NASA-CR-178,102

NASA Contractor Report 178102

ICASE REPORT NO. 86-24

ICASE

STENCILS AND PROBLEM PARTITIONINGS:
THEIR INFLUENCE ON THE PERFORMANCE OF MULTIPLE PROCESSOR SYSTEMS

Daniel A. Reed
Loyce M. Adams
Merrell L. Patrick

Contract Nos. NAS1-17070, NAS1-18107
May 1986

INSTITUTE FOR COMPUTER APPLICATIONS IN SCIENCE AND ENGINEERING
NASA Langley Research Center, Hampton, Virginia 23665

Operated by the Universities Space Research Association

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

Stencils and Problem Partitionings: Their Influence on the Performance of Multiple Processor Systems

Daniel A. Reed[†]
Department of Computer Science
University of Illinois
Urbana, Illinois 61801

Loyce M. Adams[†]
Department of Applied Mathematics
University of Washington
Seattle, Washington 98195

Merrell L. Patrick[†]
Department of Computer Science
Duke University
Durham, North Carolina 27706

ABSTRACT

Given a discretization stencil, partitioning the problem domain is an important first step for the efficient solution of partial differential equations on multiple processor systems. We derive partitions that minimize interprocessor communication when the number of processors is known *a priori* and each domain partition is assigned to a different processor. Our partitioning technique uses the stencil structure to select appropriate partition shapes. For square problem domains, we show that non-standard partitions (e.g., hexagons) are frequently preferable to the standard square partitions for a variety of commonly used stencils. We conclude with a formalization of the relationship between partition shape, stencil structure, and architecture, allowing selection of optimal partitions for a variety of parallel systems.

[†] Research supported by the National Aeronautics and Space Administration under NASA Contract Numbers NAS1-17070 and NAS1-18107 and by the U.S. Air Force Office of Scientific Research under contract No. AFOSR-76-2881 while the authors were in residence at ICASE, NASA Langley Research Center, Hampton, VA 23665

Merrell L. Patrick was also supported in part by NASA Grant Number NAG-1-466.

Daniel A. Reed was also supported in part by NSF Grant Number DCR 84-17948 and NASA Contract Number NAG-1-613.

N86-25995#

1. Introduction

Problem transformation has long been among the most successful solution paradigms. As an example, consider the solution of elliptic partial differential equations [Orte85]. Given some planar region R, the classical central difference technique covers the region R with a rectangular grid and replaces the derivatives at each grid point with central differences. The resulting system of linear equations is then amenable to solution via a variety of efficient algorithms. This transformation, from partial differential equation to linear system, makes the solution both feasible and attractive. Within this framework there remain several alternatives, both in the choice of discretization stencil (e.g., 5–point or 9–point) and the linear system solver (e.g., direct or iterative), and the most appropriate choices depend on the problem.

When one considers *parallel* solution of partial differential equations, an additional paradigm, problem domain decomposition [Voig85], arises. If multiple processors are to cooperate, each solving the linear equations on a portion of the grid, the selection of grid partitions and their assignment to processors are crucial to good performance.

In this paper, we consider the parallel solution of elliptic partial differential equations over a planar region, using both shared memory and message passing architectures. Historically, only rectangular partitions of the discretization grid have been assigned to processors, primarily because the resulting data structures are regular. However, triangles, squares (a special case of rectangles), and hexagons also tessellate the plane. The effects of these partitions on inter–processor communication and their relation to the discretization stencil are investigated. Because partitions like hexagons have a higher area to perimeter ratio than rectangles and potentially less interpartition communication, there is incentive to investigate their attributes.

Our results show that the efficiency of the parallel solution depends on the partitioning of the discretization grid, its associated stencil, *and* the underlying architecture. Observing that the amounts of required computation and communication are functions of a partition's area and perimeter, respectively, we compare the performance of a variety of associated stencil/partition pairs on both message passing and shared memory architectures. However, we begin with a survey of related work and a formal specification of the problem.

1.1. Related Work

In a study of hypercube performance, Fox and Otto [Fox84] recently noted that the efficiency of a parallel algorithm is not determined by the amount of communication but the ratio of communication to calculation. In their study, they considered the solution of Laplace's equation over a square region using a 5–point discretization stencil. Their partitioning placed squares of grid points on each node of the hypercube, using only nearest neighbor communication. This choice of partitioning has a lower communication to computation ratio than the natural alternative, partitioning the grid into an equal number of rectangular strips.

