

appeared in *Journal of Parallel Algorithms and Applications*, vol. 2, no. 1, 1994, pp. 5-26

Finding Optimum Wavefront of Parallel Computation*

Balaram Sinharoy

Boleslaw Szymanski

Enterprise Systems Division Department of Computer Science
IBM Corporation, P.O. Box 950 Rensselaer Polytechnic Institute
Poughkeepsie, NY 12602 Troy, NY 12180-3590

Keywords: parallel programming, data dependence, algorithm transformation, uniform dependence algorithm, compile-time scheduling, hyperplane.

Computer Review Categories: D.1.3. parallel programming, D.3.4. compilers, D.4.1. scheduling.

Abstract

Data parallelism, in which the same operation is performed on many elements of an n -dimensional array, is one of the most powerful methods of extracting parallelism in scientific computation. One form of data parallelism involves defining a sequence of parallel wavefronts of a computation. Each wavefront consists of an $(n - 1)$ -dimensional subarray of the evaluated array and all wavefront elements are evaluated simultaneously. Different wavefronts result in different performance, so the question arises how to determine the wavefronts that result in the minimum computation time. Wavefront determination should define also allocation of wavefront elements to processors.

In this paper we present efficient algorithms for determining the optimum wavefront and for partitioning it into sections assigned to individual processors. Presented algorithms are applicable to computations that are defined over two or higher dimensional arrays and are executed on distributed memory machines interconnected into a one or two-dimensional processor array.

1 Introduction

Scientific and engineering computations, although numerically intensive, are typically regular both in terms of the control flow patterns and the employed data structures. Such computations often involve iterative applications of numerical algorithms to all (or the majority of) the parts of

*This work sponsored in part by IBM Corp under the Development Grant, by NSF under grant CCR-8920694 and by ONR under grant N00014-93-1-0076. The content of the information does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred.

the data structures, a characteristic very suitable for massively parallel processing. Algorithms that employ such iterations are referred to here as iterative algorithms. In the spacetime representation of a computation described by an iterative algorithm, the values of different variables are to be evaluated at different index points [6]. For correct execution of the algorithm, the evaluation order must be in accordance with the various dependence relationships among the index points. In most scientific computations, variables can be evaluated at multiple index points simultaneously, resulting in a very high speedup of the computation.

In a program written in a single assignment language [9] (or a conventional language, where each variable occurrence is labeled with the appropriate indexes of the surrounding loops [3]), a *data dependence* vector can be viewed as the difference of indexes where a variable is used and where that variable is generated. The set of index points along with the set of dependence vectors define a *computational structure* [11]. If $D = \{\bar{d}_1, \bar{d}_2, \dots, \bar{d}_k\}$ is the set of dependence vectors in an algorithm, then \bar{i}_1 is *dependent* on \bar{i}_2 , if and only if $\bar{i}_2 = \bar{i}_1 + a_1\bar{d}_1 + a_2\bar{d}_2 + \dots + a_k\bar{d}_k$, where $a_l \in \mathbb{Z}_{>0}$, $1 \leq l \leq k$. Variables can be evaluated simultaneously at \bar{i}_1 and \bar{i}_2 , if and only if they are independent. If the data dependence vectors in an algorithm are all constants, then an n -deep loop nest has at least $n - 1$ degrees of parallelism.

In the classic paper [2], Lamport proposed two methods, the hyperplane and the coordinate method, for finding the set of index points at which evaluation can proceed simultaneously. In the hyperplane method, Lamport considered partitioning the set of index points into a set of parallel hyperplanes. Variables can be executed simultaneously at all index points that lie on a single hyperplane. If \bar{h} is a vector (h_1, \dots, h_p) , then equation $h_1x_1 + h_2x_2 + \dots + h_kx_k = c$ for different values of c defines a set of parallel hyperplanes. Variables can be executed at all index points on a hyperplane \bar{h} , if and only if $\bar{h} \cdot \bar{d}_l > 0$ for all dependence vectors \bar{d}_l . To find the best hyperplane \bar{h}_{opt} , we need to solve the integer programming problem (which is NP-hard):

$$\bar{h}_{opt} = \min_{\bar{h}} \{ \max \{ |\bar{h}(\bar{i}_1 - \bar{i}_2)| \mid \bar{i}_1, \bar{i}_2 \in \mathbf{I} \} \}$$

where \mathbf{I} is the *index domain*, i.e. the set of all index points in the computation (the index domain is usually defined by the loop control variables or by separate statements [13]).

Lamport's original results have been extended by many researchers over the years [7, 11, 3, 4]. Moldovan et al [7] considered a linear transformation, \mathbf{T} , of the algorithm to map it efficiently on a VLSI processor array. The linear transformation consists of two parts: space hyperplanes \mathbf{S} and time hyperplanes \bar{h} , i.e.,

$$\mathbf{T} = \begin{bmatrix} \bar{h} \\ \mathbf{S} \end{bmatrix}$$

where mapping \mathbf{S} determines the processor at which an index point will be evaluated and mapping \bar{h} determines the time of evaluation. For example, for \bar{h} as before, and

$$\mathbf{S} = \begin{pmatrix} s_{11} & \dots & s_{1p} \\ \dots & \dots & \dots \\ s_{q1} & \dots & s_{qp} \end{pmatrix}$$

index point $\bar{i} = (i_1, \dots, i_p)$ is mapped onto the q -dimensional processor array as

$$\begin{pmatrix} \bar{h} \\ \mathbf{S} \end{pmatrix} (i_1, \dots, i_p)^t = \begin{pmatrix} h_1 & \dots & h_p \\ s_{11} & \dots & s_{1p} \\ & \dots & \\ s_{q1} & \dots & s_{qp} \end{pmatrix} (i_1, \dots, i_p)^t = \begin{pmatrix} h_1 i_1 + \dots + h_p i_p \\ s_{11} i_1 + \dots + s_{1p} i_p \\ \dots \\ s_{q1} i_1 + \dots + s_{qp} i_p \end{pmatrix} = \begin{pmatrix} t \\ l_1 \\ \dots \\ l_q \end{pmatrix}$$

that is, index point \bar{i} is evaluated at time t at processor location (l_1, \dots, l_q) . Again for correct execution ordering, we should have

$$\bar{h}\bar{d} > 0 \quad \forall \bar{d} \in D \quad (1)$$

and for the minimum execution time, mapping \bar{h} should minimize

$$t = \left\lceil \frac{\max \bar{h}(\bar{i}_1 - \bar{i}_2) + 1}{\min \bar{h}\bar{d}_i} \right\rceil$$

for any $\bar{i}_1, \bar{i}_2 \in I$ and $\bar{d}_i \in D$.

Lee et al [4], stated the necessary and sufficient conditions for such a transformation \mathbf{T} to correctly map a p -nested loop algorithm onto a q -dimensional systolic array, where $1 \leq q \leq p-1$. They have also described exhaustive and heuristic search methods to determine, respectively, the optimal and suboptimal mappings \bar{h} and \mathbf{S} . Sheu et al [11] describe an algorithm to partition the index points found on a time hyperplane onto the processors in a message passing system. Their method first projects all index points and dependence vectors onto hyperplane $\bar{h}x = 0$. The projected index points are then grouped along the directions of the projected dependence vectors. The result is that all index points projected onto a point in the same group will not be executed simultaneously. To reduce the communication overhead, the group is made as large as possible.

Most methods described in the literature for mapping a computation structure onto an array of processors (or a VLSI array) reduce the problem to an instance of the integer programming optimization. However, the integer programming optimization problem is NP-hard [1]. In this paper, we assume the spacetime representation of an algorithm to be a continuous domain and determine the time hyperplane \bar{h} , termed *wavefront*, that results in the minimum execution time. In our analysis, an algorithm is assumed to be specified using single assignment statements, i.e., each variable is defined no more than once at each index point. To apply the analysis to an algorithm specified using a procedural language, appropriate indexes of the loops surrounding each variable occurrence in the algorithm are needed (a process labeling variables with such indexes is described in [3]).

