Segmented Alignment: An Enhanced Model to Align Data Parallel Programs of HPF

Gwan-Hwan Hwang
Department of Information and Computer Education,
National Taiwan Normal University, Taipei, Taiwan

Cheng-Wei Chen Jenq Kuen Lee
Department of Computer Science,
National Tsing-Hua University, Hsinchu, Taiwan

Roy Dz-Ching Ju
Microprocessor Research Labs,
Intel Corporation, Santa Clara, CA 95052, USA

Abstract

In this paper, we propose a new automatic data alignment model called segmented alignment. The conventional data alignment model, such as that used in High-Performance Fortran (HPF), aligns arrays with the whole index domain. The principle of our proposed segmented alignment is to allow alignment relations within delimited index domains. We first provide motivating examples to illustrate how code fragments of HPF with EOSHIFT or CSHIFT operations or produced by synthesis operations can benefit from our enhanced alignment scheme. Second, we show that this new model can be implemented in HPF-like languages by adding WHEN and IN constructs to them. In addition, we show that the new proposed schemes for WHEN and IN constructs can be emulated using standard HPF syntax. Finally, we address issues related to automatic data alignment for the new proposed model, and present an algorithm to automatically align programs using our segmented alignment scheme. Since the optimal algorithm to do this is NP-hard, a practical heuristic is also given. Our experiments were performed on a DEC Alpha Farm with HPF environments. Our experiments confirm our theory that our proposed alignment scheme can significantly enhance not only the performance of HPF code fragments with EOSHIFT or CSHIFT operations, but also that of codes produced by synthesis operations.

Keywords: Automatic Array Alignment, Segmented Alignment, HPF Compilers, Distributed-Memory Machines, Compiler Optimization, Parallelizing Compiler

1 Introduction

Fortran 90D and HPF (High-Performance Fortran) languages [19] provide distributed arrays to support a global name space on distributed-memory architectures. In these data parallel languages, programmers can specify the placement of distributed array data among processors, and the compiler can then take distribution information (i.e., how data are distributed among processors) and generate the communication codes [21, 18, 5] for programs to emulate the shared memory space on distributed-memory architectures. When the alignment and distribution are not specified by programmers, the compiler will need to find a data placement (e.g., array alignment) - according to a given sequence of FORALL loops and array operations in data parallel languages - such that the data communications are minimized when the programs are actually executed on distributed-memory architectures.

The problem with conventional alignment schemes in HPF programs is that they align arrays with the whole index domain. In this paper, we propose a new data alignment model called *segmented alignment*. The principle of our proposed model is

[1]G. H. Hwang's work was supported in part by NSC-89-2218-E-260-016 and MOE research excellent project under grant no. 89-E-FA04-1-4 of Taiwan. C. W. Chen and J. K. Lee's work was supported in part by NSC-90-2218-E-007-042, NSC-90-2213-E-007-074, NSC-90-2213-E-007-075, MOE research excellent project under grant no. 89-E-FA04-1-4, and MOEA research project under grant no. 91-EC-17-A-03-S1-0002 of Taiwan.

to allow alignment relations within delimited index domains. Here we first provide motivating examples to illustrate how code fragments of HPF with EOSHIFT or CSHIFT operations or generated from synthesis operations [13] can benefit from our enhanced alignment scheme. In our example, it is impossible to derive a communication-free alignment relation using the conventional HPF alignment model, but we can derive communication-free codes with our proposed segmented alignment. Second, we show that this new model can be supported in HPF-like languages by adding WHEN and IN constructs to them. The proposed schemes for WHEN and IN constructs can also be emulated using standard HPF syntax. Finally, we address issues related to automatic data alignments for the new proposed model. Related work on automatically selecting data distributions can be found in [3, 21, 8, 17, 10, 7, 4]. We present a new algorithm to perform automatic alignment using our segmented alignment scheme. The optimal algorithm to perform automatic alignment for this case is NP-hard, and so a practical heuristic is also given. Our experiments were carried out on a DEC Alpha Farm with HPF environments. Our proposed alignment scheme can be used in two reasons: (i) it can significantly enhance the performance of HPF code fragments containing EOSHIFT or CSHIFT operations, and (ii) it is particularly useful for optimizing the codes produced by synthesis operations [13]. Experiments performed on an eight-node DEC Alpha Farm show that the automatic alignment process using the segmented alignment concept can significantly outperform the conventional alignment process that does not employ this mechanism.

Previously, research work has investigated automatic data alignment and distribution. Knobe, Lucas, and Steele [17] were the first to investigate the relationships among arrays in a data parallel program in order to construct a preference graph. They then proposed a heuristic algorithm for selections based on preference graphs. Li and Chen [21, 22] considered axis alignment by developing a heuristic to reduce it to weighted bipartite graph matching. Kennedy et al. [16] determined data layouts automatically in distributed-memory environments by using 0-1 integer programming [4]. Gupta et al. [11] extended the work of Li and Chen by presenting a framework based on weighted graphs. Chatterjee, Gilbert, and Schreiber [7] presented a framework for the automatic determination of array alignments. They developed a heuristic algorithm to solve axis, stride, and offset alignment on their computation-directed acyclic graphs. Philippsen [24, 25] presented a heuristic algorithm for automatic alignment of array data and processes at the time of compilation, also based on preference graphs. Our proposed alignment model is particular useful for HPF code fragments with EOSHIFT or CSHIFT operations, or those produced by synthesis operations [13]. The work related to array operation synthesis can be found in [6, 9, 13, 15]. Array synthesis was useful for optimizing consecutive array operations.

The remainder of this paper is organized as follows. Section 2 provides motivating examples to explain the segmented alignment concept this paper proposes. Section 3 then describes a cost model, and Section 4 describes the automatic alignment scheme using our proposed segmented alignment scheme. Section 5 then describes a heuristic algorithm for the automatic alignment process, and Section 6 gives the experimental results. Finally, Section 7 concludes the paper.

2 Motivating Examples

Consider the following Fortran 90 code fragment:

HPF Code Fragment 1

```
        REAL A(N,N), B(N,N), C(N,N)
                     ⋮
        DO ITER=1, Number_Iteration
!Loop 1
           FORALL (I=1:N/2,J=1:N/2)
              A(I,J)=B(J+N/2,I+N/2)+C(I+N/2,J)
           END FORALL
!Loop 2
```

```
            FORALL (I=1:$\frac{N}{2}$,J=$\frac{N}{2}+1$:N)
               A(I,J)=-6.6+C(I+$\frac{N}{2}$,J)
            END FORALL
!Loop 3
            FORALL (I=$\frac{N}{2}+1$:N,J=1:$\frac{N}{2}$)
               A(I,J)=B(J+$\frac{N}{2}$,I-$\frac{N}{2}$)+C(I-$\frac{N}{2}$,J)
            END FORALL
!Loop 4
            FORALL (I=$\frac{N}{2}+1$:N,J=$\frac{N}{2}+1$:N)
               A(I,J)=-6.6+C(I-$\frac{N}{2}$,J)
            END FORALL
         ENDDO
                   ⋮
```

When the code fragment above runs on distributed-memory environments, an automatic scheme for alignment and distribution assignments can be used to find an alignment relationship to align arrays such that data communications will be minimized. Conventionally, automatic alignment schemes [3, 21, 8, 17, 10, 7, 4] will align arrays with the whole index domain. If we use conventional alignment methods to specify the data distribution of arrays A, B, and C in *Code Fragment 1*, it is impossible to construct a communication-free code for the four parallel loops. For example, in Loops 1 and 3, A(I,J), B(J+$\frac{N}{2}$,I+$\frac{N}{2}$) and C(I+$\frac{N}{2}$,J) should be aligned to ensure that Loop 1 is communication free. However, to ensure that Loop 3 is communication free, A(I,J), B(J+$\frac{N}{2}$,I-$\frac{N}{2}$) and C(I-$\frac{N}{2}$,J) should be aligned. Since the alignment functions can only be defined in a continuous style, there exists an inherent conflict for the requirement that Loops 1 and 3 are communication free. To enable these types of codes to be incorporated into an automatic alignment framework, we propose a new concept - called segmented alignment - to solve the above dilemma. The basic principle of segmented alignment is to use alignment relations within delimited index domains.

Before we present the details of our proposed segmented alignment scheme in HPF-like constructs, we would like to further explain our motivating example in *Code Fragment 1*. *Code Fragment 1* is actually the target code obtained after applying a compiler optimization called ”array operation synthesis” [13, 15] on *Code Fragment 2* below.

HPF Code Fragment 2

```
         REAL A(N,N), B(N,N), C(N,N)
                   ⋮
         DO I=1, Number_Iteration
            A=CSHIFT((TRANSPOSE(EOSHIFT(B,$\frac{N}{2}$,-6.6,1))+C),$\frac{N}{2}$,1)
         ENDDO
                   ⋮
```

Array operation synthesis is used to synthesize consecutive array operations in Fortran 90 to reduce data movements and synchronization overheads. The synthesis scheme can cope with compositions of extensive Fortran 90 array constructs, such as RESHAPE, SPREAD, EOSHIFT, TRANSPOSE, CSHIFT, and MERGE functions, array sections, array reduction functions, and WHERE and ELSE-WHERE constructs.

The principle of segmented alignment is to allow alignment relations within delimited index domains. We add WHEN and IN keywords to target codes for HPF-style languages. For example, line 3 of *Code Fragment 3* aligns array element A(I,J) with a template array TEMP1(I,J) when (I,J) is in the index domain (1:$\frac{N}{2}$,1:$\frac{N}{2}$). Template arrays TEMP1, TEMP2, TEMP3, and TEMP4 are used to align the array references in Loops 1-4, respectively.

