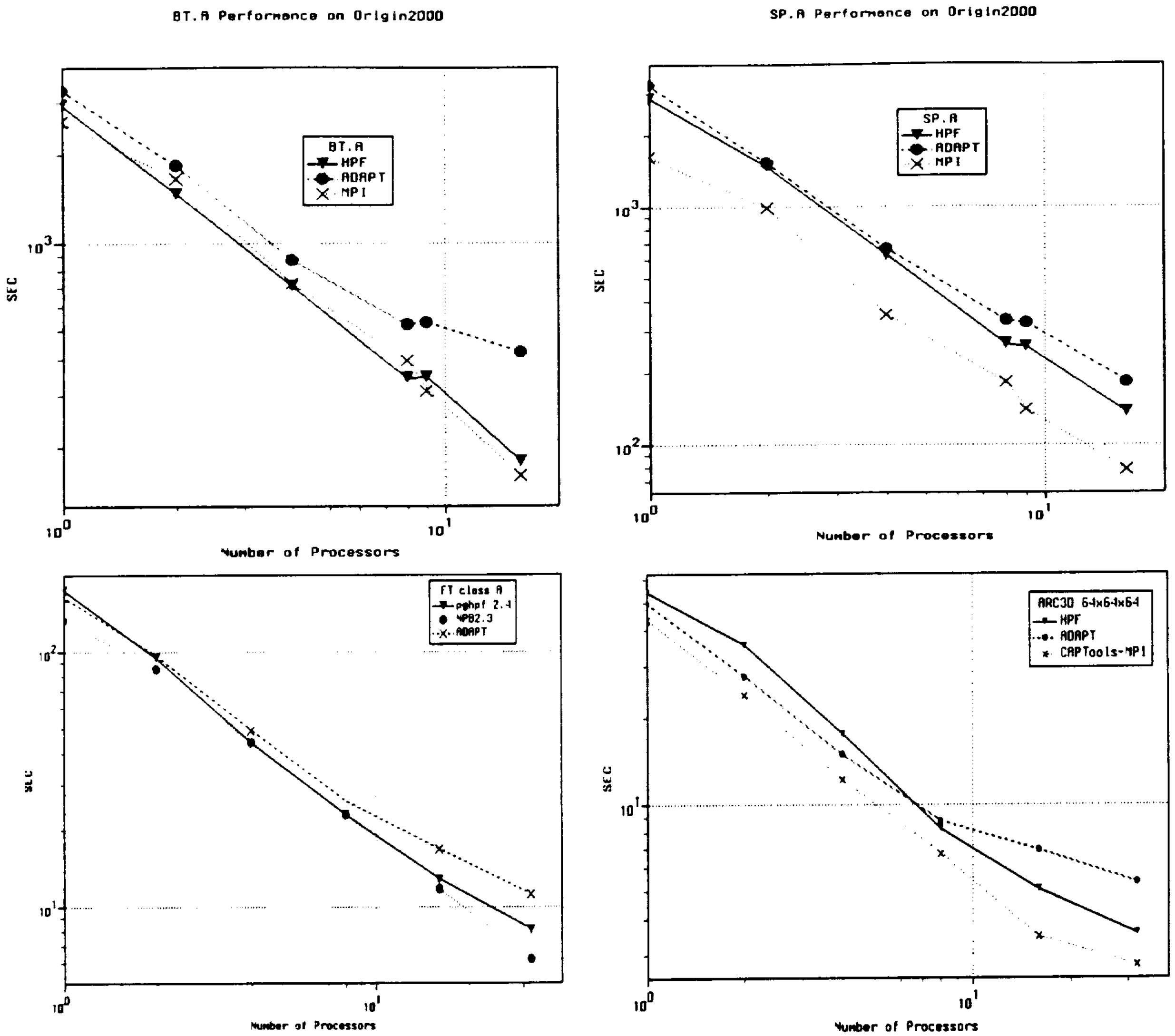


FIGURE 6. Comparison of MPI version (dotted curves), handwritten HPF version (solid curves) and ADAPT generated (dashed curves) versions of BT,SP, FT and ARC3D. The boundary condition subroutine excluded from ARC3D plot since it requires a number of hand tuning steps.