Vrsalovic, *et al.* [Vrsa85] have also considered the solution of Poisson's equation over a square region using a 5–point discretization stencil. Unlike Fox and Otto, they tested triangular, square, and hexagonal partitions. Their study used the ratio of processing time to data access time as one performance metric when comparing the speedup of different partitions on a general class of multiprocessor systems. Their hypothetical multiprocessor systems were assumed to have both local memory attached to each processor and global memories accessible via an interconnection network. Of the three partitions, hexagonal decomposition produced the largest speedup.

In an experimental study, Saltz, *et al.* [Salt86] considered solution of the heat equation using successive over–relaxation (SOR) on an Intel iPSC [Ratt85]. Rectangular strips and squares were used as grid partitions. They observed that the Intel iPSC's high startup costs for message transmission often favored decreasing the number of messages sent, even if that meant sending more bytes of data. Hence, partitions of rectangular strips were often more efficient that square partitions.

Superficially, these results by Fox and Otto, Vrsalovic, *et al.*, and Saltz *et al.* seem mutually contradictory – each favoring different partition shapes. However, these studies considered only a small portion of the possible parameter space of stencils, partitionings, and architectures. Moreover, their underlying assumptions differ. This paper presents a formal method for analyzing stencil/partition/architecture triplets and applies this method to a variety of these triplets. Section 2 begins by computing the total number of points in a partition versus the number of points that must be communicated for several common stencils using each of the rectangular, square, triangular, and hexagonal partitions. In section 3, these results are used to determine those stencil/partition pairs that maximize the ratio of computation to communication. Finally, section 4 compares the performance of an algorithm for solving Laplace's equation over a square region using different stencil/partition pairs on both shared memory and message passing architectures.

2. Communication Costs for Selected Stencil/Grid Partition Pairs

Elliptic partial differential equations, particularly the Laplace and Poisson equations, have long been used as test vehicles for new solution algorithms and parallel architectures. Consequently, our study is based on the following problem formulation.

The Problem: Consider an elliptic partial differential equation with Dirichlet boundary conditions on some square region R. If R is discretized to contain $N = n^2$ points, we wish to solve the resulting linear system using a point Jacobi iterative solver on a parallel processor containing p processors (PEs), where $p \leq N$.

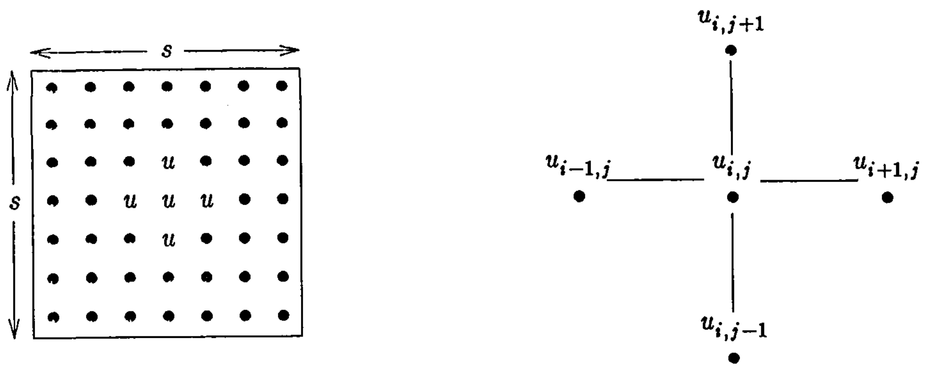
One interesting question immediately arises. Suppose the grid were divided with each partition placed in a different PE and that each PE used the point Jacobi iterative solution technique.[1] In this scenario, each PE repeatedly updates its partition of grid points and sends values associated with its partition boundary to logically adjacent partitions. What partition structure would maximize the ratio of computation to communication? One immediately observes that

- computation is a function of a partition's area,
- communication is a function of a partition's perimeter, and
- the partition's perimeter that must be sent to other partitions is a function of the stencil.

As an example, Figure 2.1 illustrates square partitions with a 5–point stencil. Each partition communicates with four neighboring partitions, and the amount of data transferred is directly proportional to the perimeter of the partition. Although convergence checking for an iterative scheme also involves communication, the amount and cost of this communication is independent of stencil type and partition shape and will not be considered. (It is interesting to note that the communication required for the inner products of the conjugate gradient method is also independent of stencil type and partition shape.)