HPF Code Fragment 3

```
1              REAL A(N,N), B(N,N), C(N,N)
2       !HPF$   TEMPLATE TEMP1($\frac{N}{2}$,$\frac{N}{2}$)
3       !HPF$   ALIGN A(I,J) WITH TEMP1(I,J) WHEN (I,J) IN (1:$\frac{N}{2}$,1:$\frac{N}{2}$)
4       !HPF$   ALIGN B(I,J) WITH TEMP1(J-$\frac{N}{2}$,I-$\frac{N}{2}$) WHEN (I,J) IN ($\frac{N}{2}+1$:N,$\frac{N}{2}+1$:N)
5       !HPF$   ALIGN C(I,J) WITH TEMP1(I-$\frac{N}{2}$,J) WHEN (I,J) IN ($\frac{N}{2}+1$:N,1:$\frac{N}{2}$)
6       !HPF$   TEMPLATE TEMP2($\frac{N}{2}$,$\frac{N}{2}$)
7       !HPF$   ALIGN A(I,J) WITH TEMP2(I,J) WHEN (I,J) IN (1:$\frac{N}{2}$,$\frac{N}{2}+1$:N)
8       !HPF$   ALIGN C(I,J) WITH TEMP2(I-$\frac{N}{2}$,J) WHEN (I,J) IN ($\frac{N}{2}+1$:N,$\frac{N}{2}+1$:N)
9       !HPF$   TEMPLATE TEMP3($\frac{N}{2}$,$\frac{N}{2}$)
10      !HPF$   ALIGN A(I,J) WITH TEMP3(I,J) WHEN (I,J) IN ($\frac{N}{2}+1$:N,1:$\frac{N}{2}$)
11      !HPF$   ALIGN B(I,J) WITH TEMP3(J+$\frac{N}{2}$,I-$\frac{N}{2}$) WHEN (I,J) IN (1:$\frac{N}{2}$,$\frac{N}{2}+1$:N)
12      !HPF$   ALIGN C(I,J) WITH TEMP3(I+$\frac{N}{2}$,J) WHEN (I,J) IN (1:$\frac{N}{2}$,1:$\frac{N}{2}$)
13      !HPF$   TEMPLATE TEMP4($\frac{N}{2}$,$\frac{N}{2}$)
14      !HPF$   ALIGN A(I,J) WITH TEMP4(I,J) WHEN (I,J) IN ($\frac{N}{2}+1$:N,$\frac{N}{2}+1$:N)
15      !HPF$   ALIGN C(I,J) WITH TEMP4(I+$\frac{N}{2}$,J) WHEN (I,J) IN (1:$\frac{N}{2}$,$\frac{N}{2}+1$:N)
16      !HPF$   DISTRIBUTE TEMP1(BLOCK,BLOCK),TEMP2(BLOCK,BLOCK),TEMP3(BLOCK,BLOCK),TEMP4(BLOCK,BLOCK)
                        ⋮
17              DO ITER=1, Number_Iteration
18      !Loop 1
19                 FORALL (I=1:$\frac{N}{2}$,J=1:$\frac{N}{2}$)
20                    A(I,J)=B(J+$\frac{N}{2}$,I+$\frac{N}{2}$)+C(I+$\frac{N}{2}$,J)
21                 END FORALL
22      !Loop 2
23                 FORALL (I=1:$\frac{N}{2}$,J=$\frac{N}{2}+1$:N)
24                    A(I,J)=-6.6+C(I+$\frac{N}{2}$,J)
25                 END FORALL
26      !Loop 3
27                 FORALL (I=$\frac{N}{2}+1$:N,J=1:$\frac{N}{2}$)
28                    A(I,J)=B(J+$\frac{N}{2}$,I-$\frac{N}{2}$)+C(I-$\frac{N}{2}$,J)
29                 END FORALL
30      !Loop 4
31                 FORALL (I=$\frac{N}{2}+1$:N,J=$\frac{N}{2}+1$:N)
32                    A(I,J)=-6.6+C(I-$\frac{N}{2}$,J)
33                 END FORALL
34              ENDDO
                        ⋮
```

Lines 2-5 align A(I,J), B(J+$\frac{N}{2}$,I+$\frac{N}{2}$), and C(I+$\frac{N}{2}$,J) in index domain (I=1:$\frac{N}{2}$,J=1:$\frac{N}{2}$) of Loop 1; similarly, lines 6-8 align A(I,J) and C(I+$\frac{N}{2}$,J) in index domain (I=1:$\frac{N}{2}$,J=$\frac{N}{2}+1$:N) of Loop 2; lines 9-12 align A(I,J), B(J+$\frac{N}{2}$,I-$\frac{N}{2}$), and C(I-$\frac{N}{2}$,J) in index domain (I=$\frac{N}{2}+1$:N,J=1:$\frac{N}{2}$) of Loop 3; and lines 13-15 align A(I,J), and C(I-$\frac{N}{2}$,J) in index domain (I=$\frac{N}{2}+1$:N,J=$\frac{N}{2}+1$:N) of Loop 4. Since all of the array references in Loops 1-4 (lines 18-33) are aligned, these loops are communication free.

The segmented alignment concept proposed in this work can be emulated using existing HPF codes. Throughout the paper, we will still use the segmented alignment concept and WHEN and IN keywords to represent the target programs, as this provides a better abstraction. *Appendix A* gives an emulated HPF code for *Code Fragment 3* by splitting arrays into subarrays.

3 Communication Cost Model

In HPF, data movements from source arrays to target arrays may involve communication between remote processors. The amount of remote communication depends on the alignment and distribution of the source and target arrays. An often-used communication-cost model for alignment analysis is to model the communication cost as the amount of data moved if the target array and the source array are misaligned. However, our cost model is developed based on modeling the communication cost as the amount of misaligned data multiplied by a distance function. Before we describe our cost model, we first give an auxiliary definition below.

Definition 1 A **segmentation descriptor** represents a set, and it is recursively defined as follows:

$$(1)\phi(\ /f_1(i_1,i_2,\ \dots\ ,i_n),\dots\ ,\ f_m(i_1,i_2,\ \dots\ ,i_n)/\ ,\ /l_1{:}u_1{:}s_1, l_2{:}u_2{:}s_2,\dots,l_m{:}u_m{:}s_m/) =$$
$$\{\ (i_1,i_2,\ \dots\ ,i_n)\ |\ \forall\ k,\ 1 \le k \le n,\ l_k \le f_k(i_1,i_2,\dots,i_n) \le u_k \text{ and }$$
$$f_k(i_1,i_2,\dots,i_n) = l_k + s_k * I, \text{ where } I \text{ is a non-negative integer. } \}$$

A regular section specifier[1] has the form $l_k{:}u_k{:}s_k$, where the l_k, u_k, and s_k indicate the lower bound, upper bound, and stride of $f_k(i_1,i_2, \dots ,i_n)$, respectively. We may omit the stride if it is equal to one. We say a segmentation descriptor is **simple** if it is of the above form.

(2)If α_1 and α_2 are two segmentation descriptors, then $\alpha_1 \wedge \alpha_2$ is also a segmentation descriptor but it is not simple: $\alpha_1 \wedge \alpha_2 = \{(i_1,i_2, \dots ,i_n) \mid (i_1,i_2, \dots ,i_n) \in \alpha_1$ and $(i_1,i_2, \dots ,i_n) \in \alpha_2\ \}$.

We now have our cost model as defined below.

Definition 2 Suppose we want to move array $B[\ g_1(i_1,\dots,i_k), g_2(i_1,\dots,i_k), \dots, g_m(i_1,\dots,i_k)\ ]$ to $A[\ f_1(i_1,\dots,i_k),\ f_2(i_1,\dots,i_k), \dots,\ f_n(i_1,\dots,i_k)\ ]$ within the index domain defined by the segmentation descriptor $\gamma = \phi(/i_1, \dots, i_k/, /l_1 : u_1 : s_1, \dots, l_k : u_k : s_k /)$. That is, we want to execute the following section movement:

$$\forall (i_1,\dots,i_k) \in \gamma,\ A[f_1(i_1,\dots,i_k),\dots,f_n(i_1,\dots,i_k)] = B[g_1(i_1,\dots,i_k),\dots,g_m(i_1,\dots,i_k)].$$

We define the communication cost of the above section movement as

$$\sum_{(i_1,\dots,i_k)\in\gamma} \mathcal{D}(A[f_1(i_1,\dots,i_k),\dots,f_n(i_1,\dots,i_k)],B[g_1(i_1,\dots,i_k),\dots,g_m(i_1,\dots,i_k)])$$

where $\mathcal{D}$ is called the *distance function*.

The distance function corresponds to the cost of moving one unit of data from the source array to the target array. To estimate the communication cost of data movement between arrays more precisely, one should consider both the alignment and the distribution of the source and target arrays when determining the distance function $\mathcal{D}$. For simplicity, we define the distance function in *Definition 4* (below). Although the model simplifies real systems, our framework presented in *Sections 4.3* and *5* is completely extensible with a more accurate definition of the distance function $\mathcal{D}$.

Before we give our distance function, we will first introduce the auxiliary definition for a concept called reference location. HPF provides compiler directives that programmers use to specify the data layout of arrays . For two-level mapping, arrays are aligned to an auxiliary Cartesian grid called a template [19], which can be used as an abstract alignment target that may then be distributed onto the parallel machine. Even if the programmer does not specify a global template array, a simple transformation can be used to derive a global template array in a program. The general form of the alignment relation in HPF is

!HPF\$ ALIGN $A(i_1,i_2,\dots,i_p)$ with $T(f_1(i_1,i_2,\dots,i_p),\ f_2(i_1,i_2,\dots,i_p),\dots,f_q(i_1,i_2,\dots,i_p))$

where $f_k(i_1,i_2,\dots,i_p)$, $k = 1,\dots,q$, is either a "$*$" (broadcasting to the whole dimension) or $C_0 * i_r + C_1$, where $1 \le r \le p$, and C_0 and C_1 are integers.