Wavefront scheduling is appropriate for Single Program Multiple Data (SPMD) [10] implementation on distributed memory architecture or for data parallelism on SIMD architectures. SPMD implementation, in general, requires larger parallel granules than SIMD implementation, therefore it is more efficient provided that the computations at each index point are fairly complex (i.e., involve computationally intensive function evaluation).

Figure 1 illustrates how the choice of a particular wavefront can affect the performance of an algorithm. A two dimensional array E is to be evaluated on a one-dimensional (logically) processor array. The elements are defined by the following statements (elements that are beyond the array boundary are considered to be zero)

$$\forall (x_1, x_2) \in \mathbf{X} : \quad E[x_1, x_2] = f(E[x_1 - 2, x_2 + 2], E[x_1 - 4, x_2 - 2]) \quad (2)$$

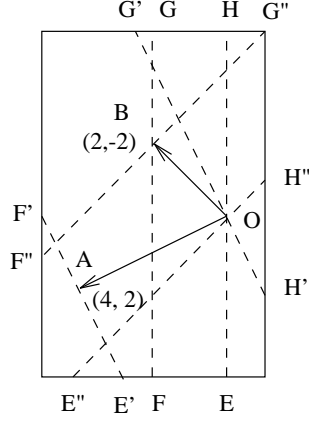


Figure 1: Different wavefronts to evaluate array E

where the index domain \mathbf{X} is $[1, n] \times [1, n]^1$.

There are two dependence vectors in this computation, OA ($[4, 2]$) and OB ($[2, -2]$). Since all dependence vectors are on one side of the lines EH , $E'H'$ and $E''H''$, all of them are potential wavefronts. However, such a wavefront does not always exist. For example, when data dependences are different at different regions of index domain, there may be no single wavefront with the required property in the entire index domain. All index points lying on hyperplane $\bar{h}\bar{x} = c$ can be executed simultaneously, if and only if, values of the array at all the index points

$$X' = \{\bar{x} | \bar{h}\bar{x} < c \ \& \ \bar{x} \in \mathbf{X}\}$$

are known at this instance of execution, that is, all array elements in an appropriate side of the wavefront have been already evaluated. Two parallel wavefronts form a strip of computation that can be divided among a number of processors for execution. The separation between the wavefronts can be made such that once all packets (containing array elements evaluated by other processors) reach their destination, no more communication is needed to complete the evaluation of all the array elements between the two wavefronts. In fig. 1, $EFGH$, $E'F'H'G'$ and $E''F''H''G''$ are three such strips. Since $EFGH$ covers a bigger area than $E'F'H'G'$, computation along this wavefront results in less frequent communication and synchronization. Wavefront EH can be preferred to $E''H''$ for another reason, namely the smaller distance that data must travel (compare projection of OA on $E''H''$ with the projection of OA on EH). Wavefront EH can be partitioned into more sections than $E''H''$ with the similar computation to communication ratio, leading to a higher degree of parallelism.

Even if there are no restrictions on the number of available processors, it is not straightforward to determine how the wavefronts should be optimally partitioned and mapped to the processors. Partitioning a wavefront into small sections increases communication time, because most of the input array elements needed to evaluate a particular index point may reside outside the evaluating

¹more conventional definition of this index domain would be a loop nest $for(x_1 = 1; x_1 \leq n; x_1++) \ for(x_2 = 1; x_2 \leq n; x_2++) \dots$

processor’s local memory. For certain dependence vectors and the sizes of the partitions, input array elements may be quite a few processors away. On the other hand the processors may be underutilized, if a large section of the wavefront is assigned to a single processor. Given a nested loop, algorithms for partitioning the index domain among a number of processors to minimize the total computation time involve solving an integer optimization problem that is NP-hard [8, 14]. In this paper, we discuss polynomial-time algorithms for such partitioning under the simplifying assumption about positions of the index points versus wavefronts.

The paper is organized as follows. In section 2 we describe an algorithm for finding the optimum wavefront in a computation defined over two-dimensional arrays executed on a machine with processors interconnected (logically) into one-dimensional array. We describe also an algorithm for optimum partitioning of the wavefront into sections that are assigned to individual processors (after slight modifications, this algorithm can be applied to processor arrays of higher dimensions). Finally, we discuss an algorithm for efficient enumeration of points that are evaluated by a single processor in one computation step. In section 3, we outline the methods for determining good wavefronts for (logically) higher dimensional distributed memory machines. Conclusions and comments on limitation of the presented approach are presented in section 4.

2 Optimum Wavefront for 1-D Interconnected Architectures

Suppose we want to compute an array E defined recursively by equation (2). Such a definition imposes a partial order on the execution of the index points of the array E , and therefore not all elements of the array E can be evaluated simultaneously. In this section we investigate several valid execution orderings of the index points. Our goal is to find the execution ordering that results in the minimum computation time.

In this section, we restrict our attention to algorithms involving two dimensional arrays processed on a linear, arbitrary large array of processors. Figure 1 shows the processing of array E . If at some point during execution, $E[x_1, x_2]$ is known for all index points $[x_1, x_2]$ located above hyperplane EH , then array E can be evaluated simultaneously at all index points on this hyperplane. Such a hyperplane is termed a *wavefront of computation*. The *direction of computation* is perpendicular to the wavefront of computation. A necessary and sufficient condition for a hyperplane to be a wavefront is that all dependence vectors attached to a point on the hyperplane must lie to one side of the wavefront. More formally, if $\mathbf{D} = \{\bar{d}_1, \bar{d}_2, \dots, \bar{d}_n\}$ is the set of all dependence vectors in an algorithm, then $\bar{h} \cdot \bar{x} = c$ is a wavefront if and only if, either

$$\bar{h}(\bar{x} + \bar{d}_i) < \bar{h} \cdot \bar{x}, \quad \forall \bar{d}_i \in \mathbf{D} \ \& \ \forall \bar{x} \in \mathbf{X}$$

or

$$\bar{h}(\bar{x} + \bar{d}_i) > \bar{h} \cdot \bar{x}, \quad \forall \bar{d}_i \in \mathbf{D} \ \& \ \forall \bar{x} \in \mathbf{X}$$

is true (these two conditions are equivalent to the condition in (1)). The direction of computation is given by $\frac{\bar{h}}{\sqrt{\bar{h} \cdot \bar{h}}}$ in the former case and by $-\frac{\bar{h}}{\sqrt{\bar{h} \cdot \bar{h}}}$ in the later. Evidently, any line between OB and OA (traversed clockwise) in Figure 1 may be a wavefront, since for these and only these lines are the dependence vectors on one side of the line. We would like to determine the wavefront that results in minimum computation time.

Suppose there are n dependence vectors, $\mathbf{D} = \{\bar{d}_1, \bar{d}_2, \dots, \bar{d}_n\}$. The projections of the dependence vectors on hyperplane $\bar{h} \cdot \bar{x} = c$ are given by

$$\bar{z}_k = \bar{d}_k - \frac{\bar{h} \cdot \bar{d}_k}{\bar{h} \cdot \bar{h}} \bar{h} \quad \forall \bar{d}_k \in \mathbf{D}$$

Let $r + 1$ of the dependence vectors extend to the right and $l = n - r + 1$ to the left. Let z_i , $i = -l, -l + 1, \dots, -1$ and z_i , $i = 0, 1, 2, \dots, r$ be the projections of the vectors that extend to the left and to the right, respectively. Without loss of generality we can assume that z_{-l} and z_r are the largest projections extended to the left and to the right, respectively, and that $z_{-l} \geq z_{-l+1} \geq \dots \geq z_{-1}$ and $z_r \geq z_{r-1} \geq \dots \geq z_0$ (see Figure 2).