9. Automatic Data Distribution Techniques

A number of methods for automatic data mapping have been designed: [2],[5],[13],[15],[16],[17],[18],[23]. Some of the techniques were developed in a framework of automatically parallelizing compiler, other in the context of parallelizing tools. An extensive survey of data layout methods is given in [15] and a case study of 4 approaches is given in [3]. General requirements to data distribution tools are listed in [21]. We will concentrate on techniques suitable for distribution of data defined on a single or multiple structured grids. Most of the existing tools (with CAPTools an exception) were implemented as a "demonstration of concept" and none of them have demonstrated ability to analyze medium or large size codes and generate an efficient data parallel program.

An approach to data layout based on a decomposition of procedures into phases and finding the best static alignment for each phase was developed by Li and Chen in [16]. The algorithm performs inter-dimensional alignment as a first step and intra-dimensional alignment as the second step. The inter-dimensional alignment is formulated as a partition problem for the *Component Affinity Graph* (CAG). The authors propose a heuristic algorithm to find the best alignment, in general, however, they show that the problem is NP-complete.

Paradigm [18]. The approach is based on the analysis of the communication graph generated by Parafrase-2. The graph nodes are program statements and graph edges are data flows between statements weighted with cost of the communications. The graph is recursively decomposed into a hierarchy of phases by removing a maximal cut on each step. The decomposition of the communication graph stops at the point when a static distribution can not be improved by further decomposition. Then a phase transition graph is built with the edges weighted by the cost of redistributions. A critical path in the graph gives the best sequence of phases and phase transitions. The tool have been successfully applied to 2D FFT and ADI kernels.

SUIF [2],[17],[23]. An algorithm for dynamic data decomposition is given in [2]. It is applicable to an arbitrary sequence of loop nests with loop boundaries and array references described by linear functions. It involves 3 main steps: 1. finding communication free decomposition, 2. if such a decomposition can not be found the algorithm searches for a decomposition with pipelined communications, 3. if such partition can not be found the algorithm applies a heuristic to group the nests to find a partition with pipelined communications within each group and redistributing data between the groups. The algorithm was enhanced in [17] to find partitions minimizing synchronizations.

dHPF [13], [15]. The approach consists of reduction of the data distribution problem to a Boolean optimization problem and applying of a commercial package (CPLEX) for solving it. The reduction proceeds in a number of steps. On the first step the program is partitioned into phases. Then for each phase a CAG is built. The partitions of CAG are candidate data layouts. The optimal layout is a critical path in the data layout graph which nodes are the candidate layouts and edges are possible remappings of the layouts between the phases. The edges are weighted with an empirical estimation of the remapping cost. The resulting optimization problem is then formulated as 0-1 programming problem and solved with aid of CPLEX. The tool was able to generate alignment and distribution statements for ADI kernel and Erlebacher and Tomcatv benchmarks.

CAPTools [6], [11], [12]. CAPTools has an ability to apply block, cyclic and block/cyclic distributions to data defined on structured [6] and on unstructured [11] grids. The distribution requires the user to specify an array and a dimension to be distributed. As soon a distribution have been defined a MPI code implementing "owner computes" rule is generated.

10. Conclusions and Future Work

We described methods for translating data dependence and affinity relations into data mapping directives. These methods have been used for generating data distributions for HPF versions of NPB [7] and ARC3D [8]. Our algorithms for program analysis and derivation of data distributions are efficient three pass algorithms. The majority of algorithms have linear complexity with exception of some graph algorithms having complexity $O(n^4)$ (n is the number of variables used in the program nests) in the worst case.

We implemented the methods in an Automatic Data Alignment and Placement Tool (ADAPT). Initial comparison shows that the data mappings generated by ADAPT are very close to the data mapping directives used in hand written HPF version of NPB and ARC3D. We aim ADAPT at real CFD applications such as OVERFLOW [10]. Also we are considering to use data affinity relations for restructuring serial programs to improve cache utilization.

Acknowledgments: The authors wish to acknowledge Haoqiang H. Jin and Rob F. Van Der Wijngaart for useful discussions. The work presented in the paper is supported under NASA High Performance Computing and Communication Program and by NASA Contract No. NAS2-14303 with MRJ Technology Solutions.

References

[1] A.V. Aho, R. Sethi and J.D. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley Publ., Reading MA, 1988.

[2] J.M. Anderson, M.S. Lam. *Global Optimizations for Parallelism and Locality on Scalable Parallel Machines.* In Proceedings of SIGPLAN'93 Conference on Programming Languages Design and Implementation (PLDI), Albuquerque, NM, June 23-25, 1993.