For the segmented alignment, we extend the original alignment compiler directive of HPF by adding WHEN and IN keywords to specify the alignment relation in a delimited index domain: array $A(i_1,\dots,i_p)$ will be aligned to $T(f_1^k(i_1,\dots,i_p), \dots, f_q^k(i_1,\dots,i_p))$ only if $(i_1,\dots,i_p) \in (R_1^k,\dots,R_p^k)$. The ordinary alignment compiler directive of HPF can be treated as a special case of the segmented alignment defined in the paper, i.e., the alignment relation definition is valid in the whole index domain of the specified array:

!HPF$ ALIGN $A(i_1,\ldots,i_p)$ with $T\,(\,f_1^1(i_1,\ldots,i_p),\,\ldots,\,f_q^1(i_1,\ldots,i_p))$ WHEN $(i_1,\ldots,i_p)$ IN $(R_1^1,\ldots,R_p^1)$
!HPF$ ALIGN $A(i_1,\ldots,i_p)$ with $T\,(\,f_1^2(i_1,\ldots,i_p),\,\ldots,\,f_q^2(i_1,\ldots,i_p))$ WHEN $(i_1,\ldots,i_p)$ IN $(R_1^2,\ldots,R_p^2)$
...
!HPF$ ALIGN $A(i_1,\ldots,i_p)$ with $T\,(\,f_1^n(i_1,\ldots,i_p),\,\ldots,\,f_q^n(i_1,\ldots,i_p))$ WHEN $(i_1,\ldots,i_p)$ IN $(R_1^n,\ldots,R_p^n)$

where R_i^j is a regular section specifier defined in [1].

If array A is aligned with a template T in the preceding manner, then we use the following definition to describe each reference of array A:

Definition 3 Assume array A is aligned with template T according to the above segmented alignment relation. A **reference location** of $\mathtt{A}(g_1(i_1,i_2,\ \ldots\ ,i_p),\ \ldots,\ g_p(i_1,i_2,\ldots,i_p))$ with respect to template T is then

$$\begin{aligned}\mathtt{T}(\ & \mathtt{f}_1^{\mathtt{k}}(\mathtt{g}_1(\mathtt{i}_1,\mathtt{i}_2,\ldots,\mathtt{i}_\mathtt{p}),\mathtt{g}_2(\mathtt{i}_1,\mathtt{i}_2,\ldots,\mathtt{i}_\mathtt{p}),\ldots,\mathtt{g}_\mathtt{p}(\mathtt{i}_1,\mathtt{i}_2,\ldots,\mathtt{i}_\mathtt{p})),\\ & \mathtt{f}_2^{\mathtt{k}}(\mathtt{g}_1(\mathtt{i}_1,\mathtt{i}_2,\ldots,\mathtt{i}_\mathtt{p}),\mathtt{g}_2(\mathtt{i}_1,\mathtt{i}_2,\ldots,\mathtt{i}_\mathtt{p}),\ldots,\mathtt{g}_\mathtt{p}(\mathtt{i}_1,\mathtt{i}_2,\ldots,\mathtt{i}_\mathtt{p})),\\ & \cdots,\\ & \mathtt{f}_\mathtt{q}^{\mathtt{k}}(\mathtt{g}_1(\mathtt{i}_1,\mathtt{i}_2,\ldots,\mathtt{i}_\mathtt{p}),\mathtt{g}_2(\mathtt{i}_1,\mathtt{i}_2,\ldots,\mathtt{i}_\mathtt{p}),\ldots,\mathtt{g}_\mathtt{p}(\mathtt{i}_1,\mathtt{i}_2,\ldots,\mathtt{i}_\mathtt{p})\ ))\end{aligned}$$

if $(i_1,i_2,\ldots,i_p) \in \phi(/g_1(i_1,\ldots,i_p),\ldots,g_p(i_1,\ldots,i_p)/,\ /R_1^k,\ldots,R_p^k/)$.

For example, arrays A and B are aligned with template T according to the following relationships:

```
!HPF$ ALIGN A(i,j) WITH T(2*i-5,*,3*j) WHEN (i,j) in (1:1000,1:1000)
!HPF$ ALIGN B(i,j) WITH T(i+3,*,2*j) WHEN (i,j) in (1:1000,1:1000)
```

The reference location of A$(2*i,2*j)$ with respect to template T is T$(2*(2*i)-5,*,3*(2*j))$=T$(4*i-5,*,6*j)$ if $(i,j) \in \phi(/2*i,2*j/,/1{:}1000,1{:}1000/)$. Similarly, the reference location of B$(4*i-8,3*j)$ with respect to template T is T$((4*i-8)+3,*,2*(3*j))$=T$(4*i-5,*,6*j)$ if $(i,j) \in \phi(/4*i-8,3*j/,/1{:}1000,1{:}1000/)$. The reference locations of A$(2*i,2*j)$ and B$(4*i-8,3*j)$ with respect to template T are identical if $(i,j) \in \phi(/2*i,2*j/,/1{:}1000,1{:}1000/) \cap \phi(/4*i-8,3*j/,/1{:}1000,1{:}1000/)$.

If two array references are with the same reference location with respect to the same template array, then the two references are located in the same processor. We now have our distance function defined below.

Definition 4 $\mathcal{D}$ is the distance function defined as follows:
$\mathcal{D}(A(f_1(i_1,\ldots,i_k),\ldots,f_n(i_1,\ldots,i_k)),B(g_1(i_1,\ldots,i_k),\ldots,g_m(i_1,\ldots,i_k)))=1$ if the reference locations of $A(f_1(i_1,\ldots,i_k),\ldots,f_n(i_1,\ldots,i_k))$ and $B(g_1(i_1,\ldots,i_k),\ldots,g_m(i_1,\ldots,i_k))$ are not identical with respect to the same template[1]; $\mathcal{D}(A(f_1(i_1,\ldots,i_k),\ldots,f_n(i_1,\ldots,i_k)),B(g_1(i_1,\ldots,i_k),\ldots,g_m(i_1,\ldots,i_k)))=0$ otherwise.

4 Automatic Segmented Alignments for Loops

In this section, we address the issues associated with automatic data alignments for our proposed model, the segmented alignment model. The related work in solving the automatic data alignment problem can be seen in [3, 21, 8, 17, 10, 7, 4]. Conventional data alignment models align arrays with the whole index domain. Here we present new algorithms to enable automated alignment problem using our segmented alignment scheme.

[1]The calculation of the reference locations of arrays A and B should properly choose the valid alignment relations according to the referred index domains of these arrays.

4.1 Segmented Alignment of Communication-Free Codes for Parallel Loops

To explain the techniques for generating communication-free codes for a sequence of FORALL loops with segmented alignments, we first provide examples of single-statement parallel loops. Let us consider *Code Fragment 4* below.

HPF Code Fragment 4

```
                  ⋮
        FORALL ($I = L : U : S$)
           $A_1(\alpha_1 * I + \beta_1) = \mathcal{F}(A_2(\alpha_2 * I + \beta_2), A_3(\alpha_3 * I + \beta_3), \ldots, A_n(\alpha_n * I + \beta_n))$
        END FORALL

        FORALL ($I = L' : U' : S'$)
           $B_1(\alpha'_1 * I + \beta'_1) = \mathcal{F}(B_2(\alpha'_2 * I + \beta'_2), B_3(\alpha'_3 * I + \beta'_3), \ldots, B_m(\alpha'_m * I + \beta'_m))$
        END FORALL
                  ⋮
```

Assume that arrays A_1, A_2, …, A_n, B_1, B_2, …, B_m, are different arrays[2]. Based on the owner-computes rule, if the *reference locations* of arrays A_1, A_2, …, A_n are all the same, there will be no remote access (i.e., communication) required in the first parallel loop. Similarly, if the *reference locations* of arrays B_1, B_2, …, B_m are all the same, there will be no remote access (communication) required in the second parallel loop. The defined ranges of the index domain can be easily derived according to the array subscript and loop index range. From the first loop in *Code Fragment 4*, we can see that the referenced section of array A_k is $A_k(\alpha_k * L + \beta_k : \alpha_k * U + \beta_k : \alpha_k * S)$. Similarly, the referenced section of array B_k is $B_k(\alpha'_k * L' + \beta'_k : \alpha'_k * U' + \beta'_k : \alpha'_k * S')$. The segmented alignment involves defining an alignment relation in a specified array section. Therefore, we can use segmented alignment to specify the data layout of these arrays as follows.