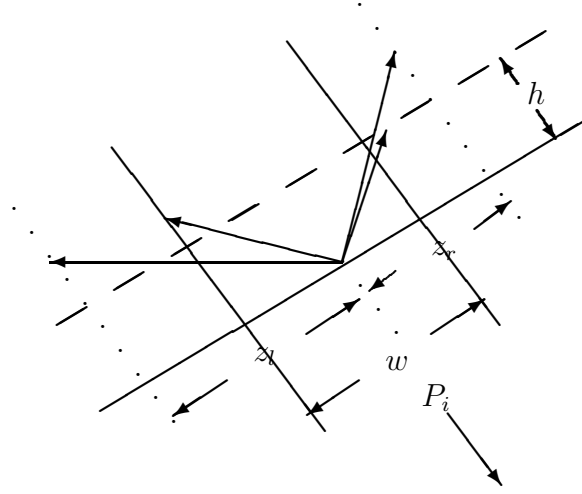


Figure 2: Projection of the dependence vectors on a wavefront

Let α (α') and β (β') denote the minimum and maximum angles made by the dependence vectors with the positive (negative) x -axis. For a wavefront to exist, either $\beta < \alpha + \pi$ or $\beta' < \alpha' + \pi$. Without loss of generality we assume in the following that $\beta < \alpha + \pi$. Let θ denote an angle that a wavefront makes with positive x -axis. Depending on the values of α, β the angle θ is in one of the five possible ranges: (i) in $(\beta, \alpha + \pi)$, or (ii) in $(-\pi/2, \alpha) \cup (\beta, \pi/2]$, or (iii) in $(\beta - \pi, \alpha)$, or (iv) in $(-\pi/2, \alpha - \pi) \cup (\beta - \pi, \pi/2]$, or (v) in $(-\pi/2, \alpha + \pi) \cup (\beta + \pi, \pi/2]$. By properly redirecting and/or relabeling x and y axes, we can, without loss of generality, restrict our attention just to the first case. Hence, we consider in the further analysis only such angles α, β, θ that $-\pi/2 < \beta < \theta < \alpha + \pi \leq \pi/2$.

Let c be the communication cost per item (which can either be an individual data element or a packet of data elements, depending on the hardware communication mechanism) and e be the execution cost of one data element. We assume that communication and computation cannot be interwoven and that there is no restriction on the number of available processors. The wavefront execution is characterized by two parameters. The first one, h , defines the distance between two consecutive parallel hyperplanes. All index points within these two hyperplanes are evaluated in one execution step. Such collection of index points is called a *computational strip*. We require that the partitions of the computational strips allocated to processors are of

rectangular shape. The second parameter describing wavefront computation is w , the width of the partition allocated to single processor. Under these assumptions, the execution time of a computation step is

$$E \leq cC(w) + N(w)e \quad (3)$$

where $C(w)$ denotes the number of items that needs to be communicated to compute one partition of the computational strip and $N(w)$ stands for the maximum number of index points in the strip. We assume that the computation is synchronized at each wavefront movement and under this assumption the right hand side of the above inequality defines the computation time between wavefront movements. To approximate the maximum number of index points within a partition of a computational strip assume that $0 \leq \theta < \pi/4$ (all other cases with $\theta \in [-\pi/2, -\pi/4), [-\pi/4, 0)$ and $(\pi/4, \pi/2)$ can be analyzed similarly). A strip of thickness $\cos \theta$ cuts out a unit along the x -axis, hence a partition of length w of this strip contains at most $\lfloor w \cos \theta \rfloor + 1$ index points. Consequently, a partition of length w of a strip with thickness h can contain at most

$$(\lfloor w \cos \theta \rfloor + 1) \left(\left\lfloor \frac{h}{\cos \theta} \right\rfloor + 1 \right)$$

points. Hence

$$N(w) \leq hw \left(1 + \frac{\cos \theta}{h} + \frac{1}{w \cos \theta} + \frac{1}{hw} \right)$$

Similarly, we can consider a partition of length $\sin \theta$ that cuts out a unit along the y -axis to arrive at approximation

$$N(w) \leq hw \left(1 + \frac{\sin \theta}{w} + \frac{1}{h \sin \theta} + \frac{1}{hw} \right)$$

that should be used when $w \leq h$. Both cases of relation of w to h are analyzed analogously, so without loss of generality we can assume that $w > h$, then

$$N(w) \approx hw \left(1 + \frac{\cos \theta}{h} \right) \quad (4)$$

Let's consider the third factor in (4), i.e. $f = 1 + \cos \theta/h$. The thickness of the computational strip is defined by the vector with the shortest projection on the wavefront line. When θ is close to β ($\alpha + \pi$) then the vector that makes an angle β (α) with x -axis has this property. In general, $h = v_h \sin(\theta - \gamma_h)$, where v_h denotes the length of the dependence vector that defines strip height h , and γ_h denotes the angle that this vector makes with the x -axis. Hence,

$$f(\theta) = 1 + \frac{\cos(\theta)}{v_h \sin(\theta - \gamma_h)} \quad f'(\theta) = \frac{-\cos(\gamma_h) \cos(\theta)}{v_h \sin^2(\theta - \gamma_h)}$$

so f is monotonically decreasing with the minimum at the upper boundary of the validity interval. If $\cos(\theta)/h \geq 1$ then $h \leq \cos \theta$ and a single wavefront movement cannot reach the next layer of index points. In such case, the number of index points that belongs to different strips in the same wavefront vary widely causing imbalance in the processor load. To avoid such imbalance, we will restrict the possible wavefront angles to the interval $[\theta_\beta, \theta_\alpha]$ in which vectors with angles α and β define strip height $h \geq \cos \phi$. Since, by definition, $\cos(\theta_\beta)/v_\beta \sin(\theta_\beta - \beta) = 1$, then

$$\theta_\beta = \arctan \left(\tan \beta - \frac{1}{v_\beta \cos \beta} \right) \quad (5)$$

with the similar definition of θ_α . If $\theta_\alpha < \theta_\beta$ then the optimum wavefront direction should be selected to minimize function f , i.e. at the equal distance between β and $\alpha + \pi$, hence $\theta_{opt} = (\beta + \alpha + \pi)/2$ (since the parallel to this wavefront crosses endpoints of two vectors, the slope of this wavefront is rational). Otherwise, since $\cos(\theta)/h < 1$, we can ignore changes in f and approximate $N(w)$ by hw .

In the following, we determine the optimum wavefront of computation for two different models of communication: (i) each array element transferred individually, and (ii) array elements transferred in packets.

2.1 Elements Transferred Individually

In this subsection it is assumed that two processors can communicate directly with each other only when they are neighbors. Under this assumption, the minimum width w along wavefront Z that a processor can be allowed to compute is $w_{min} = \max(z_{-l}, z_r)$.

For any partition of width $w \geq w_{min}$, the total number of elements communicated from the neighboring partitions can be estimated as follows. Since array elements computed by a neighboring processor are accessed individually, the total number of elements communicated from the processor on the right to the given partition is:

$$C_r(w) = N(z_0) * (r + 1) + N(z_1 - z_0) * r \cdots + N(z_r - z_{r-1})$$

Approximating the number of points in the strip fragment of length z by the average density of points in the partition we get $N(z) = z * N(w)/w$ and adding the communication resulting from the left neighbor, we obtain:

$$C(w) = \frac{N(w)}{w} \sum_{i=-l}^r z_i$$

It follows immediately from the above formula that the communication overhead is the same for all partitions of width $w \geq w_{min}$ and therefore to minimize the total computation time we just need to consider the minimum width w_{min} .

The execution time of the single wavefront movement E can now be expressed as (cf. (3))

$$E \leq N(w)c \left(f + \frac{\sum_{i=-l}^r z_i}{w} \right)$$

where $f = e/c$.

In order to minimize the total execution time we need to consider the number of wavefront movements. Let the size of the array being computed be $X \times Y$, then $L = X|\sin \theta| + Y \cos \theta$ is the length the wavefront needs to traverse and the total computation time is:

$$T = \frac{L}{h} E \leq \frac{LcN(w)}{h} \left(\frac{\sum_{i=-l}^r z_i}{w} + f \right) = Lc \left(wf + \sum_{i=-l}^r z_i \right) \quad (6)$$

The first factor, $L = X|\sin \theta| + Y \cos \theta$, represents the distance that the wavefront has to traverse to evaluate rectangle $X \times Y$. Since $\theta \in (-\pi/2, \pi/2]$, therefore for $\theta < 0$ the absolute value of the sine has to be taken.