In the remainder of this section, we analyze the expected amount of data that must be transferred between partitions, given possible stencil/partition pairs. In a later section, we will consider the influence of parallel architecture on the choice of a stencil/partition pair.

[1]The iterates generated by our parallel Jacobi method are the same as those generated by the sequential Jacobi method. We also emphasize that our analysis techniques can be applied to other point iterative solvers (e.g., multicolor SOR and conjugate gradient) as well.



computation $\approx s^2$
communication $\approx 4s$

$$u_{i,j} = \frac{1}{4}\left[u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1}\right]$$

(a) (b)

Figure 2.1 Square partitions with 5–point stencil

2.1. Five Point Stencil

Figure 2.1b shows the 5–point stencil and the equations for the unknowns in Laplace's equation that arise from the standard centered difference approximation to the partial derivatives. With an iterative solution of these equations (e.g., point Jacobi), the new value computed at each grid point depends on the previous values from its north, south, east, and west grid point neighbors.

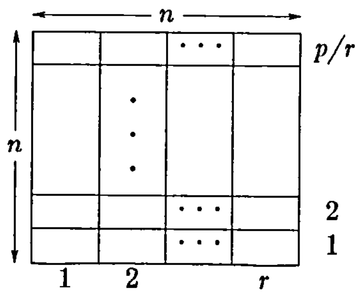
Using this stencil, we now consider the influence of partition shape on inter–PE communication. To ease comparison, we assume each partition contains n^2/p grid points (i.e., each PE's computation is proportional to n^2/p).

2.1.1. Rectangular Partitions

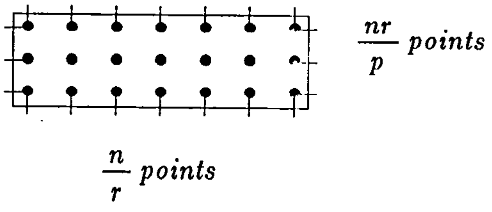
Suppose the grid of n^2 points were partitioned into $\frac{p}{r}$ horizontal strips, and each strip were again partitioned into r rectangles; see Figure 2.2a. Assuming all rectangles are of equal size, each contains $\frac{n^2}{p}$ grid points with sides $\frac{n}{r}$ and $\frac{nr}{p}$. As illustrated in Figure 2.2b, the perimeter contains $2\left[\frac{n}{r}+\frac{nr}{p}\right]-4$ grid points and all are involved in data transfer. However, the four corner points in each rectangle involve two (2) data transfers. Therefore, the data transferred from each interior rectangle is $2\left[\frac{n}{r}+\frac{nr}{p}\right]$.

To find an optimal value for r, the number of horizontal rectangles, we need only maximize the ratio of computation to communication

$$F(r) = \max_{\substack{r>1 \\ r\le p}} \frac{\frac{n^2}{p}}{2\left[\frac{n}{r}+\frac{nr}{p}\right]} = \max_{\substack{r>1 \\ r\le p}} \frac{nr}{2(p+r^2)}$$



(a)



(b)

Figure 2.2 Rectangular partitions with 5–point stencil

in a single PE. Differentiating and setting the derivative equal to zero, we obtain $p = r^2$ or $r = \sqrt{p}$ as the optimal value of r. *Therefore, squares are the optimal rectangular partitioning for the 5-point stencil with a communicating perimeter of* $\frac{4n}{\sqrt{p}}$. With the 5-point stencil, this result has a simple geometric interpretation: of all rectangular partitions, the square maximizes the area/perimeter ratio.

Finally, as an interesting special case, note that if $r = 1$, the grid of n^2 points is partitioned into p strips each containing $\frac{n^2}{p}$ points. In this case, there is no communication to the east or west and $2n - 4$ values ($n - 2$ north and $n - 2$ south) are communicated from each partition.[2]

2.1.2. Triangular Partitions

To partition an $n \times n$ grid into p triangles we assume $n = 2\sqrt{p}\,l$ and divide the grid into $\frac{p}{2}$ squares with sides $s = 2\sqrt{2}l$. Each of these $\frac{p}{2}$ squares will contain $8l^2$ grid points. Each of the squares is then divided into the two "approximate" triangles shown in Figure 2.3a. Each of the p triangles contains $4l^2$ grid points and has height s and base $s-1$.

