

Using Lagrange Multipliers to Solve the Alignment Problem

I. Couvertier-Reyes *

September 8, 1999

Abstract

Non-local memory accesses are slower than local memory accesses in any computer system. It is therefore very important that the data that needs to be accessed by a processor during the execution of the iterations that are assigned to it reside in its local memory rather than in some other processor's memory. In some applications it is possible to guarantee that all the data that need to be accessed by each processor of a parallel computer system be local to the memory it owns. For many applications, however, this simply can not be done.

The process by which data is distributed to the memory of the processors in a parallel machine is known as the data mapping process. This data mapping process is usually represented as an alignment phase followed by a distribution phase. We present a new method for determining the alignment of both computation and data. The owner-computes rule is thus relaxed. We model the distance function using the Euclidean metric and then use the Lagrange Multiplier method to come up with a set of simultaneous equations which are

*Dept. of Electrical & Computer Engineering, University of Puerto Rico-Mayaguez, PO BOX 9042, Mayaguez, PR 00681-9042 (icouver@ece.uprm.edu). Supported in part by an NSF Award CCR-9810138.

necessary for the existence of a relative minimum. We handle stride, offset, and axis alignment using the same framework.

Index terms: Parallelizing compilers, automatic data mapping, lagrange multipliers, automatic alignment, parallel computing.

1 Introduction

Non-local memory accesses are slower than local memory accesses in any computer system. This difference in the accesses to non-local versus local memory accesses is even more evident in distributed memory computers, though it can also be seen in a smaller magnitude in shared memory computers. It is therefore very important that the data that needs to be accessed by a processor during the execution of the iterations that are assigned to it reside in its local memory rather than in some other processor's memory. In some applications it is possible to guarantee that all the data that need to be accessed by each processor of a parallel computer system be local to the memory it owns. For many applications, however, this simply can not be done.

The process by which data is distributed to the memory of the processors in a parallel machine is known as the data mapping process. This data mapping process is usually represented as an alignment phase followed by a distribution phase. The alignment problem is the determination of the relative positions of the arrays within a Cartesian grid of virtual processors, called a template, so that communication among the processors is minimized by reducing the number of non-local accesses. The distribution problem is the mapping of these virtual processors onto the available physical processors. Our focus for now is the alignment problem which can be classified into offset alignment, stride alignment, and axis alignment. Another type of alignment is the replication of arrays.

In this work we show how to solve the computation and data alignment problems using the Euclidean metric and the well known Lagrange Multiplier method. Once the problem has been modeled, we use the software Mathematica from Wolfram Research, Inc., [24], to solve for the unknowns. This method allows us to specify both equality and inequality constraints whenever necessary. It is important to point out that we deal with the alignment problem in a very uniform way, that is we formulate the stride, offset, and axis alignment problems using the same constraint method and solve them using the Lagrange Multiplier method. This is as opposed to other approaches where different methods are used depending on the type of problem, for example using some heuristics for the stride problem and a different heuristics for the axis alignment problem. Also many researches have focused on the communication-free solution to the alignment problem and provide no way of dealing with the problem when no communication-free solution exists. Our approach deals with the problem when communication-free solutions exist and also when no non-trivial communication-free solution can be found. Our approach is aimed to be used as a tool which is not part of the compiler.

This work is organized as follows. In Section 2 an overview of related research is provided. Section 3 shows how to use the Lagrange Multiplier method for the stride and reversal alignment problem. In Section 4 we present the Lagrange Multiplier method for solving the offset alignment problem. Section 4 also presents the formulation and solution for several real life applications. In Section 5 we present the axis alignment problem and provide a solution using the Lagrange Multiplier method. Finally, in Section 6 we present our conclusions.

2 Related Work

Among the researchers that have addressed the problem of partitioning of data and computation among processors in a data parallel program we have chosen those we believed their work to be most important and review it as it relates to ours.

Kandemir et al. [13], use an optimization technique based on linear algebra. Hyperplanes are used to express suitable memory layouts for each array. Our work not only determines the data layout but it also finds which processor should perform the computation. Desprez et al. [9], Thakur et al. [21], Kalns and Ni [12], and Chung et al. [8] focus on data redistribution. Our work deals with determining the alignment of arrays.

Ramanujam and Sadayappan [19], present a technique which applies to one fully parallel loop nest at a time. Only array data partitions defined by a family of parallel hyperplanes are considered. Communication-free data partitioning is the main subject in Ramanujam and Sadayappan [19]. However, a formulation is given for minimizing communication while balancing the workload, which is used as a constraint, among processors when communication-free partitioning is not possible. The work in Ramanujam and Sadayappan [19] is architecture-independent, and assumes the owner-computes rule. The proposed alignment and the functions used are more general than those in the High Performance Fortran (HPF) standard, where at most one index variable per subscript expression is allowed. We deal with both the data and computation alignment problems and do not focus at finding communication-free partitions and provide a way of dealing with the problem when no communication-free partitions can be found. We do not assume the owner-computes rule but relax it.

Huang and Sadayappan [11] focus on partitions of iterations and data arrays that eliminate data communication and considers partitions of iteration and data spaces

along their hyperplanes. All iterations of an iteration hyperplane and all data of a data hyperplane are assigned to one processor, thus the owner-computes rule is implicit. A processor will execute iterations from the iteration hyperplanes which are assigned to it and in so doing it will access data from its assigned data hyperplanes. The article presents no way of dealing with cases when communication-free partitioning, while maintaining parallelism, is not possible. It begins by presenting solutions for a single hyperplane partitioning for each iteration and data space and moves on to multiple (double) hyperplanes per space at which time they propose a heuristic. Huang and Sadayappan [11] derive necessary and sufficient conditions for communication-free hyperplane partitioning of both data and computation for fully parallel loop nests in the absence of flow and anti-dependences. Like Huang and Sadayappan we deal with both computation and data problems but our method can also be used when no communication-free solution exists and we relax the owner-computes rule.