The length of projection of vector (x_i, y_i) onto the wavefront is $|x_i \cos \theta + y_i \sin \theta|$ and the sum of the projection lengths is $\sum_{i=-l}^r |x_i \cos \theta + y_i \sin \theta|$. Finally, the length of the partition w is equal to the length of the longest projection of the dependence vectors on the wavefront, i.e.

$$w = \left\{ \max_i |x_i \cos \theta + y_i \sin \theta| \right\}$$

The smallest index of the vector that defines w will be denoted by $imax$. Let $\{\gamma_1, \gamma_2, \dots, \gamma_m\}, m \geq 0$ be a sorted list (possibly empty) of those among the following angles that belong to $(\theta_\beta, \theta_\alpha)$:

1. $\arctan(-\frac{x_i}{y_i}), -l \leq i \leq r,$
2. $\pi/2,$

Let $\gamma_0 = \theta_\beta$, and $\gamma_{m+1} = \theta_\alpha$. For each range $(\gamma_j, \gamma_{j+1}), j = 0, 1, \dots, m$, expression (6) can be written as

$$(Z \sin \theta + Y \cos \theta)(a \cos \theta + b \sin \theta) \quad (7)$$

where Z is either X or $-X$ depending on the value of θ , $a = f c_{imax} x_{imax} + \sum_{i=1}^n c_i x_i$, $b = f c_{imax} y_{imax} + \sum_{i=1}^n c_i y_i$, and $c_i \in \{+1, -1\}$. The above formula can be rewritten as

$$\frac{aY - bZ}{2} \cos(2\theta) + \frac{aZ + bY}{2} \sin(2\theta) + \frac{aY + bZ}{2} \quad (8)$$

Let ϕ denote an angle such that

$$\sin \phi = \frac{aY - bZ}{\sqrt{(a^2 + b^2)(Z^2 + Y^2)}} \quad \cos \phi = \frac{aZ + bY}{\sqrt{(a^2 + b^2)(Z^2 + Y^2)}}$$

Formula (8) simplifies to

$$\frac{1}{2} \left(\sqrt{(a^2 + b^2)(Z^2 + Y^2)} \sin(\phi + 2\theta) + aY + bZ \right)$$

and because $\sqrt{(a^2 + b^2)(Z^2 + Y^2)} \geq aY + bZ$ then the minima of the above functions are all negative and hence the minimum of (6) in the range $\gamma_i \leq \theta \leq \gamma_{i+1}$ is always at either or both boundaries of the range. By simply listing all the angles $\gamma_i, i = 0, \dots, m + 1 < (r + l + 1)^2$ and evaluating (6) we can find an optimal angle θ .

Let's consider the example from Figure 1. There are just three angles to consider: $\theta_1 = -\arctan 2, \theta_2 = \pi/2, \theta_3 = \pi/4$. Let's set $X = 100, Y = 10$, then the corresponding total computation time from equation (6) is $T_1/c = 315f + 315, T_2/c = 200f + 400, T_3/c = 630f + 630$, and depending on the value of f , the first or the second angle should be selected (see Figure 3).

In the above analysis we assumed that a processor requests the same array element computed by a neighboring processor every time it needs it. However, if a processor saves the received values when they are referred to more than once, the optimum wavefront can be obtained by determining its angle θ that minimizes

$$Lc \left(fw + \left\{ \max_i \text{proj}_i^+ + \max_i \text{proj}_i^- \right\} \right)$$

where proj_i^+ stands for $\max(x_i \cos \theta + y_i \sin \theta, 0)$ and proj_i^- stands for $\max(-x_i \cos \theta - y_i \sin \theta, 0)$. The algorithm described above can be easily modified to determine the optimum value of θ in this case.

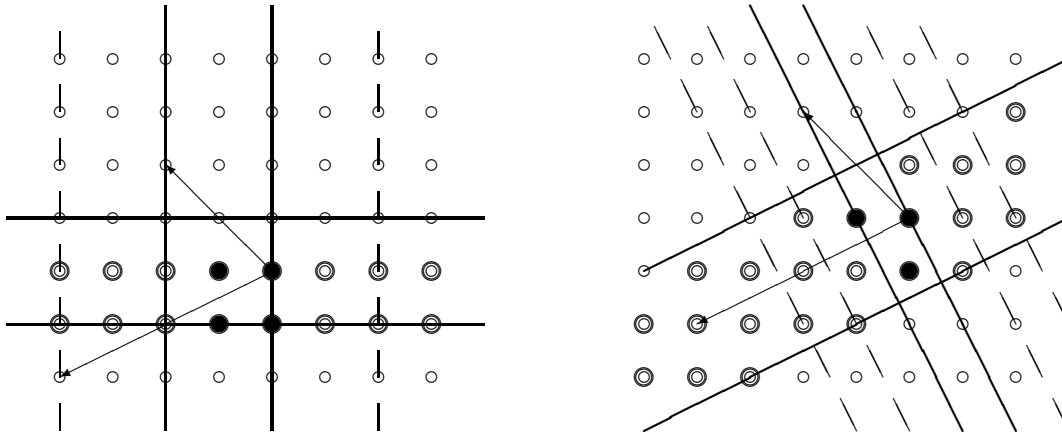


Figure 3: Optimal wavefronts for array E

2.2 Communication Through Packet Passing

Usually, array elements are not passed individually; instead, several of them are grouped together and sent in a single packet. This method is commonly used in the communication model known as *block SIMD*. In this model, off-processor values required to compute a designated block of parallel code are obtained immediately before the beginning of the block, and all off-processor values generated within the block are communicated immediately after the end of the block [10]. Typically, packets of values are formed for communication and transferred between non-neighboring processors by means of hopping. Under these assumptions, the minimum width of the wavefront partition assigned to a processor can be less than $\max(z_{-l}, z_r)$. With too small a width, processors spend less time computing and more time communicating, because less relevant information is available in the local memory. On the other hand, a large width enables processors to spend more time computing between data transfers, resulting in a smaller communication cost. Beyond a certain width, the communication cost does not decrease any further with an increase in the partition width. If the partitions are too large, the available parallelism may not be exploited fully. Selection of the optimal wavefront and the width size of its partitions under the above-mentioned assumptions is discussed below.

In figure 2, $P(i)$ represents the i -th processor that computes a width w of the wavefront. The packets being transferred are rectangular, of size $p_h \times p_w$. The side of the packet parallel to the wavefront is its width, p_w , and the side parallel to the direction of wavefront movement is its height, p_h . At each communication step $\lceil \frac{z-l}{p_w} \rceil$ packets are received by a processor from left. The processor retains some of these packets for use in the next (or a future) computation step and forwards relevant packets to its right (similarly for the packets obtained from the right).

Figure 4 shows a computation with three dependence vectors d_1, d_2 and d_3 . Let z_r be the longest projection on the wavefront of any dependence vector on the right. Since dependence vectors d_2 and d_3 have the same projection, the packets sent from processor $P(n)$ are saved by processor $P(m)$ for use at times t_1 and t_3 (for which the wavefronts are shown). The packet at dependence vector d_1 is saved by processor $P(m)$ and also sent to the processors at left for their future use (at time t_2). This example shows that the transfer of all packets generated to the

right (left) up to length z_r (z_{-l}) after every computation phase is sufficient for the computation to proceed.