Now consider the communicating perimeter of the upper triangle in Figure 2.3a, assuming a 5-point stencil. By observation, s values are sent north, $s-1$ values east, s values south, and 1 value to the west, for a total of $3s$. Note that $s - 2$ of the values transmitted south are used twice by the receiving triangle. The other triangles are reflections of this case. Because $n = 2\sqrt{p}\,l$ and $s = 2\sqrt{2}l$, the total number of values sent from each triangle is $\frac{3\sqrt{2}n}{\sqrt{p}}$.

[2]The four corner points of the partition are fixed boundary values that need not be transmitted.

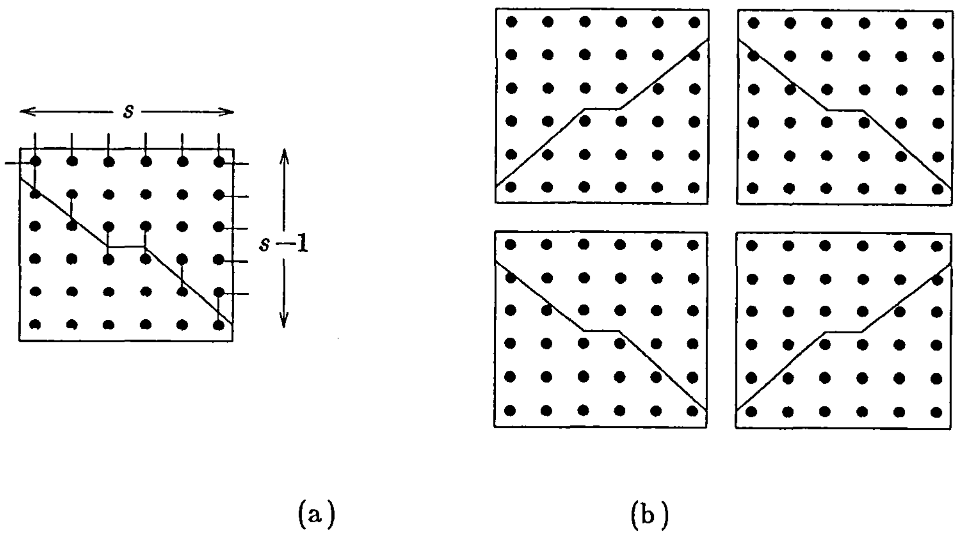


Figure 2.3 Triangular partitions with 5–point stencil

2.1.3. Hexagonal Partitions

Now consider dividing the $n \times n$ grid into p hexagonal partitions. We again assume that $n = 2\sqrt{p}\,l$ implying each partition has $\frac{n^2}{p} = 4l^2$ grid points. Figure 2.4 shows how this partitioning can be accomplished. Each hexagon has $l + 1$ grid points at the north and south edges and l grid points on each of the four remaining sides. The number of grid points in the upper or lower half of each hexagon is

$$\sum_{i=1}^{l}\left[(l+1)+2(i-1)\right] = 2l^2,$$

for a total of $4l^2$ in each hexagon.

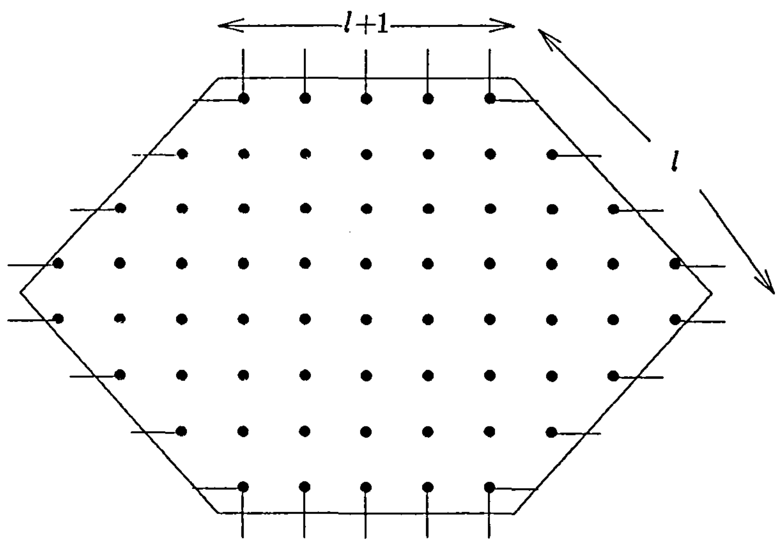
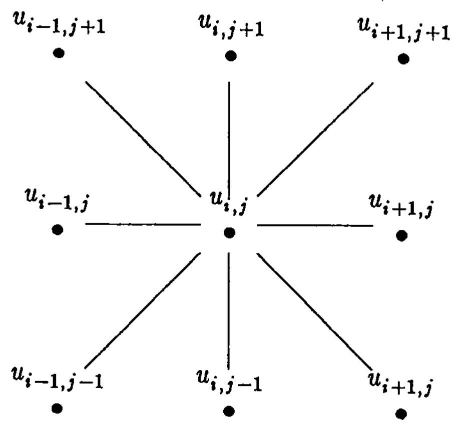