[3] E. Ayguade, J. Garcia, U. Kremer. *Tools and Techniques for Automatic Data Layout: A Case Study.* Parallel Computing, v. 24 (1998) pp. 557-578.

[4] D. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo, M. Yarrow. *The NAS Parallel Benchmarks 2.0.* Report NAS-95-020, Dec. 1995. http://science.nas.nasa.gov/Software/NPB.

[5] I. Couvertier-Reyes. *Automatic Data and Computation Mapping for Distributed Memory Machines.* Ph.D dissertation, Louisiana State University, 1996.

[6] S.P. Johnson,C.S. Ierotheou, M. Cross. *Automatic Parallel Code Generation on Distributed Memory Systems.* Parallel Computing, V. 22(1996), pp. 227-258.

[7] M. Frumkin, H. Jin, J. Yan. *Implementation of NAS Parallel Benchmarks in High Performance Fortran.* CDROM version of IPPS/SPDP 1999 Proceedings, April 12-16, 1999, San Juan, Puerto Rico, 10 p.

[8] M. Frumkin, J. Yan. *HPF Implementation of ARC3D.* Frontiers'99, February 21-25, 1999, Annapolis, pp. 81-88.

[9] M.W. Hall, S. Hiranandani, K.Kennedy, C.-W. Tseng. *Interprocedural Compilation of Fortran D for MIMD Distributed-Memory Machines.* Supercomputing '92, pp. 522-534, Minneapolis, MN, Nov. 1992.

[10] D.C. Jespersen. *Parallelism and Overflow.* NAS Technical Report NAS-98-013, October 1998.

[11] S.P. Johnson, K. McManus, C.S. Ierotheou, M. Cross. *Semi-automatic Parallelization of Unstructured Mesh Code Using Domain Decomposition.* Submitted to Parallel Computing.

[12] S.P. Johnson, M. Cross, M.G. Everett. *Exploitation of Symbolic Information in Interprocedural Dependence Analysis.* Parallel Computing, v. 22 (1966) pp.197-226.

[13] K. Kennedy, U. Kremer. *Automatic Data Layout for High Performance Fortran.* Supercomputing '95, San Diego, CA, December 1995.

[14] C.H. Koelbel, D.B. Loverman, R. Shreiber, G.L. Steele Jr., M.E. Zosel. *The High Performance Fortran Handbook.* MIT Press, 1994.

[15] U. Kremer. *Automatic Data Layout for Distributed Memory Machines*, PhD. thesis, Rice Univ., October 1995, CRPC-TR95-599-S.

[16] J. Li, M. Chen. *The Data Alignment Phase in Compiling Programs for Distributed-Memory Machines.* J. of Parallel and Distr. Computing, V. 13 n. 2, August 1991, pp. 213-221.

[17] A.W. Lim, M.S. Lam. *Maximizing Parallelism and Minimizing Synchronization with Affine Partitions.* Parallel Computing, v. 24 (1998), pp. 445-475.

[18] D.J. Palermo, P. Banerjee. *Automatic Selection of Dynamic Data Partitioning Schemes for Distributed-Memory Multicomputers.* In Proceedings of the 8th Workshop on Languages and Compilers for Parallel Computing, Columbus, OH, August 1995, LNCS, v. 1033, pp. 392-406, Springer Verlag, 1996.

[19] D.J. Palermo, E.W. Hodges IV, P. Banerjee. *Interprocedural Array Redistribution Data-Flow Analysis.* 9th Workshop on Languages and Compilers for Parallel Computing, San Jose, CA, August 8-10, 1996.

[20] T.H. Pulliam, D.S. Chaussee. *A Diagonal Form of an Implicit Approximate Factorization Algorithm.* Journal of Computational Physics, Vol. 29, p.1037, 1975.

[21] J.-L. Pazat. *Tools for High Performance Fortran: A Survey.* LNCS, v. 1132, 1996, pp. 134-158.

[22] A. Schrijver. *Theory of Linear and Integer Programming.* J. Wiley & Sons, 1998, pp. 256-259.

[23] M.E. Wolf, M.S. Lam. *A Data Locality Optimizing Algorithm.* In Proceedings of the SIGPLAN'91 Conference on Programming Language Design and Implementation, June 1991, pp. 30-44.