Gupta and Banerjee [10], present a method restricted to partitioning of arrays, i.e. no computation partitioning. In their method Gupta and Banerjee select important segments of code to determine distribution of various arrays based on some constraints. Quality measures are used to choose among contradicting constraints. These quality measures may require user intervention. The compiler *tries* to combine constraints for each array in a consistent manner to minimize overall execution time and the entire program is considered. Small arrays are assumed to be replicated on all processors. The distribution of arrays is by rows, columns, or blocks, not by hyperplanes as [19], and [11]. This work uses heuristic algorithms to determine the alignment of dimensions, i.e. component alignment, of various arrays since the problem has been shown to be NP-complete. The owner-computes rule is assumed and issues concerning the best way to communicate messages among processors, such as aggregate communication introduced in the work by Tseng [22], which are not

mentioned in any of the other works we have seen so far, are considered. Communication costs are determined by Gupta and Banerjee [10] after identifying the pairs of dimensions that should be aligned. Consideration is given to when it would be best to replicate a dimension rather than to distribute it. We deal not just with the data alignment problem but also with the computation alignment problem. Also we do not make use of heuristics and do not assume the owner-computes rule.

Amarasinghe et al. [1], show how to find partitions for *doall* and *doacross* parallelism and, in order to minimize communication across loop nests, they use a greedy algorithm that tries to avoid the largest amounts of potential communication. They give examples of how to obtain parallelism by incurring some communication when this is the only way to run in parallel. Amarasinghe et al. deal with the problem of computation partitioning along with the data partitioning problem and their method relies on computing the null space for several matrices. Our approach is a constraint based method which is fundamentally different to what is done by Amarasinghe et al.

Bau et al. [3] incorporate, as opposed to the other methods discussed so far, data dependences in their proposed solution to the problem, but as with the other methods the owner-computes rule is assumed. Replication of data is also incorporated into their proposed solution. Bau et al. also deal with the problem of computation partitioning along with the data partitioning problem. However, their approach is useful only for determining communication-free partitioning and assumes the owner-computes rule.

Chatterjee et al. [7] and [6] provide an algorithm that obtains alignments which are more general than the owner-computes rule by decomposing alignment functions into several components. They use dynamic programming and this makes it fundamentally different to our approach.

There are a number of other researchers who have also made contributions to this

problem. Among them Kim and Wolfe [14], Li and Pingali [16]. Kim and Wolfe [14] show how to find and operate on the communication *pattern matrix* from user-aligned references. Our approach generates the alignment of data and computation and frees the user from this task. Li and Pingali [16] start with user specified data distributions and develop a systematic loop transformation strategy identified by them as *access normalization* which restructures loop nests to exploit locality and block transfers whenever possible. Although we are also interested in maintaining locality our approach and theirs are different. We will develop the data and computation distributions based on our findings. The user does not have to specify them. Chatterjee et al. [6] investigate the problem of evaluating FORTRAN 90 style array expressions on massively parallel distributed-memory machines. They present algorithms based on dynamic programming which we do not intend to use here.

O'Boyle [17] proposed an automatic data partition algorithm based on the analysis of four distinct factors. We concur with him in his view that automatic data partitioning is possible and that it must be considered in the context of the whole compilation process rather than be left to the programmer. He does not consider partitioning of computation along with that of data and he is not concerned with finding the alignment that will minimize communication as we are in our work. Wakatani and Wolfe [23] addressed the problem of minimizing communication overhead but from a different context than ours. They are concerned with the communication arising from the redistribution of an array and proposed a technique called *strip mining redistribution*. They are not concerned with automatically generating the alignments as we are in order to free the programmer from this task and achieve minimum communication while preserving parallelism.

3 Stride and Reversal Alignment

When the coefficients of the loop index variables in the subscript expressions of the array references in a program are greater than unity, we have what has been termed as stride alignment. If any of these coefficients is negative, then it is called reversal alignment. Reversal alignment corresponds to mapping the reflection of the array onto the template, Tseng [22].

Stride alignment is generated by statements similar to the following

$$\text{ALIGN } X[i] \text{ WITH } T1[\alpha_X i + \beta_X]$$

where α_X is a positive number, and β_X can be any number. A multidimensional array example is given in the statement

$$\text{ALIGN } Y[i, j] \text{ WITH } T2[\alpha_{Y_1} i + \beta_{Y_1}, \alpha_{Y_2} j + \beta_{Y_2}].$$

An example of reversal alignment is shown in the following statement

$$\text{ALIGN } Y[i] \text{ WITH } T3[-i].$$

Consider the following loop, where array Y is not replicated onto the available processors:

```
do  $i = 1, N$   
     $X[2i - 1] = Y[3i - 1] + Y[3i] + Y[3i + 1]$   
enddo.
```

We could align arrays X and Y and iteration i as follows:

$$\text{ALIGN } X[i] \text{ WITH } T[3i + 4].$$
$$\text{ALIGN } Y[i] \text{ WITH } T[2i + 1].$$
$$\text{ALIGN } i \text{ WITH } T[6i + 1].$$

These alignments were found by inspection.

With the alignment as above, each processor computing an iteration will need an element of Y which is held by the processor to its left, assuming a linear array and a block distribution with block size equal to 3, except of course for the processor at the leftmost position.

Using the owner-computes rule we can obtain the alignment shown below.

ALIGN $X[i]$ WITH $T[3i + 1]$.

ALIGN $Y[i]$ WITH $T[2i + 1]$.

This would require each processor to send two elements to the processor on its right, except for the last processor, assuming the same configuration and distribution as before. Note that there are infinitely many ways to align the arrays in the problem above. We should, however, be concerned with one that results in the least interprocessor communication.