HPF Code Fragment 5

```
!HPF$ TEMPLATE TEMP(X1:X2)
!HPF$ ALIGN $A_1$(I) WITH TEMP($\frac{LCM(\alpha_1,\alpha_2,\ldots,\alpha_n)}{\alpha_1} * I$) WHEN ($I$) IN ($\alpha_1 * L + \beta_1 : \alpha_1 * U + \beta_1 : \alpha_1 * S$)
!HPF$ ALIGN $A_2$(I) WITH TEMP($\frac{LCM(\alpha_1,\ldots,\alpha_n)}{\alpha_2} * I + \beta_1 * \frac{LCM(\alpha_1,\ldots,\alpha_n)}{\alpha_1} - \beta_2 * \frac{LCM(\alpha_1,\ldots,\alpha_n)}{\alpha_2}$) WHEN ($I$) IN
      ($\alpha_2 * L + \beta_2 : \alpha_2 * U + \beta_2 : \alpha_2 * S$)
                   …
!HPF$ ALIGN $A_n$(I) WITH TEMP($\frac{LCM(\alpha_1,\ldots,\alpha_n)}{\alpha_n} * I + \beta_1 * \frac{LCM(\alpha_1,\ldots,\alpha_n)}{\alpha_1} - \beta_n * \frac{LCM(\alpha_1,\ldots,\alpha_n)}{\alpha_n}$) WHEN ($I$) IN
      ($\alpha_n * L + \beta_n : \alpha_n * U + \beta_n : \alpha_n * S$)
!HPF$ ALIGN $B_1$(I) WITH TEMP($\frac{LCM(\alpha'_1,\ldots,\alpha'_n)}{\alpha'_1} * I$) WHEN ($I$) IN ($\alpha'_1 * L' + \beta'_1 : \alpha'_1 * U' + \beta'_1 : \alpha'_1 * S'$)
!HPF$ ALIGN $B_2$(I) WITH TEMP($\frac{LCM(\alpha'_1,\ldots,\alpha'_n)}{\alpha'_2} * I + \beta'_1 * \frac{LCM(\alpha'_1,\ldots,\alpha'_n)}{\alpha'_1} - \beta'_2 * \frac{LCM(\alpha'_1,\ldots,\alpha'_n)}{\alpha'_2}$) WHEN ($I$) IN
      ($\alpha'_2 * L' + \beta'_2 : \alpha'_2 * U' + \beta'_2 : \alpha'_2 * S'$)
                   …
!HPF$ ALIGN $B_m$(I) WITH TEMP($\frac{LCM(\alpha'_1,\ldots,\alpha'_n)}{\alpha'_n} * I + \beta'_1 * \frac{LCM(\alpha'_1,\ldots,\alpha'_m)}{\alpha'_1} - \beta'_m * \frac{LCM(\alpha'_1,\ldots,\alpha'_m)}{\alpha'_m}$) WHEN ($I$) IN
      ($\alpha'_m * L + \beta'_m : \alpha'_m * U' + \beta'_m : \alpha'_m * S'$)
                   ⋮
      FORALL ($I = L : U : S$)
         $A_1(\alpha_1 * I + \beta_1) = \mathcal{F}(A_2(\alpha_2 * I + \beta_2), A_3(\alpha_3 * I + \beta_3), \ldots, A_n(\alpha_n * I + \beta_n))$
      END FORALL
      FORALL ($I = L' : U' : S'$)
         $B_1(\alpha'_1 * I + \beta'_1) = \mathcal{F}(B_2(\alpha'_2 * I + \beta'_2), B_3(\alpha'_3 * I + \beta'_3), \ldots, B_m(\alpha'_m * I + \beta'_m))$
      END FORALL
```

[2]If a pair of them are the same, it may not be possible to generate communication-free code for this code fragment. We will discuss this case in *Section 4.2*.

$\vdots$

Note that all the *reference locations* of arrays $A_1, A_2, \ldots, A_n$ with respect to the template array TEMP are all the same. This ensures that the first parallel loop is communication free. The similar situation also holds for the second parallel loop. We give one more example below to demonstrate the way to produce communication-free codes with our segmented alignment.

Example 1 Consider the *Code Fragment 6* below. The segmented alignment compiler directives in lines 1-4 ensure that the loop comprising lines 5-7 is communication free.

HPF Code Fragment 6
```
1     !HPF$   TEMPLATE TEMP(N)
2     !HPF$   ALIGN A(I) WITH TEMP(5*I) WHEN (I) IN (605:3005:18)
3     !HPF$   ALIGN B(I) WITH TEMP(6*I+61) WHEN (I) IN (494:2494:15)
4     !HPF$   ALIGN C(I) WITH TEMP(10*I-175) WHEN (I) IN (320:1520:9)
                            ⋮
5             FORALL (I=100:500:3)
6                A(6*I+5)=B(5*I-6)/C(3*I+20)
7             END FORALL
                            ⋮
```

4.2 Array Reference Conflicts and Communication-Free Constraints

We first describe the concept of reference conflicts, which are used later for our automatic alignment schemes. There are two types of reference conflicts: intra-loop and inter-loop. Both types will result in communications. In the following we give the definition of reference conflicts. We use *Code Fragment 5* as an example code to illustrate the principle.

Definition 5 We say that an **intra-loop reference conflict** occurs in the first parallel loop of *Code Fragment 5* if the following conditions hold:

(1) There exist two arrays A_i and A_j, $1 \leq i \leq n$ and $1 \leq j \leq n$, where A_i and A_j are the the same array.

(2) The intersection of reference sections of A_i and A_j in this loop is not an empty set, which means $(\alpha_i * L + \beta_i : \alpha_i * U + \beta_i : \alpha_i * S) \cap (\alpha_j * L + \beta_j : \alpha_j * U + \beta_j : \alpha_j * S) \neq \emptyset$. Since A_i and A_j are the same array, such conditions may hold.

(3) According to the segmented alignment relation defined in *Code Fragment 5*, the reference locations of $A_i(\alpha_i * I + \beta_i)$ and $A_j(\alpha_j * I + \beta_j)$ with respect to **TEMP** are different.

Definition 6 An **inter-loop reference conflict** occurs in the first and second parallel loops of *Code Fragment 5* if the following conditions hold:

(1) There exist two arrays A_i and B_j, $1 \leq i \leq n$ and $1 \leq j \leq m$, where A_i and B_j are the the same array.

(2) The intersection of reference sections of A_i and B_j in this loop is not an empty set, which means $(\alpha_i * L + \beta_i : \alpha_i * U + \beta_i : \alpha_i * S) \cap (\alpha'_j * L' + \beta'_j : \alpha'_j * U' + \beta'_j : \alpha'_j * S') \neq \emptyset$. Since A_i and B_j are the same array, such conditions may hold as well.

(3) According to the segmented alignment relation defined in *Code Fragment 5*, the reference locations of $A_i(\alpha_i * I + \beta_i)$ and $B_j(\alpha'_j * I + \beta'_j)$ with respect to **TEMP** are different.

We give an example below to illustrate reference conflicts.

Example 2 Consider the *Code Fragment 7* below. Array A is referenced three times in the loop body. The index variable *I* is in the range (I=1:100:1). Thus, the reference section of A(3∗I+50) is A(53:350:3), the reference section of A(2∗I+50) is A(52:250:2), and the reference section of A(I+500) is A(501:600:1). Since (53:350:3) ∩ (52:250:2) is not the empty set, it is impossible to find an alignment relation such that A(3∗I+50) and A(2∗I+50) have the same *reference location* if I=1:100:1. That is, there is an intra-loop reference conflict for A(3∗I+50) and A(2∗I+50), which means that the compiler can not generate communication-free code for this loop.

HPF Code Fragment 7

```
                ⋮
    FORALL (I=1:100:1)
       A(3∗I+50)=A(2∗I+50)/A(I+500)
    END FORALL
                ⋮
```

For a sequence of FORALL loops there are two constraints for the generation of communication-free code. Suppose we have N FORALL loops $\mathcal{P}_1$, $\mathcal{P}_2$, ..., and $\mathcal{P}_N$. The first constraint is that there should be no intra-loop conflict of array references in $\mathcal{P}_1$, $\mathcal{P}_2$, ..., and $\mathcal{P}_N$; the second is that there should be no inter-loop reference conflict in each pair of $\mathcal{P}_1$, $\mathcal{P}_2$, ..., and $\mathcal{P}_N$. For each array used in these FORALL loops, the intersection of index domains defined by segmented alignment for each FORALL loop ($\mathcal{P}_1$, $\mathcal{P}_2$, ..., and $\mathcal{P}_N$) should be the empty set. For example, in *Code Fragment 3* there is no conflict of array references for the four FORALL loops (loops 1-4). This satisfies the first requirement. For the second requirement, we take array A as an example. The index domains defined for the four FORALL loops of array A are (1:$\frac{N}{2}$,1:$\frac{N}{2}$), (1:$\frac{N}{2}$,$\frac{N}{2}+1$:N), ($\frac{N}{2}+1$:N,1:$\frac{N}{2}$), and ($\frac{N}{2}+1$:N,$\frac{N}{2}+1$:N). Their intersection is the empty set. Thus, there exists a communication-free code for those FORALL loops in *Code Fragment 1*.

Once the target FORALL loops satisfy the two requirements for communication-free operation, we can apply the scheme presented in Section 4.1 to $\mathcal{P}_1$, $\mathcal{P}_2$, ..., and $\mathcal{P}_N$. This will produce segmented alignment statements related to each name array in the target FORALL loops, whilst ensuring that the execution of $\mathcal{P}_1$, $\mathcal{P}_2$, ..., and $\mathcal{P}_N$ will not result in remote data communications.