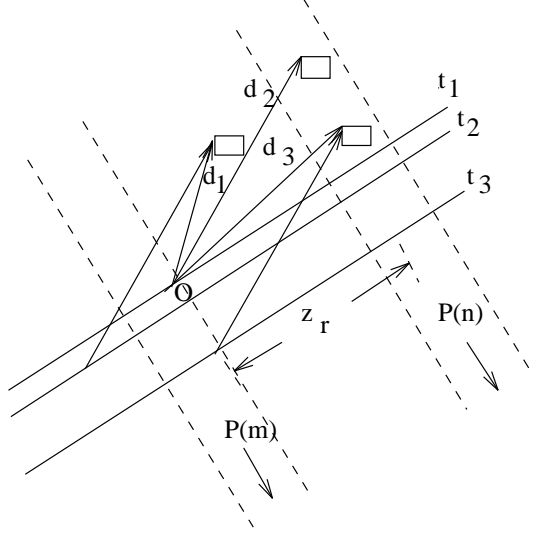


Figure 4: Packet requirements of different processors

The number of packets of size $p = p_h \times p_w$ that pass through $P(i)$ while it computes a strip of length p_h along the direction of computation is

$$C(w) = \left\lfloor \frac{z_{-l}}{w} \right\rfloor \left\lceil \frac{w}{p_w} \right\rceil + \left\lfloor \frac{z_r}{w} \right\rfloor \left\lceil \frac{w}{p_w} \right\rceil + \left\lceil \frac{z_{-l}(\bmod w)}{p_w} \right\rceil + \left\lceil \frac{z_r(\bmod w)}{p_w} \right\rceil \quad (9)$$

where w is the size of the wavefront partitions assigned to the processors. A corollary of the following theorem gives the smallest value of w that minimizes $C(w)$.

Theorem 1 *The minimum value of*

$$C(w) = \left\lfloor \frac{x}{w} \right\rfloor \left\lceil \frac{w}{p} \right\rceil + \left\lceil \frac{x(\bmod w)}{p} \right\rceil$$

is $\left\lceil \frac{x}{p} \right\rceil$. *The smallest value of w for which $C(w)$ reaches the minimum is equal to $\left\lceil \frac{x}{\lceil \frac{x}{p} \rceil} \right\rceil$.*

Proof The proof is performed by case analysis.

Case 1: $0 < w \leq p$

In this case $C(w) = \left\lfloor \frac{x}{w} \right\rfloor \left\lceil \frac{w}{p} \right\rceil \geq \left\lceil \frac{x}{p} \right\rceil$, in particular $C(p) = \left\lceil \frac{x}{p} \right\rceil$.

Case 2: $mp < w \leq (m+1)p$, $m > 0$

$C(w) = (m+1) \left\lfloor \frac{x}{w} \right\rfloor + \left\lceil \frac{x(\bmod w)}{p} \right\rceil$. Let $x = dw + r$ where $0 \leq r < w$, then $C(w) = d(m+1) + \left\lceil \frac{r}{p} \right\rceil$.

If $r = 0$ then $C(w) = d(m + 1)$ and we have $\left\lceil \frac{x}{p} \right\rceil \leq \left\lceil \frac{d(m+1)p}{p} \right\rceil = d(m + 1)$. Hence, $\left\lceil \frac{x}{p} \right\rceil \leq C(w)$.
If $np < r \leq (n + 1)p$ for some n , then we have $C(w) = (m + 1)d + (n + 1)$.

Now, $\left\lceil \frac{x}{p} \right\rceil = \left\lceil \frac{dw+r}{p} \right\rceil \leq \left\lceil \frac{d(m+1)p+(n+1)p}{p} \right\rceil = d(m + 1) + (n + 1) = C(w)$.

Thus, $\left\lceil \frac{x}{p} \right\rceil$ is the minimum of $C(w)$.

We know from case 1 that the smallest w_0 that minimizes $C(w)$ satisfies $w_0 \leq p$, so $C(w_0) = \left\lceil \frac{x}{w_0} \right\rceil$.

By definition, at the minimum, $\frac{x}{w} \leq \left\lceil \frac{x}{p} \right\rceil$, so w_0 is the smallest w for which $w \geq \frac{x}{\left\lceil \frac{x}{p} \right\rceil}$. Hence,

$$w_0 = \left\lceil \frac{x}{\left\lceil \frac{x}{p} \right\rceil} \right\rceil. \quad \square$$

Corollary : *The minimum value of*

$$C(w) = \left\lfloor \frac{z_{-l}}{w} \right\rfloor \left\lceil \frac{w}{p} \right\rceil + \left\lfloor \frac{z_r}{w} \right\rfloor \left\lceil \frac{w}{p} \right\rceil + \left\lfloor \frac{L}{p} \right\rfloor + \left\lfloor \frac{R}{p} \right\rfloor$$

where $L = z_{-l} \pmod{w}$ and $R = z_r \pmod{w}$ is $\left\lceil \frac{z_{-l}}{p} \right\rceil + \left\lceil \frac{z_r}{p} \right\rceil$. The smallest w for which $C(w)$ reaches minimum is $\max \left(\left\lceil \frac{z_{-l}}{\left\lceil \frac{z_{-l}}{p} \right\rceil} \right\rceil, \left\lceil \frac{z_r}{\left\lceil \frac{z_r}{p} \right\rceil} \right\rceil \right)$.

Let's consider now the expression (3). We assume that the packet height is selected equal to the strip height, i.e., $p_h = h$. The above corollary shows the smallest w that minimizes $C(w)$ in (9) to be less than or equal to p_w . Consequently, if w_0 minimizes expression (3) then $w_0 \leq p_w$. But for $w \leq p_w$

$$\frac{z_{-l}}{w} + \frac{z_r}{w} \leq C(w) = \left\lfloor \frac{z_{-l}}{w} \right\rfloor + \left\lfloor \frac{z_r}{w} \right\rfloor \leq \frac{z_{-l}}{w} + \frac{z_r}{w} + 2$$

Hence, approximating $N(w)$ with wh , we obtain

$$\left(\frac{c}{w} \right) \left(\frac{M}{p_h} \right) + we \leq \frac{E}{p_h} \leq \left(\frac{c}{w} \right) \left(\frac{M}{p_h} \right) + we + \frac{2c}{p_h} \quad (10)$$

where $M = z_{-l} + z_r$. E/p_h is the execution cost (computation cost plus communication cost) per unit length of computation along the direction of wavefront movement. Expression (10) suggests that to find the optimum wavefront, $\frac{z_{-l} + z_r}{p_h}$ should be minimized.

$$P(\theta) = \min \left(\frac{z_{-l} + z_r}{p_h} \right) = \min_{\theta \in (\theta_\beta, \theta_\alpha)} \left[\frac{\max_i (proj_i)^+ + \max_i (proj_i)^-}{\min_i (y_i \cos \theta - x_i \sin \theta)} \right]$$

where $proj_i = x_i \cos \theta + y_i \sin \theta$, as previously defined. At any angle θ of the wavefront, at most three dependence vectors are needed to determine P . The angles at which the needed vectors change satisfies at least one of the following equations

$$x_i \cos \theta + y_i \sin \theta = x_j \cos \theta + y_j \sin \theta$$

$$y_i \cos \theta - x_i \sin \theta = y_j \cos \theta - x_j \sin \theta$$

for some $1 \leq i, j \leq n$, $i \neq j$. Let $\{\gamma_1, \dots, \gamma_m\}$ be a sorted list (possibly empty, in which case, similarly as in the previous subsection, we select $(\beta + \alpha + \pi)/2$ as the angle of the optimum

wavefront) of all θ from the interval $(\theta_\beta, \theta_\alpha)$ for which the above conditions are satisfied, i.e., sorted list of $\{\theta_{ij}, \theta'_{ij} | 1 \leq i, j \leq n, i \neq j\}$ where $\theta_{ij} = \arctan\left(\frac{x_j - x_i}{y_i - y_j}\right)$ and $\theta'_{ij} = \arctan\left(\frac{y_j - y_i}{x_j - x_i}\right)$ and $\theta_{ij}, \theta'_{ij} \in [\theta_\beta, \theta_\alpha]$. As done previously, let $\gamma_0 = \theta_\beta$ and $\gamma_{m+1} = \theta_\alpha$. For each range of angles θ in (γ_j, γ_{j+1}) , $j = 0, 1, \dots, m$ we need to find the n dependence vectors ($n \leq 3$) that define P . In each of the ranges $\gamma_j \leq \theta \leq \gamma_{j+1}$, P can be written as

$$P = \frac{a \cos \theta + b \sin \theta}{c \cos \theta - d \sin \theta} \quad c \cos \theta \neq d \sin \theta$$

for some constants a, b, c and d . Now,

$$\frac{\partial P}{\partial \theta} = \frac{ad + bc}{(c \cos \theta - d \sin \theta)^2} \quad c \cos \theta \neq d \sin \theta$$

At the extrema of P , $ad + bc = 0$ and

$$P = \frac{a \cos \theta + b \sin \theta}{c \cos \theta - d \sin \theta} = \frac{a}{c}$$

Thus if $ad + bc = 0$, P is a constant function. Otherwise P does not have any extremum point within the range and so the minimum is at either of the boundaries, i.e., either at γ_j or at γ_{j+1} .