3.1 Stride Alignment via Lagrange Multipliers

Consider the following piece of code:

```
do  $i = 1, N$ 
   $X[a_1i + b_1] = Y[c_1i + d_1] + Y[c_2i + d_2] + \dots + Y[c_r i + d_r]$ 
enddo
```

where arrays X and Y (which is not replicated) will be aligned to a template T as shown below

```
ALIGN  $X[i]$  WITH  $T[\alpha_X i + \beta_X]$ 
ALIGN  $Y[i]$  WITH  $T[\alpha_Y i + \beta_Y]$ 
```

and the iterations of the loop will be aligned with the following declaration

```
ALIGN  $i$  WITH  $T[\alpha_I i + \beta_I]$ 
```

where α_X , β_X , α_Y , β_Y , α_I , and β_I are to be determined.

Let us consider for now iteration i only. We want to minimize the distance from the processor(s) holding the elements of arrays X and Y that are needed to perform

the computation of the element on the left hand side to the processor which will be performing the computation during iteration i . Using the alignment as specified above we can find that the processor which holds the element on the *lhs* is processor $\alpha_X(a_1i+b_1)+\beta_X$. Similarly, the processor holding the first term of array Y is processor $\alpha_Y(c_1i+d_1)+\beta_Y$, the one holding the second term is $\alpha_Y(c_2i+d_2)+\beta_Y$, and so on. Using the l_2 or Euclidean metric the distance from the processor which holds the *lhs* element and the processor which performs the computation during iteration i is given by

$$\left[([\alpha_X(a_1i+b_1)+\beta_X]-[\alpha_Ii+\beta_I])^2\right]^{\frac{1}{2}}.$$

Similarly we can find the distance from the processor(s) holding each one of the elements on the right hand side as:

$$\left[([\alpha_Y(c_ji+d_j)+\beta_Y]-[\alpha_Ii+\beta_I])^2\right]^{\frac{1}{2}}, \text{ for } 1 \leq j \leq r$$

Combining all the terms shown in the last two equations above we can find the sum of the distances of each processor holding an element of X and each processor holding an element of Y from the processor which performs the computation during iteration i as follows:

$$\begin{aligned} \text{distance} &= \left[((\alpha_X(a_1i+b_1)+\beta_X)-(\alpha_Ii+\beta_I))^2 \right. \\ &\quad \left. + \sum_{j=1}^r ((\alpha_Y(c_ji+d_j)+\beta_Y)-(\alpha_Ii+\beta_I))^2 \right]^{\frac{1}{2}}. \end{aligned}$$

If we now consider all the iterations of the loop nest, then the equation above becomes

$$\text{distance} = \left[\sum_{i=1}^N ((\alpha_X(a_1i+b_1)+\beta_X)-(\alpha_Ii+\beta_I))^2 \right]$$

$$+ \left[\sum_{j=1}^r \sum_{i=1}^N (\alpha_Y (c_j i + d_j) + \beta_Y - (\alpha_I i + \beta_I))^2 \right]^{\frac{1}{2}}. \quad (1)$$

Collecting terms and rearranging the above equation we obtain the following

$$\begin{aligned} distance &= \left[\sum_{i=1}^N ((\alpha_X a_1 - \alpha_I) i + \alpha_X b_1 + \beta_X - \beta_I)^2 \right. \\ &\quad \left. + \sum_{j=1}^r \sum_{i=1}^N ((\alpha_Y c_j - \alpha_I) i + \alpha_Y d_j + \beta_Y - \beta_I)^2 \right]^{\frac{1}{2}}. \end{aligned} \quad (2)$$

We can generalize the findings above to the case when we have an arbitrary number l of loop nests, an arbitrary number w of statements over the various loop nests (w^g is the number of statements in loop nest g), and q is the total number of arrays in the program which are actually used. In other words

$$\begin{aligned} distance &= \left[\sum_{g=1}^l \sum_{u=1}^{w^g} \left\{ \sum_{k=1}^q \sum_{j=1}^{r_k} \sum_{i=1}^N ((\alpha_{Y_k} c_{jY_k}^{g,u} - \alpha_I) i + \alpha_{Y_k} d_{jY_k}^{g,u} + \beta_{Y_k} - \beta_I)^2 \right. \right. \\ &\quad \left. \left. + \sum_{i=1}^N ((\alpha_X a^{g,u} - \alpha_I) i + \alpha_X b^{g,u} + \beta_X - \beta_I)^2 \right\} \right]^{\frac{1}{2}} \end{aligned} \quad (3)$$

where $r_k \geq 0$ and it represents the number of terms of an array Y_k that appear on the right hand side of statement u in loop nest g , and $q \geq 1$. Note that in Equation 3, X is used for the array which appears on the *lhs* of a statement u in loop nest g , and Y_k is used for the k^{th} occurrence of an array Y which appears on the *rhs* of statement u in loop nest g , including X .

Adopting the convention that Y_1 corresponds to the array on the *lhs* of statement u in loop nest g , and accounting for the term Y_1 in r_1 , we can rewrite Equation 3 as shown below

$$\left[\sum_{g=1}^l \sum_{u=1}^{w^g} \sum_{k=1}^q \sum_{j=1}^{r_k} \sum_{i=1}^N ((\alpha_{Y_k} c_{jY_k}^{g,u} - \alpha_I) i + \alpha_{Y_k} d_{jY_k}^{g,u} + \beta_{Y_k} - \beta_I)^2 \right]^{\frac{1}{2}}. \quad (4)$$

Note that for any array Y_k for which $r_k = 1$ we can reduce its contribution to the above equation to zero by choosing

$$\alpha_I = \alpha_{Y_k} c_{1Y_k}$$

and

$$\beta_I = \beta_{Y_k} + \alpha_{Y_k} b_{1Y_k}$$

This will also be the case if $r_k > 1$ and the subscript expressions for array Y_k are always the same. In this case we can use these equations as constraints on the values of both α_I and β_I and we can also use it to impose constraints on the values of α_{Y_k} and β_{Y_k} .