4.3 Optimal Solution

In this section, we describe methods to automatically align arrays with segmented alignment schemes. We first present the segmented alignment graph which is used to calculate the communication cost for evaluating a sequence of FORALL loops. Given a sequence of FORALL loops $\mathcal{F}$, assume that $\mathcal{F}$ comprises N FORALL loops $\mathcal{P}_1$, $\mathcal{P}_2$, ..., and $\mathcal{P}_N$, and that there are M arrays, A_1, A_2, $\cdots$, and A_M, in $\mathcal{F}$. The segmented alignment graph of $\mathcal{F}$ is a union of all the expression trees of $\mathcal{P}_1$, $\mathcal{P}_2$, ..., and $\mathcal{P}_N$, with each expression tree labeled with the corresponding index domain of its FORALL loop. Basically, a segmented alignment graph is a forest consisting of one or multiple expression trees that are labeled with index domains. Thus, the union of several segmented alignment graphs is still a segmented alignment graph. The communication cost for evaluating a segmented alignment graph depends on the segmented alignment relation defined for each name array in the graph. Assume that we encode the specific segmented alignment relation as p, which represents a specific segmented alignment relation of A_1, A_2, $\cdots$, and A_M. Let the segmented alignment graph of $\mathcal{F}$ be $\mathcal{G}$. Then the communication cost for evaluating $\mathcal{F}$ according to the segmented alignment relation specified by π is as follows:

$$\sum_{(r,l)\in E} \sum_{(i_1,\cdots,i_p)\in I} \mathcal{D}_\pi(r,l)$$

where $E = \{\ (r,l) \mid r$ and l represent each pair of parent and child nodes in $\mathcal{G}\}$; and I is the segmentation descriptor of r and l. The distance function $\mathcal{D}_\pi$ is defined according to π.

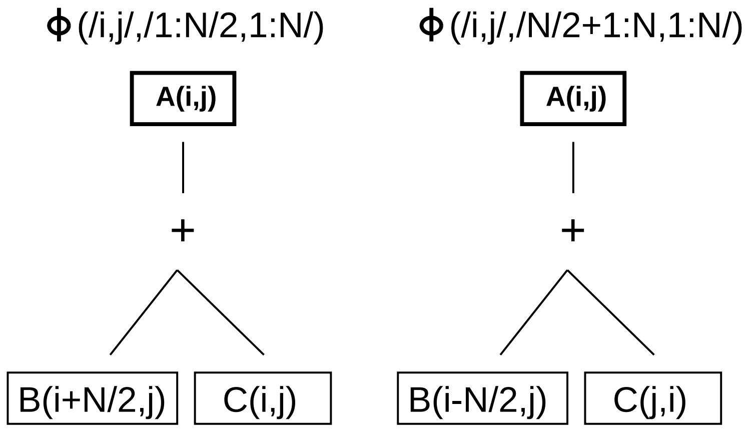

Figure 1: An example of Segmented Alignment Graph.

Example 3 Assume $\mathcal{F}$ is as *Code Fragment 8* below.

HPF Code Fragment 8

```
FORALL (I=1:$\frac{N}{2}$,J=1:$N$)
   A(I,J)=B[I+$\frac{N}{2}$,J]+C[I,J]
END FORALL
FORALL (I=$\frac{N}{2}+1$:N,J=1:N)
   A(I,J)=B[I-$\frac{N}{2}$,J]-C[J,I]
END FORALL
```

The segmented alignment graph of the above data-access function is shown in *Figure 1*. For specific segmented alignment relations of arrays *A*, *B*, and *C*, the communication cost for evaluating $\mathcal{F}$ is $\sum_{(i,j)\in\phi(/i,j/,/1:N/2,1:N/)} \mathcal{D}(A(i,j),B(i+N/2,j)) + \sum_{(i,j)\in\phi(/i,j/,/1:N/2,1:N/)} \mathcal{D}(A(i,j),C(i,j))$ $+ \sum_{(i,j)\in\phi(/i,j/,/N/2+1:N,1:N/)} \mathcal{D}(A(i,j),B(i-N/2,j)) + \sum_{(i,j)\in\phi(/i,j/,/N/2+1:N,1:N/)} \mathcal{D}(A(i,j),C(j,i))$. Note that the segmented alignment relations of arrays *A*, *B*, and *C* are represented by the distance function $\mathcal{D}$.

Assume $\mathcal{A}$ encodes all the sets of possible segmented alignment relations of the arrays in $\mathcal{F}$. Then the optimal solution is

$$\Omega = MIN_{\pi\in\mathcal{A}}\Big(\sum_{(r,l)\in E} \; \sum_{(i_1,\cdots,i_p)\in I} \mathcal{D}_\pi(r,l)\Big)$$

Comparing the segmented alignment and the alignment of HPF, it is obvious that the alignment of HPF is only a special case of the segmented alignment. Since the complexity of the automatic alignment problem for the HPF program is NP-complete [21, 28], we conclude that the automatic segmented alignment problem is NP-hard. Moreover, the number of all possible alignments of the temporary arrays is potentially very large. The alignment function can be decomposed into three constituents: axis, stride, and offset. Although there are only a finite number of possible axis alignments, there is potentially an infinite space of strides and offsets [7]. Also, for considering segmented alignment, an array may have different alignment functions for each data element within it. This makes obtaining the optimal solution computationally prohibitive except for small problems. Theoretically, every possible alignment needs to be tested, but practically it is acceptable to test only those alignments or distributions appearing in a parse tree. Or, to simplify it further, one can try only those related to source or target arrays.

5 Heuristic Algorithm

As mentioned earlier, if a sequence of FORALL loops satisfies the two communication-free requirements (i.e., the absence of intra- or inter-loop reference conflicts), we can easily derive an appropriate segmented alignment relation in polynomial time for the arrays in those loops to obtain communication-free execution. However, *Section 4.3* states that if a sequence of FORALL loops cannot satisfy the two communication-free requirements, the optimal algorithm for deriving the segmented alignment relation of the arrays that enables execution with minimal communication cost is NP-hard. Therefore, we present a practical and efficient heuristic algorithm in this section to guide the process of code generation. The heuristic algorithm is given below.

Algorithm 1 A heuristic algorithm to choose the segmented alignment relations for arrays in a sequence of FORALL loops.

Input: A sequence of FORALL loops

Begin

Step 1. Construct the segmented alignment graphs of the target FORALL loops. Assume that $\mathcal{G}$ is the derived segmented alignment graph.

Step 2. Detect the reference conflicts[3] in $\mathcal{G}$. For each reference conflict which exists in two leaf nodes, a conflict edge is added to link these nodes. Designate the resulting graph as $\mathcal{G}'$.

Step 3. Label all the leaf nodes in $\mathcal{G}'$ as ”marked,” except those leaf nodes which have a reference conflict relation to the root node. Designate the resulting graph as $\mathcal{G}''$.

Step 4. Consider the conflict edges in $\mathcal{G}''$. For each strongly connected component which is linked by these conflict edges, choose the node that has the largest segmentation descriptor[4]. Let this node in the strongly connected component remain labeled as ”marked,” and label the others as ”unmarked.”

Step 5. Use the scheme presented in *Section 4.1* to define the segmented alignment relation for the root node and leaf nodes which are labeled ”marked.”

End

According to the owner-computes rule of HPF, communication will be required between the unmarked nodes and their corresponding root nodes. This also explains why we want to choose a node with the largest segmentation descriptor in Step 4. That is, we prefer aligning array references with larger index domains to array references with smaller index domain if two array references conflict with each other. We use the following example to illustrate the above scheme.

Example 4 See the following code fragment:

HPF Code Fragment 9
```
        REAL A(N,N), B(N,N), C(N,N), D(N,N), E(N,N), F(N,N)
                    ⋮
        F=TRANSPOSE(B+C+D+E)/(D*E)
        A=CSHIFT(EOSHIFT(F,N/10,-7.7,2),N/5,1)
                    ⋮
```

We will demonstrate our alignment scheme after the above codes are optimized by so-called array operation synthesis. A straightforward compilation for these consecutive array functions or array expressions may translate each array operation into a (parallel) nested loop, and use a temporary array to pass intermediate results to subsequent array functions. The synthesis of multiple consecutive array functions or array expressions can transform several data-access functions into an equivalent composite reference pattern. Thus, the synthesis can improve performance by reducing the movement of redundant data,

[3]Reference conflict is defined in *Section 4.2*. If there is no reference conflict in $\mathcal{G}$, then we can apply the scheme presented in *Section 4.2* to obtain communication-free code.

[4]If there are several nodes that have the same maximal size of segmentation descriptor, arbitrarily choose one of them.

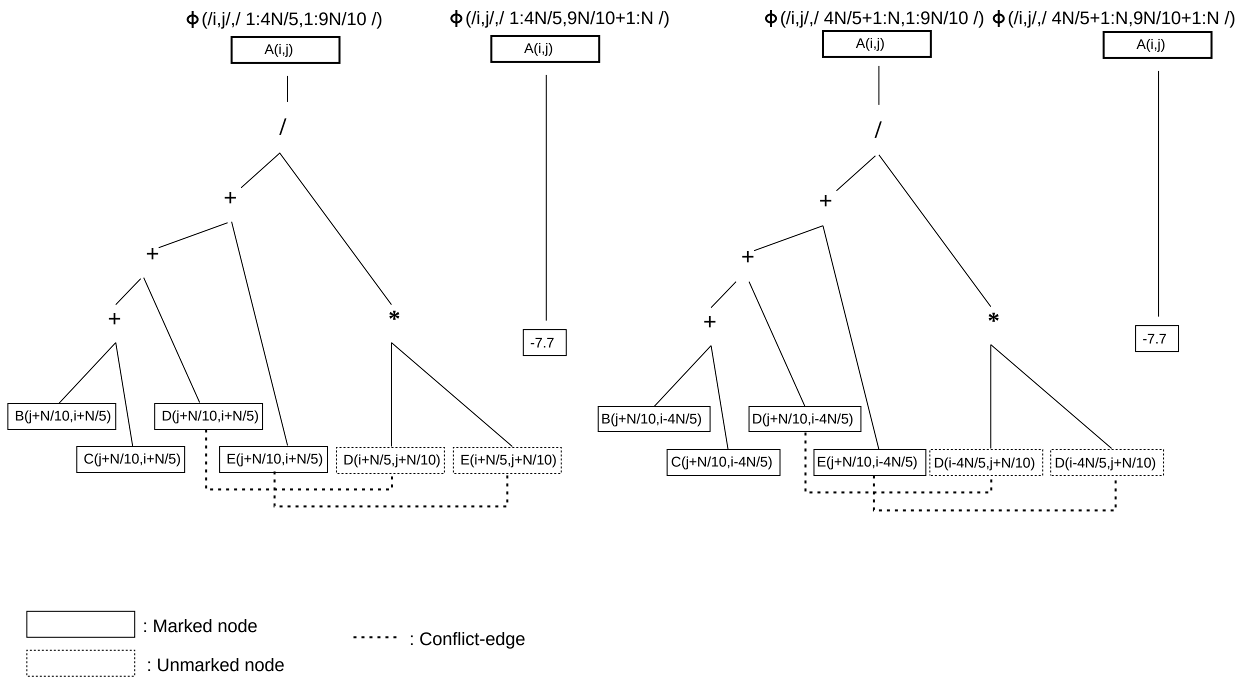

Figure 2: A segmented alignment graph of FORALL loops.