Figure 2.4 Hexagonal partitions with 5–point stencil

As Figure 2.4 shows, $l+1$ values must be sent north, $l+1$ values south, l northeast, l southeast, l southwest, and l northwest, a total of $6l+2$. Because $l = \frac{n}{2\sqrt{p}}$, each hexagon must communicate $\frac{3n}{\sqrt{p}}$ +2 values.

2.2. Nine Point Stencil

The 9–point stencil, shown in Figure 2.5, is a higher order finite difference approximation to the partial derivatives than the 5–point stencil discussed earlier. When using this stencil, the iteration value computed at each grid point is a function of its north, northeast, east, southeast, south, southwest, west, and northwest grid point neighbor values. In this section we examine the amount of inter–PE communication for the same partitions discussed earlier and observe the change in a partition's communicating perimeter as the stencil changes.



$$u_{i,j} = \frac{1}{5}\left[u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1}\right] + \frac{1}{20}\left[u_{i-1,j+1} + u_{i+1,j+1} + u_{i-1,j-1} + u_{i+1,j-1}\right]$$

Figure 2.5 9–point star stencil

2.2.1. Rectangular Partitions

Figures 2.2 and 2.5 show that the communicating perimeter of rectangular partitions for the 9–point stencil is nearly the same as the communicating perimeter for the 5–point stencil. Only the four corner points of a partition are each involved in an additional communication. As before, squares are the optimal rectangular partitioning with a communicating perimeter of $\frac{4n}{\sqrt{p}} + 4$. Because there is no communication to the left or right, rectangular strips ($r = 1$) have the same communicating perimeter for both the 5 and 9–point stencils.

2.2.2. Triangular Partitions

The dashed lines between grid points in Figure 2.6 highlight the additional communications required for triangular partitions when using the 9-point stencil rather than the 5-point stencil. The solid lines between grid points are the communicating perimeter for the 5-point stencil ($3s$). The 9-point stencil requires the following additional communications: 1 to the northeast, 1 to the southeast, 1 to the northwest, 1 to the southwest, and $s-2$ to the south. This yields a total communicating "perimeter" for an interior triangular partition with the 9-point stencil of $4s+2$ or $\frac{4\sqrt{2n}}{\sqrt{p}}+2$. "Perimeter" is perhaps a misnomer here, for the perimeter of points along the diagonal in Figure 2.6 is "two deep" for the 9-point stencil.