In order to solve for the unknowns we require that $\alpha_{Y_k} c_{jY_k}^{g,u} - \alpha_I^{g,u} = 0$. In this way we can eliminate the terms that are multiplied by i in Equation 4 and arrive at the following equation

$$\left[\sum_{g=1}^l \sum_{u=1}^{w^g} \sum_{k=1}^q \sum_{j=1}^{r_k} \sum_{i=1}^N \left(\alpha_{Y_k} d_{jY_k}^{g,u} + \beta_{Y_k} - \beta_I \right)^2 \right]^{\frac{1}{2}}. \quad (5)$$

To solve for the unknowns in Equation 5 we will use the Lagrange Multiplier method as reported by Avriel [2], Bazaraa et al. [4], Bertsekas [5], Kuhn and Tucker [15], Pike [18], and Reklaitis et al. [20]. To minimize the Euclidean distance function shown in Equation 5 subject to the conditions $f_i(x) \leq 0, i = 1, 2, \dots, h$, and $f_i(x) = 0, i = h + 1, h + 2, \dots, m$, where $n \geq m$, the necessary conditions for the existence of a relative minimum at a point x^* are:

1. $\frac{\partial L}{\partial x_j}(x^*) + \sum_{i=1}^h \lambda_i \frac{\partial L}{\partial x_j}(x^*) + \sum_{i=h+1}^m \lambda_i \frac{\partial L}{\partial x_j}(x^*) = 0 \quad \text{for } j = 1, 2, \dots, n$
2. $f_i(x^*) \leq 0 \quad \text{for } i = 1, 2, \dots, h$
3. $f_i(x^*) = 0 \quad \text{for } i = h + 1, h + 2, \dots, m$
4. $\lambda_i f_i(x^*) = 0 \quad \text{for } i = 1, 2, \dots, h$

$$5. \lambda_i \geq 0 \quad \text{for } i = 1, 2, \dots, h$$

$$6. \lambda_i \text{ is unrestricted in sign for } i = h + 1, h + 2, \dots, m$$

where n is the number of unknowns, h is the number of inequality constraints, and m ($n \geq m$) is the total number of constraints including equality constraints, Pike [18]. The first condition sets the first partial derivatives of the Lagrangian function with respect to x_i , $i = 1, 2, \dots, n$, equal to zero to locate the Kuhn-Tucker point x^* . Conditions 2 and 3 are the inequality and equality constraints, respectively, that must be met at the minimum point found by solving the system of equations obtained from condition 1. The fourth condition comes from setting the partial derivatives of the Lagrangian with respect to the slack variables equal to zero. Condition 5 arises from the fact that the rate of change of the distance function with respect to the parameters on the *rhs* of the constraints is equal to the negative of the corresponding Lagrange multiplier. By increasing the *rhs* of a constraint the constraint region would be enlarged, which could not result in a larger value for the distance function evaluated at x^* but could result in a lower value. Thus the Lagrange multiplier must be positive to satisfy the rate of change mentioned above, Pike [18], Reklaitis [20]. Condition 6 is due to a proof that the Lagrange multipliers associated with the equality constraints are not restricted in sign, Pike [18]. Note that a new variable is added for each equality constraint and that two variables are added for each inequality constraint.

To illustrate the Lagrange Multiplier method applied to the stride alignment problem we will use the code shown below which is the same code we used in a previous example.

```

do  $i = 1, N$ 
     $X[2i - 1] = Y[3i - 1] + Y[3i] + Y[3i + 1]$ 
enddo

```

The Lagrangian function in this case is given by

$$\begin{aligned}
L &= ((\beta_X - \alpha_X - \beta_I)^2 + (\beta_Y - \beta_I)^2 \\
&+ (\beta_Y - \alpha_Y - \beta_I)^2 + (\beta_Y + \alpha_Y - \beta_I)^2)^{\frac{1}{2}} \\
&+ \lambda_1 (1 + s^2 - \alpha_I) + \lambda_2 (2\alpha_X - \alpha_I) + \lambda_3 (3\alpha_Y - \alpha_I)
\end{aligned} \tag{6}$$

The solutions that we obtain are $\lambda_1 = 0.47$, $\lambda_2 = 0$, $\lambda_3 = -0.47$, $s = 0$, $\alpha_I = 1$, $\alpha_Y = 1/3$, $\alpha_X = 1/2$, $\beta_Y = \beta_I$, and $\beta_X = \beta_I + 1/2$. Since $\lambda_1 \neq 0$, and $s = 0$ the equality holds. Note that for this example we had added two equality constraints and one inequality constraint to the Euclidean distance function in order to form the Lagrangian function. Two variables, the Lagrange multiplier and the slack variable, are added for each inequality constraint and one variable, the Lagrange multiplier, for each equality constraint. The total time that takes a IBM RISC System/6000 to solve this system of equations is 3.6 seconds. This is the computer we used for all the examples given in this work. We should point out that, for this example, we obtained several sets of possible solutions which makes for the excess time.

Say that $\alpha_I = 6$, $\alpha_X = 3$, $\alpha_Y = 2$, $\beta_I = 1$, and $\beta_X = 4$. With this alignment, each processor computing an iteration will need an element of Y which is held by the processor to its left, assuming a linear array and a block distribution with block size equal to 3, except of course for the processor at the leftmost position.

Using the owner-computes rule we obtain $\beta_X = 1$ for the same values of $\alpha_Y = 2$, $\alpha_X = 3$, and $\beta_Y = 1$. This would require each processor to send two elements to the processor on its right, except for the last processor, assuming the same configuration and distribution as before.

4 Offset Alignment

Offset alignment can be viewed as an special case of stride alignment where the stride coefficients are equal to unity.