temporary storage usage, and parallel-loop synchronization overhead. Applying array operation synthesis developed in [13, 15, 14], we have the synthesized codes as follows:

```
        REAL A(N,N), B(N,N), C(N,N), D(N,N), E(N,N), F(N,N)
                  ⋮
!Loop 1
        FORALL (i=1:$\frac{4N}{5}$,j=1:$\frac{9N}{10}$)
           $A(i,j) = (B(j+\frac{N}{10},i+\frac{N}{5})+C(j+\frac{N}{10},i+\frac{N}{5})+D(j+\frac{N}{10},i+\frac{N}{5})+E(j+\frac{N}{10},i+\frac{N}{5}))/(D(i+\frac{N}{5},j+\frac{N}{10})*E(i+\frac{N}{5},j+\frac{N}{10}))$
        END FORALL
!Loop 2
        FORALL (i=1:$\frac{4N}{5}$,j=$\frac{9N}{10}$+1:N)
           A(i,j)=-7.7
        END FORALL
!Loop 3
        FORALL (i=$\frac{4N}{5}+1:N$,j=1:$\frac{9N}{10}$)
           $A(i,j) = (B(j+\frac{N}{10},i-\frac{4N}{5})+C(j+\frac{N}{10},i-\frac{4N}{5})+D(j+\frac{N}{10},i-\frac{4N}{5})+E(j+\frac{N}{10},i-\frac{4N}{5}))/(D(i-\frac{4N}{5},j+\frac{N}{10})*E(i-\frac{4N}{5},j+\frac{N}{10}))$
        END FORALL
!Loop 4
        FORALL (i=$\frac{4N}{5}+1:N$,j=$\frac{9N}{10}+1:N$)
           A(i,j)=-7.7
        END FORALL
```

The segmented alignment graph of the above code after applying array operation synthesis is shown in *Figure 2*. We use the scheme we presented in *Section 4.1* to define the segmented alignment relation for the root node and leaf nodes labeled as ”marked.” The conflict edges are also illustrated in *Figure 2*. After employing our heuristic alignment algorithm, we have the following codes:

```
        REAL A(N,N), B(N,N), C(N,N), D(N,N), E(N,N), F(N,N)
!HPF$   TEMPLATE TEMP1(N,N)
!HPF$   ALIGN A(I,J) WITH TEMP1(I,J) WHEN (I,J) IN (1:$\frac{4N}{5}$,1:$\frac{9N}{10}$)
!HPF$   ALIGN B(I,J),C(I,J),D(I,J),E(I,J) WITH TEMP1(J-$\frac{N}{5}$,I-$\frac{N}{10}$) WHEN (I,J) IN ($\frac{N}{10}+1$:N,$\frac{N}{5}+1$:N)
!HPF$   TEMPLATE TEMP2(N,N)
!HPF$   ALIGN A(I,J) WITH TEMP2(I,J) WHEN (I,J) IN ($\frac{4N}{5}+1:N$,1:$\frac{9N}{10}$)
!HPF$   ALIGN B(I,J),C(I,J),D(I,J),E(I,J) WITH TEMP1(J+$\frac{4N}{5}$,I-$\frac{N}{10}$) WHEN (I,J) IN ($\frac{N}{10}+1$:N,1:$\frac{N}{5}$)
```

```
!HPF$   DISTRIBUTE TEMP1(BLOCK,BLOCK),TEMP2(BLOCK,BLOCK)
                 ⋮
!Loop 1
        FORALL (i=1:$\frac{4N}{5}$,j=1:$\frac{9N}{10}$)
           $A(i,j) = (B(j+\frac{N}{10}, i+\frac{N}{5}) + C(j+\frac{N}{10}, i+\frac{N}{5}) + D(j+\frac{N}{10}, i+\frac{N}{5}) + E(j+\frac{N}{10}, i+\frac{N}{5}))/(D(i+\frac{N}{5}, j+\frac{N}{10}) * E(i+\frac{N}{5}, j+\frac{N}{10}))$
        END FORALL
!Loop 2
        FORALL (i=1:$\frac{4N}{5}$,j=$\frac{9N}{10}$+1:N)
           A(i,j)=-7.7
        END FORALL
!Loop 3
        FORALL (i=$\frac{4N}{5}+1:N$,j=1:$\frac{9N}{10}$)
           $A(i,j) = (B(j+\frac{N}{10}, i-\frac{4N}{5}) + C(j+\frac{N}{10}, i-\frac{4N}{5}) + D(j+\frac{N}{10}, i-\frac{4N}{5}) + E(j+\frac{N}{10}, i-\frac{4N}{5}))/(D(i-\frac{4N}{5}, j+\frac{N}{10}) * E(i-\frac{4N}{5}, j+\frac{N}{10}))$
        END FORALL
!Loop 4
        FORALL (i=$\frac{4N}{5}+1:N$,j=$\frac{9N}{10}+1:N$)
           A(i,j)=-7.7
        END FORALL
```

The communication cost for executing Loop 1 of the above code is $\sum_{(i,j)\in\phi(/i,j/,/1:4N/5,1:9N/10/)} \mathcal{D}(A(i,j), D(i+N/5, j+N/10) + \mathcal{D}(A(i,j), E(i+N/5, j+N/10) = \frac{4N}{5} * \frac{9N}{10} = \frac{18N^2}{25}$, and the communication cost for executing the loop 3 of the above code is $\sum_{(i,j)\in\phi(/i,j/,/4N/5+1:N,1:9N/10/)} \mathcal{D}(A(i,j), D(i-N/5, j+N/10) + \mathcal{D}(A(i,j), E(i-N/5, j+N/10) = \frac{N}{5} * \frac{9N}{10} = \frac{9N^2}{50}$. Thus, the total communication cost is $\frac{18N^2}{25} + \frac{9N^2}{50} = \frac{9N^2}{10}$.

The previous algorithm done in [14] shows that the unmarked nodes will need remote communication based on the owner-computes rule of HPF. Note that this was an algorithm for handling performance anomaly with array operation synthesis on distributed-memory parallel environments. However, if several unmarked nodes are in the same subtree, we may further reduce the communication, since we can derive the reference locations for all these nodes. We can then traverse the graph to find the maximal subtrees that satisfy the following three criteria: (i) the number of their leaf nodes is greater than 1, (ii) all their leaf nodes are unmarked, and (iii) all their leaf nodes have the same reference location with respect to the same template array. Once the subtree is identified, we use a separate parallel loop to execute its corresponding subexpression and rollback a temporary array to save the intermediate results of the computation of the subtree. Finally, the newly generated temporary array is aligned with any one of the arrays in the found subtree. The generated temporary array is then inserted into the original expression tree to pass the intermediate results.

6 Experiments

In this section, we report experiments and performance results to show the effectiveness of segmented alignments. Our input programs are written in Fortran 90, and they become HPF programs after optimizations with automatic alignments. Our testbed is a computing farm comprising eight Alpha 3000/900 workstations connected by FDDI (ring) networks, and running the DEC HPF compiler [12]. Because segmented alignment is not supported in the conventional HPF compiler, we split arrays into subarrays and use HPF compiler directives to specify the alignment relation for these subarrays. *Appendix A* gives an example of such an emulation.

We first give experimental results for the motivating example in *Code Fragments 1* and *2* to demonstrate the performance of programs when the communication-free requirements are met with array operation synthesis and segmented alignments. Below we listed the same code as in *Section 2*, except that we also experiment with different **Shift_Size** (see *Table 1*).

```
REAL A(N,N), B(N,N), C(N,N)
                ⋮
DO I=1, Number_Iteration
```

	Shift_Size=128				Shift_Size=64			
	p=1	p=2	p=4	p=8	p=1	p=2	p=4	p=8
Code A	0.38	3.10	2.72	2.67	0.46	2.46	2.01	2.12
Code B	0.33	0.16	0.12	0.066	0.42	0.21	0.12	0.064
	Shift_Size=32				Shift_Size=16			
	p=1	p=2	p=4	p=8	p=1	p=2	p=4	p=8
Code A	0.52	2.36	1.88	1.83	0.57	2.37	1.85	1.80
Code B	0.42	0.21	0.12	0.062	0.31	0.19	0.13	0.062
	Shift_Size=8				Shift_Size=4			
	p=1	p=2	p=4	p=8	p=1	p=2	p=4	p=8
Code A	0.59	2.48	1.98	1.98	0.63	2.53	2.19	1.99
Code B	0.32	0.17	0.13	0.064	0.48	0.18	0.13	0.062

Table 1: The performance of the *Code Fragment 1* with synthesis and with/without segmented alignments

```
    A=CSHIFT((TRANSPOSE(EOSHIFT(B,Shift_Size,-6.6,1))+C),Shift_Size,1)
ENDDO
            ⋮
```

Code A is the performance result obtained by executing *Code Fragment 1* and performing conventional data alignments with the whole index domain of an array; *Section 2* shows that communications occur in this case. Code B is generated according to *Code Fragment 3*, and it is communication free. The size of the data grid in our test case was 256 (i.e., N=256). The execution times were measured by running the program fragments 100 times. The experimental results show that Code B was much more scalable than code A regardless of the shift strides used. The use of segmented alignment makes the synthesized code more scalable in parallel environments than codes produced by conventional automatic alignments. Without our proposed segmented alignment, communication-free execution would not be possible in this case when using the conventional HPF data alignment model to align arrays with the whole index domain.