The angle θ , that minimizes P can be obtained by selecting γ_k such that

$$P(\gamma_k) = \min P(\theta), \quad \theta \in \{\gamma_1, \dots, \gamma_m\}$$

Since m is $O(n^2)$, the optimum direction of wavefront can be obtained in $O(n^2)$ time. Once the optimal angle θ is known, all parameters other than w in expression (3) can be calculated.

We have seen that the optimum w is less than equal to p_w . From (3) we have

$$\frac{E}{p_h} = \left(\frac{c}{p_h}\right) \left(\left\lceil \frac{z_{-l}}{w} \right\rceil + \left\lceil \frac{z_r}{w} \right\rceil\right) + we = \frac{c(z_{-l} + z_r)}{p_h w} + we \quad (11)$$

From equation (11), the minimum value of $\frac{E}{p_h}$ is obtained at

$$w_0 = \sqrt{\frac{c(z_{-l} + z_r)}{p_h e}}$$

Thus the suitable width w of an individual partition is $\sqrt{\frac{c(z_{-l} + z_r)}{p_h e}}$ when $w_0 \leq p_w$ and p_w otherwise.

2.3 Index Point Enumeration

In this section we present an efficient method for enumerating points in a partition assigned to the given processor. As previously, let θ denote the angle that the optimal wavefront makes with the x -axis, h be the height of its strip and w be the length of its partition. Let's assume also that the index domain contains all integer points $\{(x, y) | 0 \leq x \leq X, 0 \leq y \leq Y\}$. The enumeration is very simple when $\theta = 0$ or $\theta = \pi/2$, i.e., when the optimum wavefront is parallel to one of the axes of the coordinate system; hence we can exclude such cases from further analysis. As

demonstrated in the preceding sections, the optimal wavefront and its direction have a rational slope towards the coordinate axes; consequently there are two relatively prime non-zero integers, q_y, q_x such that:

$$\tan(\theta + \frac{\pi}{2}) = \frac{q_y}{q_x}$$

Hence, all index points lie on two families of parallel lines given by:

$$y_i = \frac{q_y}{q_x}x + \frac{i}{q_x} \text{ and } z_i = -\frac{q_x}{q_y}x + \frac{i}{q_y}$$

The boundaries between partitions are cut by some of these lines because the partition thickness is defined by the end-points of the dependence vectors, so $w_i = wq_x$ is an integer. Similarly, $h_i = hq_y$ is also an integer.

Before starting the main computation, each processor P_j finds its initial points and w_i lines on which these points lie. The lines are given by:

$$y_{jk} = \frac{q_y}{q_x}x + \frac{jw_i + k}{q_x} \text{ where } k = 0, 1, \dots, w_i - 1$$

During the t -th time step, processors evaluate strip contained between the lines

$$z_t = -\frac{q_x}{q_y}x + \frac{th_i}{q_y} \text{ and } z_{t+1} = z_t + \frac{h_i}{q_y}$$

For each processor P_j and its line y_{jk} a single initial point $(x_{jk}^{(0)}, y_{jk}^{(0)})$ is found from the following equations:

$$x_{jk}^{(0)} = \begin{cases} -(jw_i + k)q_y^{-1}i \bmod q_x & \text{if } j \geq 0 \\ \left\lfloor \frac{-jw_i - k}{q_y} \right\rfloor + \left(-\left\lfloor \frac{-jw_i - k}{q_y} \right\rfloor - (jw_i + k)q_y^{-1} \right) \bmod q_x & \text{if } j < 0 \end{cases}$$

$$y_{jk}^{(0)} = \frac{q_y x_{jk}^{(0)} + jw_i + k}{q_x}$$

where q_y^{-1} is an arithmetic reciprocal of q_y modulo q_x , i.e. $q_y^{-1}q_y \equiv 1 \pmod{q_x}$ (the well known Euclidian algorithm can be used to find q_y^{-1} efficiently). Additionally, the position of the initial point versus the first wavefront line z_0 is set using an equation

$$d_{jkt} = q_x^2 q_y (y_{jk}(x_{jk}) - z_t(x_{jk})) = (q_x^2 + q_y^2) q_x x_{jk} + (jw_i + k) q_x^2 - th_i q_x q_y$$

To execute the main computation, the processor P_j runs the code in Figure 5. There are only a few integer arithmetic operations required for each evaluated index point.

In the example of figure 6, there are three dependence vectors, $\vec{OP}(-2, -1)$, $\vec{OQ}(-1, -2)$ and $\vec{OR}(-1, -3)$. Using the results of the previous sections we need to consider only four possible wavefronts: hyperplane parallel to the y -axis and hyperplanes that are perpendicular to the vectors \vec{OP} , \vec{OQ} and \vec{OR} (shown by dashed, solid and dotted lines, respectively, passing through O). The sum of the projections of the vectors on a wavefront is minimum for the wavefront perpendicular to \vec{OQ} and the sum is $4/\sqrt{5}$. If there are enough processors available then each

```

/* initialization */
for(k=0;k++;k < wi) {
    x[k]=xjk(0);
    y[k]=yjk(0);
    d[k]=(qx2 + qy2)qxx[k]+(jwi+k)qx2;
}
/* evaluation */
for(t=0;t < tmax;t++)
    for(k=0;k < wi;k++) {
        while(d[k] < 0) {
            evaluate(x[k],y[k]);
            x[k]+=qx;
            y[k]+=qy;
            if (x[k]>X | y[k]>Y) d[k]=(qx2 + qy2)qx2tmax;
            else d[k]+=(qx2 + qy2)qx2;
        }
        d[k]-=hiqxqy;
    }
}

```

Figure 5: Index Point Enumeration Code

one can evaluate a partition of width $4/\sqrt{5}$ resulting in maximum parallelism possible with this wavefront. For example, all index points within the dashed lines PP' and RR' can be allocated to a single processor. All points in the rectangle $MNLP$ (which constitutes a partition) are then computed by the assigned processor in one computational step.

All points within the rectangle $MNPL$ lie on two families of parallel lines (each consisting of four lines and the lines of one family are perpendicular to the lines of the other). For this example $q_x = 1$, $q_y = 1$, $h_i = w_i = 4$. Each line contributes at most one point to each partition. Assuming that the point origin of the coordinates (i.e. the point $(0, 0)$) coincides with the point P, the coordinates of the initial points and their lines for the marked partition are:

$$(0, 0), y_0 = 2x; (1, 1), y_{-1} = 2x - 1; (1, 0), y_{-2} = 2x - 2; (2, 1), y_{-3} = 2x - 3;$$

3 Higher Dimension

The problem of finding the best wavefront for distributed memory machines becomes much more complicated for higher dimensions. Suppose we are given a set of mutually recursive equations defining a set of three dimensional arrays. In this case the wavefront is a plane which passes through the three dimensional arrays. For one-dimensional distributed memory machines the compiler can easily determine which of the packets should be retained by a particular processor.