4.1 Offset Alignment via Lagrange Multipliers

Consider the following piece of code:

```

ALIGN X[i] WITH T[i + βX]
ALIGN Y[i] WITH T[i + βY]
:
ALIGN i WITH T[i + βI]
do i = 1, N
    X[i + b1] = Y[i + d1] + Y[i + d2] + ⋯ + Y[i + dr]
enddo

```

where β_X , β_Y , and β_I are to be determined.

Consider iteration i only. We want to minimize the distance from the processor(s) holding the elements of arrays X and Y that are needed to perform the computation of the element on the left hand side to the processor which will be performing the computation during iteration i . Using the alignment specified above we find that the processor which holds the element on the *lhs* is processor $i + b_1 + \beta_X$. Similarly, the processor holding the first term of array Y is processor $i + d_1 + \beta_Y$, the one holding the second term is $i + d_2 + \beta_Y$, and so on.

This is the same problem we dealt with in Section 3.1 but with all the stride coefficients equal to one (1). Therefore, the distance function can be found from our earlier result. Thus

$$\left[\sum_{g=1}^l \sum_{u=1}^{w^g} \sum_{k=1}^q \sum_{j=1}^{r_k} \sum_{i=1}^N \left(d_{jY_k}^{g,u} + \beta_{Y_k} - \beta_I \right)^2 \right]^{\frac{1}{2}} \quad (7)$$

where l is the number of loops, w^g is the number of statements in loop nest g , q is the total number of arrays, and Y_1 corresponds to the array on the *lhs* of statement

STATE- MENT	ARRAY NAME	DIMENSION		STATE- MENT	ARRAY NAME	DIMENSION	
		1ST	2ND			1ST	2ND
S1	I	0	0	S5	I	0	0
	A	0	0		A	0	0
	B	0	1		B	1	0
	X	0	1/3		X	1/3	0
S2	I	0	0	S6	I	0	0
	A	0	0		A	0	0
	B	0	1/3		B	1/3	0
S3	I	0	-	S7	I	-	0
	B	0	-		B	-	0
	X	0	-		X	-	0
S4	I	0	0	S8	I	0	0
	A	0	-1		A	-1	0
	B	0	0		B	0	0
	X	0	-1/3		X	-1/3	0

Table 1: Constant Offsets (β 's) Found Using Lagrange Method on ADI.

u in loop nest g .

To solve for the unknowns in Equation 7 we will use the Lagrange Multiplier method as reported by Avriel [2], Bazaraa et al. [4], Bertsekas [5], Kuhn and Tucker [15], Pike [18], and Reklaitis et al. [20]. Though this method allows us to include in our system of equations a set of equality and inequality constraints we do not do it for the offset alignment case. To illustrate this method let us consider the following:

```

do  $i = 1, N$ 
   $X[i] = Y[i + 2]$ 
enddo

```

The first step is to form the function $L = ((\beta_X - \beta_I)^2 + (2 + \beta_Y - \beta_I)^2)^{\frac{1}{2}}$. By solving the corresponding system of equations, the general solution for our example would be given by $\beta_X = \beta_I$, and $\beta_Y = \beta_I - 2$.

4.1.1 ADI Using Lagrange Multipliers

The results of applying this method to the ADI program segment shown in Figure 1 are shown in Table 1.

```

do j = 2, N
  do i = 1, N
S1:      x[i, j] = F1(x[i, j], x[i, j - 1], a[i, j], b[i, j - 1])
S2:      b[i, j] = F2(b[i, j], a[i, j], b[i, j - 1])
  enddo
enddo
do i = 1, N
S3:      x[i, N] = F3(x[i, N], b[i, N])
enddo
do j = N - 1, 1, -1
  do i = 1, N
S4:      x[i, j] = F4(x[i, j], a[i, j + 1], x[i, j + 1], b[i, j])
  enddo
enddo
do j = 1, N
  do i = 2, N
S5:      x[i, j] = F5(x[i, j], x[i - 1, j], a[i, j], b[i - 1, j])
S6:      b[i, j] = F6(b[i, j], a[i, j], b[i - 1, j])
  enddo
enddo
do j = 1, N
S7:      x[N, j] = F7(x[N, j], b[N, j])
enddo
do j = 1, N
  do i = N - 1, 1, -1
S8:      x[i, j] = F8(x[i, j], a[i + 1, j], x[i + 1, j], b[i, j])
  enddo
enddo

```

Figure 1: Alternating-Direction-Implicit (ADI) Program Segment.

STATE- MENT	ARRAY	DIMENSION		STATE- MENT	ARRAY	DIMENSION		
		ONE	TWO			ONE	TWO	
S1	I	-1	0	S3	I	0	0	
	ZA	-1/2	0		ZA	0	0	
	ZB	0	0		ZB	0	0	
	ZM	0	0		ZU	0	0	
	ZP	0	0		ZV	0	0	
	ZQ	0	0		ZZ	0	0	
	ZR	0	0		S4	I	0	0
	ZU	0	0			ZA	0	0
	ZV	0	0			ZB	0	0
	ZZ	0	0			ZU	0	0
			ZV	0		0		
S2	I	-1	0	ZR	0	0		
	ZA	0	0	S5	I	0	0	
	ZB	-1/2	0		ZR	0	0	
	ZM	0	0		ZU	0	0	
	ZP	0	0	S6	I	0	0	
	ZQ	0	0		ZV	0	0	
	ZR	0	0		ZZ	0	0	
	ZU	0	0					
	ZV	0	0					
	ZZ	0	0					

Table 2: Constant Offsets (β 's) Found Using Lagrange Method on Livermore 18.

4.1.2 Livermore 18 Using Lagrange Multipliers

A program segment is shown for Livermore 18 in Figure 2. The results of applying our method are shown in Table 2.