Table 2 provides more examples to explicitly compare the performance with or without our segmented alignment scheme, which further demonstrates the reductions in communication costs that our method produces. *Appendix B* gives the source codes of these code fragments. The first test suite is the *Code Fragment 9* used earlier in this paper. We experimented with two versions of the code. The first version used array synthesis [13, 15] and conventional alignment to align the arrays with the whole index domain. The second version integrated array operation synthesis with our proposed segmented alignment scheme using our heuristic algorithm. With our proposed segmented alignment scheme, the parallel algorithm is more scalable and runs much better on the eight processors constituting our farm. The second code fragment is the example in Chatterjee et al. [25] modified by adding CSHIFT and EOSHIFT array primitives, since Chatterjee et al. [7] could not handle cases including those two array functions, which need segmented alignment to obtain communication-free codes. Comparison of the two versions of our code illustrates the aggregate benefits of the integration of segmented alignment scheme presented in this paper and the array operation synthesis scheme we invented previously [15]. Finally, we performed experiments on the third test suite in *Table 2*, which comes from page 1242 of Numerical Recipes in Fortran 90 [26] and involves two FFTs. The variables were as follows. We set n to 256*256, and h1, h2, fft1, and fft2 were arrays of complex numbers. In addition, h1(2:n2) was aligned with fft1(2:n2) and fft1(n:n2:-1), and h2(2:n2) was aligned with fft2(2:n2) and fft2(n:n2:-1). The program was executed with or without our proposed segmented alignment - with our proposed segmented alignment and automatic alignment, the program is more scalable than the one without it.

	p=1	p=2	p=4	p=8
The 1st suite after synthesis and segmented alignment	2.19	1.23	0.63	0.32
The 1st suite after synthesis, but without segmented alignment	1.88	3.38	2.61	2.06
The 2nd suite without synthesis and segmented alignment	6.01	13.3	8.41	5.03
The 2nd suite after synthesis and segmented alignment	3.34	6.01	3.578	1.89
The 3rd suite with segmented alignment	1.21	1.89	0.68	0.38
The 3rd suite without segmented alignment	1.01	1.93	1.42	0.81

Table 2: Experimental results with codes using our heuristic algorithms

7 Conclusion

In this paper, we have proposed a new array alignment concept that we call segmented alignment. Our strategy for solving the alignment problem in data parallel programs uses a two-step scheme. In the first step, we test if it is possible to obtain a communication-free code with segmented alignment. If a communication-free code can be obtained in this way, the output code is the optimal solution. If we cannot obtain a communication free code in the first step, we use our new methods in the second step. Due to the optimal problem being NP-hard, we have developed a practical heuristic algorithm for compilers to incorporate segmented alignment into an automatic alignment framework. Experiments performed on an eight-node DEC Alpha Farm have shown that an automated alignment process using the segmented alignment concept can significantly outperform conventional alignment models.

References

[1] J. C. Adams, W. S. Brainerd, J. T. Martin, B. T. Smith, and J. L. Wagener. *Fortran 90 Handbook complete ANSI/ISO reference*. Intertext Publications McGraw-Hill Book Company, 1992.

[2] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers Principles, Techniques, and Tools*. Addison-Wesley Publishing Company, 1986.

[3] J. M. Anderson and M. S. Lam. Global optimizations for parallelism and locality on scalable parallel machines. In *Proceedings of the ACM SIGPLAN'93 Conference on Programming Language Design and Implementation*, pp. 112-125, June 1993.

[4] R. Bixby, K. Kennedy, and U. Kremer. Automatic Data Layour Using 0-1 Intefer Programming. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, Montreal, Canada, August 1994.

[5] Z. Bozkus, A. Choudhary, G. Fox, T. Haupt, S. Ranka, and M. Y. Wu. Compiling Fortran 90D/HPF for Distributed Memory MIMD Computers. *Journal of parallel and Distributed Computing*, 21, 1 (April 1994), 15-26.

[6] T. A. Budd. An APL Compiler for a Vector Processor. *ACM Transactions on Programming Languages and Systems*, V6, No 3, pp. 297–313, July 1984.

[7] S. Chatterjee, J. R. Gilbert, R. Schreiber, and S.-H. Teng. Automatic Array Alignment in Data-Parallel Programs. In *Proceedings of the Twentieth Annual SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Charleston, SC, January 1993, pp. 16-28.

[8] J. R. Gilbert and R. Schreiber. Optimal expression evaluation for data parallel architectures. *Journal of Parallel and Distributed Computing*, Vol.13, pp. 58-64, Sept. 1991.

[9] L. J. Guibas and D. K. Wyatt. Compilation and Delayed Evaluation in APL. In *Proceeding of the Fifth Annual ACM Symp. on Principles of Programming Languages*, pp. 1-8, 1978.

[10] M. Gupta and P. Banerjee. Demonstration of automatic data partitioning techniques for parallelizing compilers on multicomputers. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 3, pp. 179-193, Mar. 1992.

[11] M. Gupta, and P. Banerjee. PARADIGM: A compiler for automatic data distribution on multicomputers. In *ACM International Conference on Supercomputing*, Tokyo, July 1993.

[12] J. Harris *et al.* Compiling High Performance Fortran for Distributed-memory Systems. *Digital Technical Journal*, Volume 7, Number 3.

[13] G. H. Hwang, J. K. Lee, and D. C. Ju. An Array Operation Synthesis Scheme to Optimize Fortran 90 Programs. In *Proceedings of ACM SIGPLAN Conference on Principles and Practice of Parallel Programming*, pp. 112-122, July 1995. Also, ACM SIGPLAN NOTICES, Volumn 30, Number 8, Auguest 1995.

[14] G. H. Hwang, J. K. Lee, and D. C. Ju. Array Operation Synthesis to Optimize HPF Programs. In *Proceedings of the 1996 International Conference on Parallel Processing*, Volumn 3, pp. 1-8, August 1996.

[15] G. H. Hwang, J. K. Lee, and D. C. Ju. A Function-Composition Approach to Synthesize Fortran 90 Array Operations. *Journal of Parallel and Distributed Computing,* 54, pp.1-47, 1998.

[16] K. Kennedy and U. Kremer. Automatic data layout for distribution-memory machines. *ACM Transactions on Parallel Languages and Systems*, 20(4):869–916, July 1998.

[17] K. Knob, J. D. Lukas, and G. L. Steele. Data optimization: allocation of arrays to reduce communication on SIMD Machines. *Journal of parallel and Distributed Computing*, Vol.2, pp. 102-118, Feb. 1990.

[18] C. Koelbel. Compile-time generation of regular communications patterns. In *Proceedings of Supercomputing'91*, pages 101–110, 1991.

[19] C. Koelbel, D. Loveman, R. Schreiber, G. Steele and M. Zosel. *The High Performance Fortran Handbook*, MIT-press, Cambridge, 1994.

[20] J. K. Lee and D. Gannon. Object-Oriented Parallel Programming: Experiments and Results. In *Proceedings of Supercomputing '91*, New Mexico, November, 1991.

[21] J. Li and M. Chen. The data alignment phase in compiling programs for distributed-memory machines. *Journal of parallel and Distributed Computing*, Vol. 13, pp. 213-221, Oct. 1991.

[22] J. Li and M. Chen. Compiling Communication-Efficient Programs for Massively Parallel Machines. *IEEE Tran. On Parallel and Distributed Systems*, Vol. 2, No. 3, July 1991.

[23] A. Mohamer, G. Fox, G. Laszewski, M. Parashar, T. Haupt, K. Mills, Y. Lu, N. Lin, and N. Yeh. Applications Benchmark Set for Fortran-D and High Performance Fortran. Technical Report SCCS327, NPAC, Syracuse University.

[24] M. Philippsen and M. U. Mock. Data and process alignment in Modula-2*. *Automatic Parallelization: New Approaches*, pages 177–191. Verlag Vieweg, 1994.

[25] M. Philippsen. Automatic alignment of array data and process to reduce communication time on DMPPs. In *Proceedings of the Fifth ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming*, pages 112–122. Santa Barbara, California, USA, July 1995. ACM Press.

[26] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in Fortran90: The Art of Parallel Scientific Computer, Volume 2 of Fortran Numerical Recipes, Second Edition*, Cambridge University Press, 1996

[27] J. R. Rice and J. Jing. Problems to Test Parallel and Vector Languages. Purdue University Technical Report CSD-TR-1016, 1990.

[28] T. J. Sheffler, R. Schreiber, J. R. Gilbert, and S. Chatterjee. Aligning Parallel Arrays to Reduce Communication. In *Proceedings of Frontiers '95*, McLean, VA, February 1995, pp. 324-331.

[29] H. Zima, and B. Chapman. Compiling for distributed-memory systems. In *Proceeding IEEE*, 81, 2 (Feb.1993), 264-287.