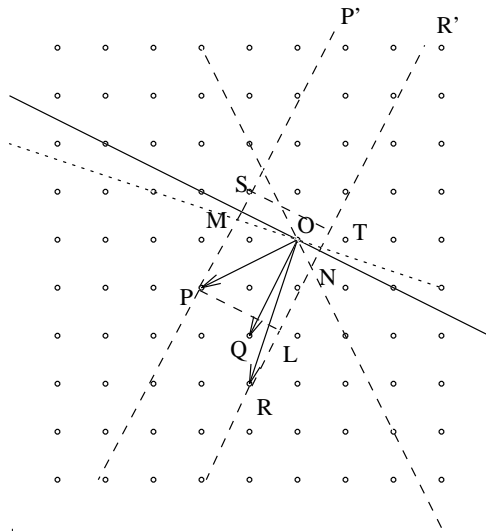


Figure 6: Determination of the wavefront

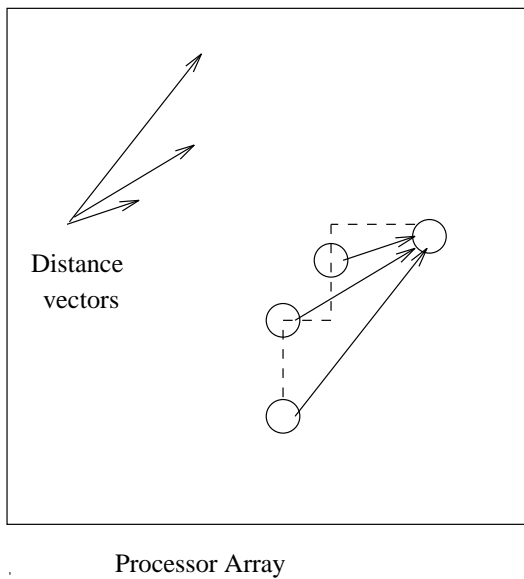


Figure 7: Routing of packets in a two dimensional processor array

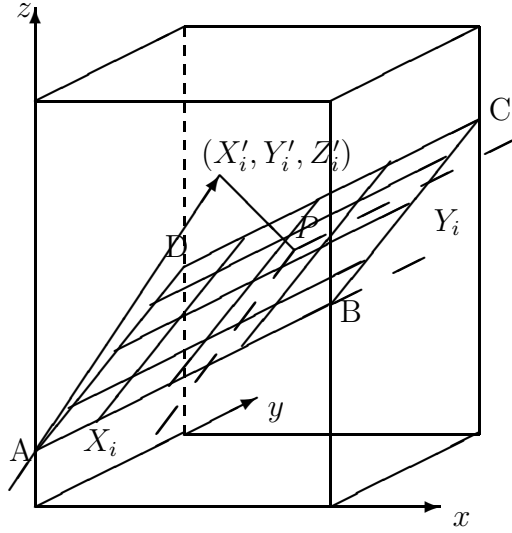


Figure 8: A two dimensional wavefront ABCD, sweeping through a three dimensional array

However, with two-dimensional distributed memory machines, this determination is very difficult. Figure 7 shows an example in which the compiler can determine the specific routing of the packets that will minimize the number of messages (circles denote processors; routing is shown by the dotted line which passes through all the processors that need a particular packet at some point of the execution). If all *inverse dependence vectors* can be arranged in such a way that elements in each dimension of the dependence vectors form a sorted list, then such a routing of the packets is easy to arrange. However, the authors are not aware of any general method of arranging the inverse dependence vectors in such a manner.

A two dimensional wavefront ABCD is shown in figure 8 sweeping through a three dimensional array. The wavefront is tessellated with rectangles. All the index points in each of the rectangles are processed by a distinct processor. As the wavefront sweeps through the array, each of the rectangles sweeps through a parallelepiped. Index points in each of the parallelepiped are computed by a distinct processor.

Let the direction cosines² of a perpendicular to this wavefront be $(\gamma_1, \gamma_2, \gamma_3)$. Let (X'_i, Y'_i, Z'_i) be a dependence vector and its projection on the wavefront is at point N. The processor computing index point A need to communicate with the processor that computes index point at (X'_i, Y'_i, Z'_i) . As shown in figure 8, the relative position of the later processor with respect to the former is (X_i, Y_i) . The projection of $(\gamma_1, \gamma_2, \gamma_3)$ on the xz -plane is $(\gamma_1, 0, \gamma_3)$. Since \vec{AB} is perpendicular to $(\gamma_1, 0, \gamma_3)$, its direction cosines are

$$\left(\frac{\gamma_3}{\sqrt{\gamma_1^2 + \gamma_3^2}}, 0, -\frac{\gamma_1}{\sqrt{\gamma_1^2 + \gamma_3^2}} \right)$$

²*Direction cosines* of a vector are the cosines of the angles that the vector makes with the positive x -axis, positive y -axis and positive z -axis.

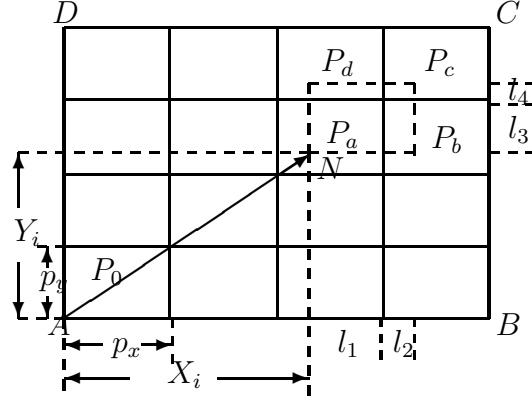


Figure 9: Necessary data transfers: Processor P_0 needs to communicate with processors P_a , P_b , P_c and P_d to compute the rectangle $w_x \times w_y$ assigned to it

Similarly the direction cosines of \vec{AD} are

$$\left(0, \frac{\gamma_3}{\sqrt{\gamma_2^2 + \gamma_3^2}}, -\frac{\gamma_2}{\sqrt{\gamma_2^2 + \gamma_3^2}}\right)$$

N is the projected point of the position vector (X'_i, Y'_i, Z'_i) , so

$$\vec{AN} = (X'_i, Y'_i, Z'_i) - (X'_i, Y'_i, Z'_i) \cdot (\gamma_1, \gamma_2, \gamma_3)$$

Thus

$$\begin{aligned} X_i &= \vec{AN} \cdot \vec{AB} = \left| (X'_i - X'_i \cdot \gamma_1) \cdot \Gamma_{13} - Z'' \cdot \Gamma_{31} \right| \\ Y_i &= \vec{AN} \cdot \vec{AD} = \left| (Y'_i - Y'_i \cdot \gamma_2) \cdot \Gamma_{32} - Z'' \cdot \Gamma_{23} \right| \end{aligned}$$

where $\Gamma_{ij} = \frac{\gamma_i}{\sqrt{\gamma_i^2 + \gamma_j^2}}$, and $Z'' = Z'_i - Z'_i \cdot \gamma_3$. The communication requirement for the processor P_0 computing the rectangle of index points at A is shown in figure 9. Rectangle of size $(l_1 \times l_3)$ needs to be communicated from processor P_a to processor P_0 . The number of packets of size $p_x \times p_y$ in this rectangle located at N is

$$\left\lceil \frac{w_x - X_i \pmod{w_x}}{p_x} \right\rceil \cdot \left\lceil \frac{w_y - Y_i \pmod{w_y}}{p_y} \right\rceil.$$

The relative position of processor P_a with respect to processor P_0 is

$$\left(\left\lfloor \frac{X_i}{w_x} \right\rfloor, \left\lfloor \frac{Y_i}{w_y} \right\rfloor \right)$$

So if the packets are not retained by a processor for future use, the total number of packet transferred between neighboring processors P_a and P_0 is

$$W_X^i \cdot W_Y^i \cdot \left(\left\lfloor \frac{X_i}{w_x} \right\rfloor + \left\lfloor \frac{Y_i}{w_y} \right\rfloor \right)$$

where $W_X^i = \left\lceil \frac{w_x - X_i \pmod{w_x}}{p_x} \right\rceil$, and $W_Y^i = \left\lceil \frac{w_y - Y_i \pmod{w_y}}{p_y} \right\rceil$.