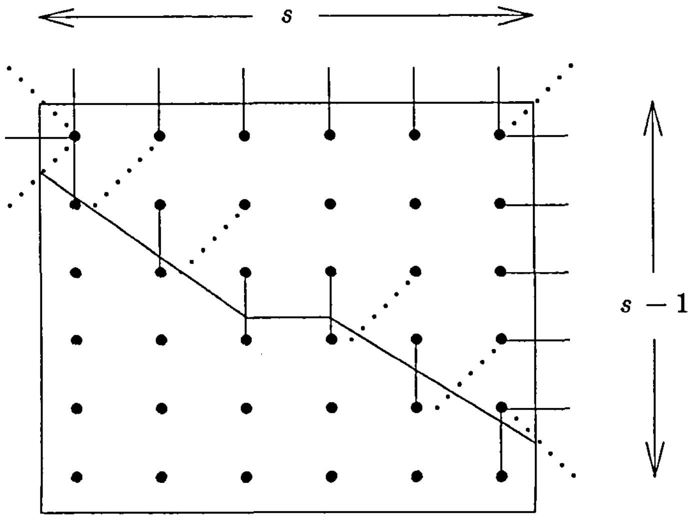


Figure 2.6 Triangular partitions with 9-point star stencil

2.2.3. Hexagonal Partitions

The dashed lines in Figure 2.7 illustrate the the additional communications required with hexagonal partitions when using the 9-point stencil instead of the 5-point stencil. The solid lines of Figure 2.7 correspond to the communicating perimeter of the 5-point stencil, shown to be $6l + 2$ in section 2.1.3. The 9-point stencil requires l communications to the northeast, southeast, southwest, and northwest in addition to those for the 5-point stencil. This gives a total communicating perimeter, for interior hexagonal partitions, of

$$10l + 2 = \frac{5n}{\sqrt{p}} + 2$$

where

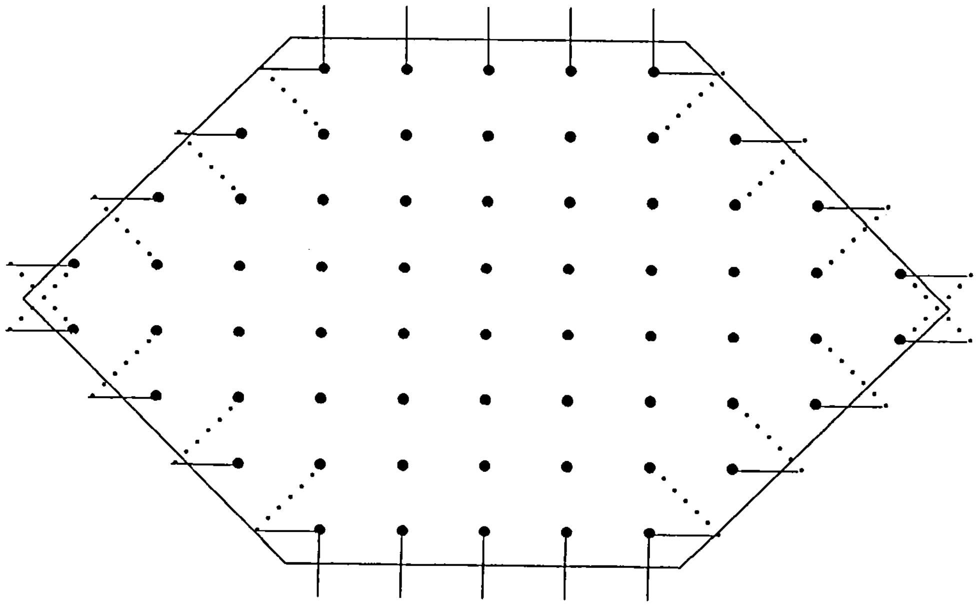


Figure 2.7 Hexagonal partitions with 9-point star stencil

$$l = \frac{n}{2\sqrt{p}}.$$

Note that the communicating "perimeter" is depth 2 along four of the six edges.

2.3. Other Stencils

Many stencils other than the 5-point and 9-point stencils analyzed above are frequently used when solving partial differential equations. Figure 2.8 illustrates some of the most common. For brevity's sake, we do not include the analysis of the communication required for their associated partitions. However, the results of this analysis are summarized in Table 2. The interested reader can verify these results by applying the methods discussed earlier to compute the additional grid points involved in data transfer for each of these stencils.