A Implementation of Segmented Alignments

Instead of having to reconstruct a compiler specifically for the segmented alignment directives, we can split an array into several subarrays according to the disjointed index domains resulting from array operation synthesis. We then specify those subarrays with suitable alignment relationships to produce communication-free code. Consider the following code fragment:

HPF Code Fragment 10
```
                REAL A(N,N), B(N,N), C(N,N)
                REAL A1(1:N/2,1:N/2),A2(1:N/2,N/2+1:N), A3(N/2+1:N,1:N/2), A4(N/2+1:N,N/2+1:N)
                REAL B1(N/2+1:N,N/2+1:N),B3(N/2+1:N,1:N/2)
                REAL C1(N/2+1:N,1:N/2),C2(N/2+1:N,N/2+1:N),C3(1:N/2,1:N/2), C4(1:N/2,N/2+1:N)
        !HPF$   TEMPLATE TEMP1(N/2,N/2)
        !HPF$   ALIGN A1(I,J) WITH TEMP1(I,J)
        !HPF$   ALIGN B1(I,J) WITH TEMP1(J-N/2,I-N/2)
        !HPF$   ALIGN C1(I,J) WITH TEMP1(I-N/2,J)
        !HPF$   TEMPLATE TEMP2(N/2,N/2)
        !HPF$   ALIGN A2(I,J) WITH TEMP2(I,J)
        !HPF$   ALIGN C2(I,J) WITH TEMP2(I-N/2,J)
        !HPF$   TEMPLATE TEMP3(N/2,N/2)
        !HPF$   ALIGN A3(I,J) WITH TEMP3(I,J)
        !HPF$   ALIGN B3(I,J) WITH TEMP3(J+N/2,I-N/2)
        !HPF$   ALIGN C3(I,J) WITH TEMP3(I+N/2,J)
        !HPF$   TEMPLATE TEMP4(N/2,N/2)
        !HPF$   ALIGN A4(I,J) WITH TEMP4(I,J)
        !HPF$   ALIGN C4(I,J) WITH TEMP4(I+N/2,J)
        !HPF$   DISTRIBUTE TEMP1(BLOCK,BLOCK),TEMP2(BLOCK,BLOCK),TEMP3(BLOCK,BLOCK),TEMP4(BLOCK,BLOCK)
                                 ⋮
        !Data Realignment before computation bound parallel loops
                A1(1:N/2,1:N/2)=A(1:N/2,1:N/2)
                A2(1:N/2,N/2+1:N=A(1:N/2,N/2+1:N)
                A3(N/2+1:N,1:N/2)=A(N/2+1:N,1:N/2)
                A4(N/2+1:N,N/2+1:N=A(N/2+1:N,N/2+1:N)
                B1(N/2+1:N,N/2+1:N)=B(N/2+1:N,N/2+1:N)
```

```
26              B3(N/2+1:N,1:N/2)=B(N/2+1:N,1:N/2)
27              C1(N/2+1:N,1:N/2)=C(N/2+1:N,1:N/2)
28              C2(N/2+1:N,N/2+1:N)=C(N/2+1:N,N/2+1:N)
29              C3(1:N/2,1:N/2=C(1:N/2,1:N/2)
30              C4(1:N/2,N/2+1:N)=C(1:N/2,N/2+1:N)
31              DO ITER=1, Number_Iteration
32      !Loop 1
33                 FORALL (I=1:N/2,J=1:N/2)
34                    A1(I,J)=B1(J+N/2,I+N/2)+C1(I+N/2,J)
35                 END FORALL
36      !Loop 2
37                 FORALL (I=1:N/2,J=N/2+1:N)
38                    A2(I,J)=-6.6+C2(I+N/2,J)
39                 END FORALL
40      !Loop 3
41                 FORALL (I=N/2+1:N,J=1:N/2)
42                    A3(I,J)=B3(J+N/2,I-N/2)+C3(I-N/2,J)
43                 END FORALL
44      !Loop 4
45                 FORALL (I=N/2+1:N,J=N/2+1:N)
46                    A4(I,J)=-6.6+C4(I-N/2,J)
47                 END FORALL
48              ENDDO
49      !Data Realignment after computation bound parallel loops
50              A(1:N/2,1:N/2)=A1(1:N/2,1:N/2)
51              A(1:N/2,N/2+1:N=A2(1:N/2,N/2+1:N)
52              A(N/2+1:N,1:N/2)=A3(N/2+1:N,1:N/2)
53              A(N/2+1:N,N/2+1:N=A4(N/2+1:N,N/2+1:N)
54              B(N/2+1:N,N/2+1:N)=B1(N/2+1:N,N/2+1:N)
55              B(N/2+1:N,1:N/2)=B3(N/2+1:N,1:N/2)
56              C(N/2+1:N,1:N/2)=C1(N/2+1:N,1:N/2)
57              C(N/2+1:N,N/2+1:N)=C2(N/2+1:N,N/2+1:N)
58              C(1:N/2,1:N/2)=C3(1:N/2,1:N/2)
59              C(1:N/2,N/2+1:N)=C4(1:N/2,N/2+1:N)
                           ⋮
```

As in *Code Fragment 3*, array A is separated into A1, A2, A3, and A4; array B is separated into B1 and B3; and array C is separated into C1, C2, C3 and C4. Template arrays TEMP1, TEMP2, TEMP3, and TEMP4 are used to align the array references in Loops 1-4, respectively. Lines 5-8 align A(I,J), B(J+$\frac{N}{2}$,I+$\frac{N}{2}$), and C(I+$\frac{N}{2}$,J) in index domain (I=1:$\frac{N}{2}$,J=1:$\frac{N}{2}$) of Loop 1; lines 9-11 align A(I,J) and C(I+$\frac{N}{2}$,J) in index domain (I=1:$\frac{N}{2}$,J=$\frac{N}{2}+1$:N) of Loop 2; lines 12-15 align A(I,J), B(J+$\frac{N}{2}$,I-$\frac{N}{2}$), and C(I-$\frac{N}{2}$,J) in index domain (I=$\frac{N}{2}+1$:N,J=1:$\frac{N}{2}$) of Loop 3; lines 16-18 align A(I,J), and C(I-$\frac{N}{2}$,J) in index domain (I=$\frac{N}{2}+1$:N,J=$\frac{N}{2}+1$:N) of Loop 4. Since all the array references in Loops 1-4 (lines 31-48) are all aligned, these loops are communication free. Lines 20-30, and 49-59 realign the split subarrays with the original array.

B Code Fragments for Experiments

1.
```
PARAMETER (N=256,N_ITER=100)
INTEGER ITER
REAL A(N,N),B(N,N),C(N,N),D(N,N),E(N,N),F(N,N),G(N,N),H(N,N)

DO ITER=1,N_ITER
```

```
         F=TRANSPOSE(B+C+D+E)/(D*E)
         A=CSHIFT(EOSHIFT(F,N/10,-7.7,2),N/5,1)
         H=TRANSPOSE(G)
        ENDDO

        STOP
        END
```

2.

```
        PARAMETER (N=256,N_ITER=100)
        INTEGER ITER
        REAL     P(N,N),B(N,N),C(N,N),D(N,N),E(N,N),K(N,N),M(N,N)
        REAL     N(N,N),F(N,N),G(N,N),H(N,N),A(N,N),R(N,N)
        REAL     T1(N,N),T2(N,N),T3(N,N),T4(N,N),T5(N,N),T6(N,N),T7(N,N)

        DO ITER=1,N_ITER
         T1=EOSHIFT(P,16,-SIN(.375),1)
         T2=B+C
         T3=D+E
         T4=TRANSPOSE(T2)+TRANSPOSE(T3)
         K=TRANSPOSE(M)+N
         T5=D*E
         T6=T5+F
         G=T4/T6
         T7=TRANSPOSE(T1)+C
         H=CSHIFT(T7,16,1)
         A=ESHIFT(G,26,-COS(.375),1)
         R=CSHIFT(A,50,1)
        ENDDO

        STOP
        END
```

3. In the page 1242 of Numerical Recipe in Fortran 90

```
        SUBROUTINE  twofft(data1, data2, fft1,fft2)
        USE nrtype; USE nrutil, ONLY : assert, assert_eq
        USE nr, ONLY: four1
        IMPLICIT NONE
        REAL (SP), DIMENSION(:), INTENT(IN) :: data1, data2
        COMPLEX(SPC), DIMENSION(:), INTENT(OUT) :: fft1, fft2
        INTEGER(I4B) :: n, n2
        COMPLEX (SPC), PARAMETER :: C1=(0.5_sp, 0.0_sp), C2=(0.0_sp, -0.5_sp)
        COMPLEX, DIMENSION(size(data1)/2+1) :: h1, h2
        n=assert_eq(size(data1),size(data2),size(fft1),size(fft2),'twofft')
        call assert(iand(n,n-1)==0,'n must be a power of 2 in twofft')
        fft1=cmplx(data1,data2, kind=spc)
        call four1(fft1,1)
        fft2(1)=cmplx(aimag(fft1(1)),0.0_sp, kind=spc)
        fft1(1)=cmplx(aimag(fft1(1)),0.0_sp, kind=spc)
        n2=n/2+1
        h1(2:n2)=C1*(fft1(2:n2)+conjg(fft1(n:n2:-1)))
        h2(2:n2)=C2*(fft1(2:n2)-conjg(fft1(n:n2:-1)))
        fft1(2:n2)=h1(2:n2)
        fft1(n:n2:-1)=conjg(h1(2:n2))
        fft2(2:n2)=h2(2:n2)
        fft2(n:n2:-1)=conjg(h2(2:n2))
        END SUBROUTINE twofft
```