Adding all such packet transfers from processors P_a, P_b, P_c and P_d to processor P_0 we get the total communication cost as

$$\begin{aligned}
C(w) &= \sum_{i=1}^n W_x^i \cdot W_y^i \cdot \left(\left\lceil \frac{X_i}{w_x} \right\rceil + \left\lceil \frac{Y_i}{w_y} \right\rceil \right) + \sum_{i=1}^n W_X^i \cdot W_Y^i \cdot \left(\left\lfloor \frac{X_i}{w_x} \right\rfloor + \left\lfloor \frac{Y_i}{w_y} \right\rfloor \right) \\
&+ \sum_{i=1}^n W_x^i \cdot W_Y^i \cdot \left(\left\lceil \frac{X_i}{w_x} \right\rceil + \left\lfloor \frac{Y_i}{w_y} \right\rfloor \right) + \sum_{i=1}^n W_X^i \cdot W_y^i \cdot \left(\left\lfloor \frac{X_i}{w_x} \right\rfloor + \left\lceil \frac{Y_i}{w_y} \right\rceil \right) \quad (12)
\end{aligned}$$

where p_x and p_y are the dimensions of the packets, w_x and w_y the widths of the array that a processor is assigned to compute, $[X_i, Y_i]$ are the co-ordinates of the projections of the i -th dependence vector on the wavefront. W_x^i and W_y^i stands for $\left\lceil \frac{X_i \pmod{w_x}}{p_x} \right\rceil$ and $\left\lceil \frac{Y_i \pmod{w_y}}{p_y} \right\rceil$, respectively.

If we ignore the ceiling and floor functions (i.e., ceiling and floor functions are replaced by the identity function), expression (12) reduces to

$$C(w) = \sum_{i=1}^n \frac{w_x w_y}{p_x p_y} \left(\frac{X_i}{w_x} + \frac{Y_i}{w_y} \right)$$

which suggests that to find a good wavefront attempt should be made to minimize $\sum_{i=1}^n |X_i| + |Y_i|$. Let $[x_i, y_i, z_i]$ $1 \leq i \leq n$ be the dependence vectors and γ_1, γ_2 and γ_3 denote the direction cosines of the perpendicular to the wavefront, then we would like to find values of γ_1, γ_2 and γ_3 that minimize

$$|(x_i - x_i \cdot \gamma_1) \cdot \Gamma_{31} - (z_i - z_i \cdot \gamma_3) \cdot \Gamma_{13}| + |(y_i - y_i \cdot \gamma_2) \cdot \Gamma_{32} - (z_i - z_i \cdot \gamma_3) \cdot \Gamma_{23}|$$

(Γ s are as defined before) under the constraint that

$$\gamma_1^2 + \gamma_2^2 + \gamma_3^2 = 1 \quad 0 \leq \gamma_1, \gamma_2, \gamma_3 \leq 1$$

Further studies are necessary to design an efficient algorithm for determining γ_1 and γ_2 that satisfy all these conditions.

4 Final Remarks

The problem of finding the optimum wavefront that minimizes total computation time is discussed in this paper for iterative computations over two-dimensional arrays executed on linear or two-dimensional processor arrays. Assuming a continuum of data elements in two-dimensional arrays computed on a linear array of processors, efficient algorithms have been presented to determine the optimum wavefront of computation and the optimum partitioning of the wavefront into sections assigned to individual processors. An $O(n^2)$ time algorithm (where n is the number of data dependence vectors) for finding the optimum wavefront for one-dimensional processor arrays has been described. In addition, a method for finding a good wavefront for two-dimensional SIMD machines has been presented.

In our analysis, we have assumed a continuum of data elements in an array. In reality, arrays are discrete, so the analysis is approximate. For example in mapping a computation onto a linear

array of processors, the algorithm provides very good wavefront when z_{-l} and z_r , the longest projections (on each sides) of the data dependence vectors on the selected wavefront, are much larger than p_w (the length of the packets along the wavefront).

The methods described here can be applied to any set of uncoupled recurrence equations. To decrease the communication cost a good alignment of all arrays in the program should be determined first (see [12, 5]). Many methods described in the literature [11, 3, 4, 7] determine the actual mapping of the computation onto the processors, once the wavefront is determined by solving an integer programming optimization problem (see section 1). These algorithms can be used for the wavefronts obtained by our method.

When communication is accomplished by means of packet switching, there is evident difficulty in handling packets that are not cut out in parallel to the natural sides of the arrays. How to generate packets of convenient sizes and shapes that can be handled efficiently, once their size and orientation are known, is an interesting topic for further research.

There are many open problems in this area. One major issue concerns finding an efficient algorithm to determine a good wavefront when a set of recurrence equations involving m -dimensional arrays are to be computed on an n -dimensional array of processors ($m \geq n$).

References

- [1] M. R. Garey and D. S. Johnson, *Computers and Intractability A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, New York, 1979.
- [2] L. Lamport. "The parallel execution of do loops", *CACM*, **17**, 1974.
- [3] P.-Z. Lee and Z. M. Kedem, "Synthesizing Linear Array Algorithms from Nested For Loop Algorithms", *IEEE trans. on Computers*, Vol. 37, No. 12, December 1988.
- [4] P.-Z. Lee and Z. M. Kedem, "Mapping Nested Loop Algorithms into Multidimensional Systolic Arrays," in *IEEE Transactions on Parallel and Distributed Processing*, vol 1, no 1, January 1990.
- [5] J. Li and M. Chen, *Index Domain Alignment: Minimizing Cost of Cross-Referencing Between Distributed Arrays*, Tech. Rep., Dept. of Comp. Sc., Yale, Univ., YALEU/DCS/TR-725, November 1989.
- [6] W. L. Miranker and A. Winkler, "Spacetime Representations of Computational Structures," in *Computing* Vol 32, No 2, pp. 93-114, 1984.
- [7] D. I. Moldovan, "Partitioning and Mapping Algorithms into Fixed Size Systolic Arrays," in *IEEE Transactions on Computers*, Vol C-35, No 1, pp. 1-12, January 1986.
- [8] C. D. Polychronopoulos, D. J. Kuck and D. A. Padua, "Utilizing Multidimensional loop Parallelism on Large-Scale Parallel Processor Systems," in *IEEE Transactions on Computers*, Vol 38, No 9, pp. 1285-1296, September 1989.
- [9] S. K. Rao, *Regular Iterative Algorithms and their Implementations on a Processor Arrays*, Ph. D. Thesis, Dept. of Electrical Engineering, Stanford University, California, 1985.

- [10] M. Rosing, R. B. Schnabel and R. P. Weaver, Scientific Programming Languages for Distributed Memory Multiprocessors: Paradigms and Research Issues, in *Languages, Compilers and Run-Time Environments for Distributed Memory Machines* by J. Saltz and P. Mehrotra (eds), Elsevier Science Publishers, 1992.
- [11] J.-P. Sheu and T.-H. Tai, "Partitioning and Mapping nested Loops on Multiprocessor Systems," in *IEEE Trans. on Parallel and Distributed Systems*, vol. 2, No. 4, pp. 430-439, October 1991.
- [12] B. Sinharoy and B. K. Szymanski, "Complexity Issues of the Alignment Problem and the Closest Vectors in a Lattice", Technical Report 91-10, Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180, May 1991. Submitted to *IEEE Transactions on Parallel and Distributed Systems*.
- [13] Szymanski, B.K. (edt): *Parallel Functional Languages and Environments* ACM Press: New York, NY, 1991
- [14] C.-M. Wang and S.D. Wang, "Efficient Processor Assignment Algorithms and Loop Transformations for Executing Nested Parallel Loops on Multiprocessors," in *IEEE Transactions on Parallel and Distributed Systems*, Vol 3, No 1, pp. 71-82, January 1992.