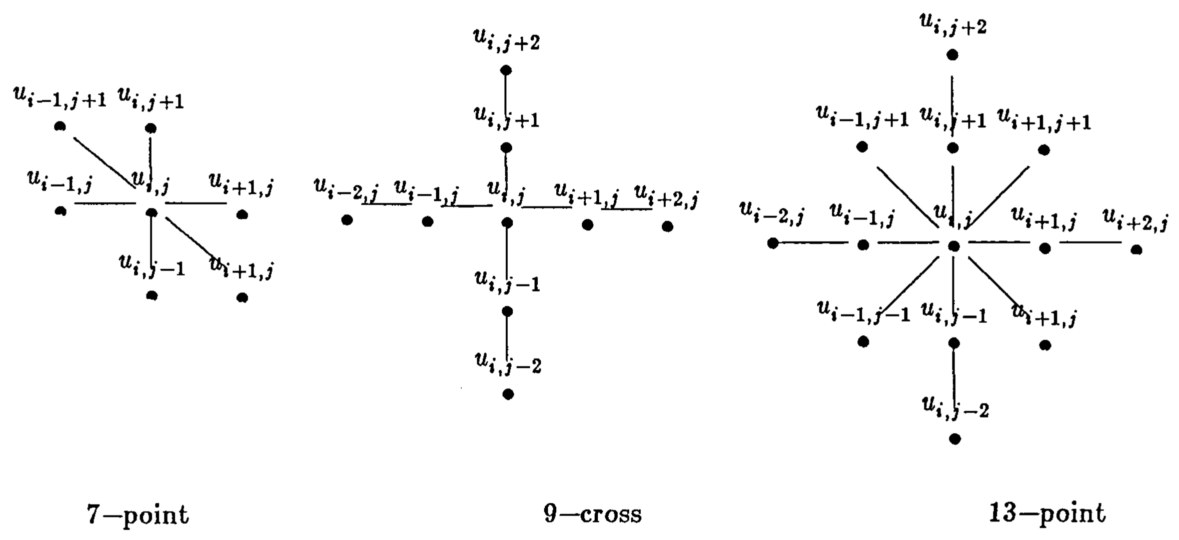


Figure 2.8 Frequently used discretization stencils

2.4. Computation/Communication Ratios

Before summarizing the results of the previous section, we introduce the notation shown in Table 1. Using this notation, Table 2 shows relative amounts of computation and communication for selected stencil/partition pairs. For simplicity, the effects of boundaries on communication have been elided. (Recall that n^2 is the number of grid points, and p is the number of processors.) Table 2 also includes one quantity not discussed earlier, parallel communication, the amount of data transfer if partition sides can communicate in parallel. This parallel communication will later allow us to determine if the optimal stencil/partition changes when communication to neighboring partitions can be done in parallel.

The entries of most interest in Table 2 are the ratio of computation to communication (R) and the ratio of computation to parallel communication (PR). Table 3 illustrates the relative magnitude of these quantities for a square grid containing 256×256 points and a parallel system with 64 processors.

Table 1 Static scheduling notation

Quantity	*Definition*
$Comp$	$\frac{n^2}{p}$, the computational complexity of a stencil/partition pair
$Comm$	communication complexity of a stencil/partition pair
$Pcomm$	parallel communication complexity of a stencil/pair
R	the ratio $\frac{Comp}{Comm}$
PR	the ratio $\frac{Comp}{Pcomm}$