4.1.3 Shallow Using Lagrange Multipliers

As reported by Tseng [22] Shallow is a 200 line benchmark that uses stencil computation that applies finite-difference methods to solve shallow-water equations and is a representative of a large class of existing supercomputer applications. Table 3 shows the result of applying our method to Shallow. The code for Shallow is shown in Figure 3. In Table 4 we present the time it took to obtain the results for the codes shown so far along with the result for some pieces of code not shown here for several

```

:
do l = 1, time
  do k = 2, 99
    do j = 2, 99
S1:      ZA[j, k] = F1(ZP[j - 1, k], ZQ[j - 1, k], ZM[j - 1, k],
                    ZR[j - 1, k], ZZ[j - 1, k], ZA[j - 1, k], ZU[j - 1, k],
                    ZV[j - 1, k], ZB[j - 1, k])
S2:      ZB[j, k] = F2(ZP[j - 1, k], ZQ[j - 1, k], ZM[j - 1, k],
                    ZR[j - 1, k], ZZ[j - 1, k], ZA[j - 1, k], ZU[j - 1, k],
                    ZV[j - 1, k], ZB[j - 1, k])
    enddo
  enddo
  do k = 2, 99
    do j = 2, 99
S3:      ZU[j, k] = F3(ZZ[j - 1, k], ZZ[j + 1, k], ZA[j - 1, k],
                    ZA[j + 1, k], ZU[j - 1, k], ZU[j + 1, k], ZV[j - 1, k],
                    ZV[j + 1, k], ZB[j - 1, k], ZB[j + 1, k])
S4:      ZV[j, k] = F4(ZZ[j - 1, k], ZZ[j + 1, k], ZA[j - 1, k],
                    ZA[j + 1, k], ZU[j - 1, k], ZU[j + 1, k], ZV[j - 1, k],
                    ZV[j + 1, k], ZB[j - 1, k], ZB[j + 1, k])
    enddo
  enddo
  do k = 2, 99
    do j = 2, 99
S5:      ZR[j, k] = F5(ZR[j, k], ZU[j, k])
S6:      ZZ[j, k] = F6(ZZ[j, k], ZV[j, k])
    enddo
  enddo
enddo

```

Figure 2: Livermore 18 Program Segment.

STATE- MENT	ARRAY NAME	DIMENSION		STATE- MENT	ARRAY NAME	DIMENSION		
		1ST	2ND			1ST	2ND	
S1	I	1	0	S7	U	-1/2	0	
	U	0	0		V	0	-1/2	
	PSI	0	-1/2		I	1	0	
S2	I	0	1		UNEW	0	0	
	V	0	0		UOLD	0	0	
	PSI	-1/2	0		Z	0	-1/2	
S3	I	1	0		CV	1/2	-1/2	
	CU	0	0		H	1/2	0	
	P	1/2	0		S8	I	0	1
	U	0	0			VNEW	0	0
S4	I	0	1	VOLD		0	0	
	CV	0	0	Z		-1/2	0	
	P	0	1/2	CU		-1/2	1/2	
	V	0	0	H		0	1/2	
S5	I	1	1	S9		I	0	0
	Z	0	0			PNEW	0	0
	V	1/2	0		POLD	0	0	
	U	0	1/2		CU	-1/2	0	
	P	1/2	1/2		CV	0	-1/2	
S6	I	0	0					
	H	0	0					
	P	0	0					

Table 3: Constant Offsets (β 's) Found Using Lagrange Method on Shallow.

well known benchmarks.

5 Axis Alignment

Axis alignment arises when we have a dimension permutation in the alignment statement for multidimensional arrays. For example, in the statement

ALIGN $X[i, j]$ WITH $T1[j, i]$

we have that each row i of array X is aligned with column i of template $T1$. Likewise, each column j of X is aligned with row j of template $T1$. In other words, the first dimension of array X is aligned with the second dimension of template $T1$, and the

```

:
do j = 1, N - 1
  do i = 1, N - 1
S1:      u[i + 1, j] =  $\mathcal{F}_1(psi[i + 1, j + 1], psi[i + 1, j])$ 
S2:      v[i, j + 1] =  $\mathcal{F}_2(psi[i + 1, j + 1], psi[i, j + 1])$ 
      enddo
  enddo
:
do j = 1, N - 1
  do i = 1, N - 1
S3:      cu[i + 1, j] =  $\mathcal{F}_3(p[i + 1, j], p[i, j], u[i + 1, j])$ 
S4:      cv[i, j + 1] =  $\mathcal{F}_4(p[i, j + 1], p[i, j], u[i, j + 1])$ 
S5:      z[i + 1, j + 1] =  $\mathcal{F}_5(v[i + 1, j + 1], v[i, j + 1], u[i + 1, j + 1],$ 
      u[i + 1, j], p[i, j], p[i + 1, j], p[i + 1, j + 1], p[i, j + 1])
S6:      h[i, j] =  $\mathcal{F}_6(p[i, j], u[i + 1, j], u[i, j], v[i, j + 1], v[i, j])$ 
      enddo
  enddo
:
do j = 1, N - 1
  do i = 1, N - 1
S7:      unew[i + 1, j] =  $\mathcal{F}_7(uld[i + 1, j], z[i + 1, j + 1], z[i + 1, j],$ 
      cv[i + 1, j + 1], cv[i, j + 1], cv[i, j], cv[i + 1, j],
      h[i + 1, j], h[i, j])
S8:      vnew[i, j + 1] =  $\mathcal{F}_8(vold[i, j + 1], z[i + 1, j + 1], z[i, j + 1],$ 
      cu[i + 1, j + 1], cu[i, j + 1], cu[i, j], cu[i + 1, j],
      h[i, j + 1], h[i, j])
S9:      pnew[i, j + 1] =  $\mathcal{F}_9(pold[i, j], cu[i + 1, j], cu[i, j],$ 
      cv[i, j + 1], cv[i, j])
      enddo
  enddo

```

Figure 3: Shallow Program Segment.

PROGRAM	TIME
Jacobi	0.18
ADI	1.67
Disper	0.23
Livermore 18	5.98
Livermore 23	0.12
Red Black SOR	0.24
Shallow	1.96

Table 4: Time (seconds) to solve the system of equations for the different applications.

second dimension of X is aligned with the first dimension of $T1$. Other examples are

ALIGN $Y[j, i]$ WITH $T2[i, j]$

where the first dimension of array Y is aligned with the second dimension of template $T2$, and

ALIGN $Z[k, j, i]$ WITH $T3[i, j, k]$

where the first, second, and third dimension of array Z are aligned with the third, second, and first dimension of template $T3$, respectively.

5.1 Axis Alignment Problem Formulation

Consider the following code segment.

```

do  $i = 1, N$ 
  do  $j = 1, N$ 
     $X[i, j] = Y[i, j] + Y[j, i]$ 
  enddo
enddo

```

and assume that the alignment directives will be

ALIGN $X[i, j]$ WITH $T[\alpha_X^1 i + \beta_X^1, \alpha_X^2 j + \beta_X^2]$

ALIGN $Y[i, j]$ WITH $T[\alpha_Y^1 i + \beta_Y^1, \alpha_Y^2 j + \beta_Y^2]$

and

ALIGN i, j WITH $T[\alpha_I^1 i + \beta_I^1, \alpha_I^2 j + \beta_I^2]$.

Using the l_2 metric the total distance from the processor performing iteration i, j to the processors holding each of the operands would be given by

$$\begin{aligned}
distance &= \left((\alpha_X^1 - \alpha_I^1) i + \beta_X^1 - \beta_I^1 \right)^2 + \left((\alpha_X^2 - \alpha_I^2) j + \beta_X^2 - \beta_I^2 \right)^2 \\
&+ \left((\alpha_Y^1 - \alpha_I^1) i + \beta_Y^1 - \beta_I^1 \right)^2 + \left((\alpha_Y^2 - \alpha_I^2) j + \beta_Y^2 - \beta_I^2 \right)^2 \\
&+ \left(\alpha_Y^1 j - \alpha_I^1 i + \beta_Y^1 - \beta_I^1 \right)^2 + \left(\alpha_Y^2 i - \alpha_I^2 j + \beta_Y^2 - \beta_I^2 \right)^2. \quad (8)
\end{aligned}$$

Note the last two terms of Equation 8 above. There is not a non-trivial way of eliminating the i and the j from the above equation when we consider all the possible values that i , and j can take on.

Here the problem is not the metric but the actual mapping. We have a mapping from a two-dimensional array space and a two-dimensional iteration space to a two-dimensional template space, and because of the nature of the problem itself, this mapping approach we have used so far is not very useful for this particular problem. However, consider the same problem but with a different mapping. In particular, consider that arrays X , and Y will be aligned using

$$\begin{aligned}
&\text{ALIGN } X[i, j] \text{ WITH } T[\alpha_X^1 i + \beta_X^1 + \alpha_X^2 j + \beta_X^2] \\
&\text{ALIGN } Y[i, j] \text{ WITH } T[\alpha_Y^1 i + \beta_Y^1 + \alpha_Y^2 j + \beta_Y^2]
\end{aligned}$$

and iteration i, j using

$$\text{ALIGN } i, j \text{ WITH } T[\alpha_I^1 i + \beta_I^1 + \alpha_I^2 j + \beta_I^2]$$

so that the two-dimensional array space and the two-dimensional iteration space are mapped onto a one-dimensional template. The distance function will then be given by

$$\begin{aligned}
distance &= \left[\left((\alpha_X^1 - \alpha_I^1) i + (\alpha_X^2 - \alpha_I^2) j + \beta_X^1 - \beta_I^1 + \beta_X^2 - \beta_I^2 \right)^2 \right. \\
&+ \left. \left((\alpha_Y^1 - \alpha_I^1) i + (\alpha_Y^2 - \alpha_I^2) j + \beta_Y^1 - \beta_I^1 + \beta_Y^2 - \beta_I^2 \right)^2 \right]
\end{aligned}$$

$$+ \left[\left(\alpha_Y^2 - \alpha_I^1 \right) i + \left(\alpha_Y^1 - \alpha_I^2 \right) j + \beta_Y^1 - \beta_I^1 + \beta_Y^2 - \beta_I^2 \right]^{\frac{1}{2}}. \quad (9)$$

We can require that $\alpha_X^1 = \alpha_I^1 = \alpha_X^2 = \alpha_I^2 = \alpha_Y^1 = \alpha_Y^2$ and thus Equation 9 would be reduced to

$$\begin{aligned} distance &= \left[\left(\beta_X^1 - \beta_I^1 + \beta_X^2 - \beta_I^2 \right)^2 + \left(\beta_Y^1 - \beta_I^1 + \beta_Y^2 - \beta_I^2 \right)^2 \right. \\ &\quad \left. + \left(\beta_Y^1 - \beta_I^1 + \beta_Y^2 - \beta_I^2 \right)^2 \right]^{\frac{1}{2}} \end{aligned} \quad (10)$$

which can be reduced to zero distance by allowing $\beta_X^1 = \beta_I^1 + \beta_I^2 - \beta_X^2$ and $\beta_Y^1 = \beta_I^1 + \beta_I^2 - \beta_Y^2$ which is the result we obtain when using our method. This means that each diagonal of arrays X and Y , and each diagonal of the iteration space would be mapped to a point in the one-dimensional template. The above result was arrived at in 0.11 seconds using our method to solve the system of equations.

6 Conclusions

We presented a new method to solve the computation and data alignment problems. Our method uses the Euclidean metric to model the distance function between the processor executing an iteration and the processors holding the data items to determine the best placement of data and of computation. The owner-computes rule is thus relaxed. We also use the Lagrange Multiplier method to add constraints to our distance function and determine a system of simultaneous equations that must be satisfied for the existence of a relative minimum at any specific point.

Solutions were presented for cases which include stride, offset, and axis alignment. These solutions include several real life applications. Our method is meant to be used as a tool which is not part of the compiler.

References

- [1] S.P. Amarasinghe, J. M. Anderson, M.S. Lam, and A.W. Lim. An overview of a compiler for scalable parallel machines. In *Proceedings of the Sixth Annual Workshop on Languages and Compilers for Parallel Computing*, Portland, OR, August 1993.
- [2] M. Avriel. *Nonlinear Programming, Analysis and Methods*. Prentice-Hall, Inc, Englewood Cliffs, N.J., 1976.
- [3] D. Bau, I. Kodukula, V. Kotlyar, K. Pingali, and P. Stodghill. Solving alignment using elementary linear algebra. In K. Pingali, U. Banerjee, D. Gelernter, A. Nicolau, and D. Padua, editors, *Languages and Compilers for Parallel Computing: Seventh International Workshop*. Springer-Verlag, 1994.
- [4] M.S. Bazaraa, H.D. Sherali, and C.M. Shetty. *Nonlinear Programming: Theory and Algorithms*. John Wiley & Sons, Inc., New York, 1993.
- [5] D.P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. Academic Press, Inc., New York, 1982.
- [6] S. Chatterjee, John R. Gilbert, Robert Schreiber, and Shang-Hua Teng. Optimal evaluation of array expressions on massively parallel machines. Technical Report TR 92.17, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffett Field, CA, September 1992. Also available as Xerox PARC Technical Report CSL-92-11. Submitted to *ACM Transactions on Programming Languages and Systems*.
- [7] S. Chatterjee, John R. Gilbert, Robert Schreiber, and Shang-Hua Teng. Automatic array alignment in data-parallel programs. In *Proceedings of the Twentieth Annual ACM SIGACT/SIGPLAN Symposium on Principles of Programming*

Languages, pages 16–28, Charleston, SC, January 1993. Also available as RIACS Technical Report 92.18 and Xerox PARC Technical Report CSL-92-13.

- [8] Y.-C. Chung, C.-H. Hsu, S.-W. Bai. A Basic-Cycle Calculation Technique for Efficient Dynamic Data Redistribution. *IEEE Transactions on Parallel and Distributed Systems*, 9(4):359–377, April 1998.
- [9] F. Desprez, J. Dongarra, A. Petite, C. Randriamaro, Y. Robert. Scheduling Block-Cyclic Array Redistribution. *IEEE Transactions on Parallel and Distributed Systems*, 9(2):192–205, February 1998.
- [10] M. Gupta and P. Banerjee. Demonstration of automatic data partitioning techniques for parallelizing compilers on multicomputers. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):179–193, March 1992.
- [11] C.-H. Huang and P. Sadayappan. Communication-free hyperplane partitioning of nested loops. *Journal of Parallel and Distributed Computing*, 19(2):90–102, October 1993.
- [12] E.T. Kalns and L.M. Ni. Processor Mapping Techniques Toward Efficient Data Redistribution. *IEEE Transactions on Parallel and Distributed Systems*, 6(12):1234–1247, December 1995.
- [13] M. Kandemir, A. Choudhary, N. Shenoy, P. Banerjee, J. Ramanujam. A Linear Algebra Framework for Automatic Determination of Optimal Data Layouts. *IEEE Transactions on Parallel and Distributed Systems*, 10(2):115–135, February 1999.
- [14] I. Kim and M. Wolfe. Communication analysis for multicomputer compilers. In *Proceedings of Parallel Architectures and Compilation Techniques (PACT 94)*, aug 1994.

- [15] H.W. Kuhn and A.W. Tucker. Nonlinear programming. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492, 1951.
- [16] W. Li and K. Pingali. Access normalization: Loop restructuring for numa compilers. In *Proceedings Fifth International Conf. on Architectural Support for Programming Languages and Operating Systems*, pages 285–295, Boston, MA, October 1992.
- [17] M. O’Boyle. A data algorithm for distributed memory compilation. In *Proceedings of the Sixth International PARLE Conference*, Athens, Greece, July 1994. Springer-Verlag.
- [18] R.W. Pike. *Optimization for Engineering Systems*. Van Nostrand Reinhold Company, New York, 1986.
- [19] J. Ramanujam and P. Sadayappan. Compile-time techniques for data distribution in distributed memory machines. *IEEE Transactions on Parallel and Distributed Systems*, 2(4):472–482, October 1991.
- [20] G.V. Reklaitis, A. Ravindran, and K.M. Ragsdell. *Engineering Optimization: Methods and Applications*. John Wiley & Sons, Inc., New York, 1983.
- [21] R. Thakur, A. Choudhary, J. Ramanujam. Efficient Algorithms for Array Redistribution. *IEEE Transactions on Parallel and Distributed Systems*, 7(6):587–594, June 1996.
- [22] C.-W. Tseng. *An Optimizing Fortran D Compiler for MIMD Distributed-Memory Machines*. PhD thesis, Department of Computer Science, Rice University, Houston, TX, January 1993. Available as technical report CRPC-TR93291.

- [23] A. Wakatani and M. Wolfe. A new approach to array redistribution: Strip mining redistribution. In *Proceedings of the Sixth International PARLE Conference*, Athens, Greece, July 1994. Springer-Verlag.
- [24] S. Wolfram. *Mathematica: A System for Doing Mathematics by Computer*. Addison-Wesley Publishing Company, 1988.

Isidoro Couvertier-Reyes is an Assistant Professor at the College of Engineering at the University of Puerto Rico-Mayaguez. He is a member of the IEEE, the ACM, the IEEE Computer Society, and the CIAPR.