Table 2 Summary of stencil/partition analysis

Partition	Stencil				
	5-point	*7-point*	*9-point star*	*9-point cross*	*13-point*
Rectangular Strips					
Comm:	$2n$	$2n$	$2n$	$4n$	$4n$
Pcomm:	n	n	n	$2n$	$2n$
R:	$\frac{n}{2p}$	$\frac{n}{2p}$	$\frac{n}{2p}$	$\frac{n}{4p}$	$\frac{n}{4p}$
PR:	$\frac{n}{p}$	$\frac{n}{p}$	$\frac{n}{p}$	$\frac{n}{2p}$	$\frac{n}{2p}$
Triangle					
Comm:	$\frac{3\sqrt{2}n}{\sqrt{p}}$	$\frac{3\sqrt{2}n}{\sqrt{p}}+2$	$\frac{4\sqrt{2}n}{\sqrt{p}}+2$	$\frac{6\sqrt{2}n}{\sqrt{p}}$	$\frac{6\sqrt{2}n}{\sqrt{p}}+2$
Pcomm:	$\frac{\sqrt{2}n}{\sqrt{p}}$	$\frac{2\sqrt{2}n}{\sqrt{p}}-2$	$\frac{2\sqrt{2}n}{\sqrt{p}}-2$	$\frac{2\sqrt{2}n}{\sqrt{p}}-1$	$\frac{2\sqrt{2}n}{\sqrt{p}}-1$
R:	$\frac{n}{3\sqrt{2p}}$	$\simeq\frac{n}{3\sqrt{2p}}$	$\simeq\frac{n}{4\sqrt{2p}}$	$\frac{n}{6\sqrt{2p}}$	$\simeq\frac{n}{6\sqrt{2p}}$
PR:	$\frac{n}{\sqrt{2p}}$	$\simeq\frac{n}{2\sqrt{2p}}$	$\frac{n}{2\sqrt{2p}}$	$\frac{n}{2\sqrt{2p}}$	$\simeq\frac{n}{2\sqrt{2p}}$
Square					
Comm:	$\frac{4n}{\sqrt{p}}$	$\frac{4n}{\sqrt{p}}+2$	$\frac{4n}{\sqrt{p}}+4$	$\frac{8n}{\sqrt{p}}$	$\frac{8n}{\sqrt{p}}+4$
Pcomm:	$\frac{n}{\sqrt{p}}$	$\frac{n}{\sqrt{p}}$	$\frac{n}{\sqrt{p}}$	$\frac{2n}{\sqrt{p}}$	$\frac{2n}{\sqrt{p}}$
R:	$\frac{n}{4\sqrt{p}}$	$\simeq\frac{n}{4\sqrt{p}}$	$\simeq\frac{n}{4\sqrt{p}}$	$\frac{n}{8\sqrt{p}}$	$\simeq\frac{n}{8\sqrt{p}}$
PR:	$\frac{n}{\sqrt{p}}$	$\frac{n}{\sqrt{p}}$	$\frac{n}{\sqrt{p}}$	$\frac{n}{2\sqrt{p}}$	$\frac{n}{2\sqrt{p}}$
Hexagon					
Comm:	$\frac{3n}{\sqrt{p}}+2$	$\frac{4n}{\sqrt{p}}+2$	$\frac{5n}{\sqrt{p}}+2$	$\frac{6n}{\sqrt{p}}+4$	$\frac{6n}{\sqrt{p}}+8$
Pcomm:	$\frac{n}{2\sqrt{p}}+1$	$\frac{n}{\sqrt{p}}$	$\frac{n}{\sqrt{p}}$	$\frac{n}{\sqrt{p}}+2$	$\frac{n}{\sqrt{p}}+2$
R:	$\simeq\frac{n}{3\sqrt{p}}$	$\simeq\frac{n}{4\sqrt{p}}$	$\simeq\frac{n}{5\sqrt{p}}$	$\simeq\frac{n}{6\sqrt{p}}$	$\simeq\frac{n}{6\sqrt{p}}$
PR:	$\simeq\frac{2n}{\sqrt{p}}$	$\frac{n}{\sqrt{p}}$	$\frac{n}{\sqrt{p}}$	$\simeq\frac{n}{\sqrt{p}}$	$\simeq\frac{n}{\sqrt{p}}$

NOTE: Comp $= n^2/p$ is used in computing R and PR in all cases.

An inspection of Table 3 shows that hexagonal partitions yield the highest ratio of computation to serial communication, except for the 9-point star stencil, where squares are better. However, if one assumes the inter-partition communication can be done in parallel (i.e., all edges of a partition can be transmitted in parallel), hexagons yield the highest ratio in all cases. With parallel communication, the improvement obtained with hexagons is even greater (e.g., $\frac{R_{hexagon}}{R_{square}} = 1.33$ for the 5-point stencil but $\frac{PR_{hexagon}}{PR_{square}} = 2$).

The patterns in Table 3 suggest there is some formal relation between partitions and stencils, with certain combinations preferred. In the next section we develop techniques for selecting optimal partition/stencil combinations.

Table 3
Ratio of computation to communication ($n = 256$ and $p = 64$)

Partition Type	**Stencil**				
	5-point	7-point	9-point star	9-point cross	13-point
Rectangle					
R:	2	2	2	1	1
PR:	4	4	4	2	2
Triangle					
R:	7.5	7.5	5.65	3.75	3.75
PR:	22.5	11.3	11.3	11.3	11.3
Square					
R:	8	8	8	4	4
PR:	32	32	32	16	16
Hexagon					
R:	10.66	8	6.4	5.3	5.3
PR:	64	32	32	32	32

3. Determining Optimal Stencil/Partition Pairs

Using the following definition, a partition can be categorized *with respect to a given stencil* by the number of partition perimeters that must be communicated.

Definition: A partition is a *k-partition with respect to stencil S* if k perimeters are communicated when stencil S is used.

For example, the square is a 1–partition with respect to the 5–point, 7–point, and 9–point star stencils but is a 2–partition with respect to the 9–point cross and 13–point stencils. The hexagon is a 1–partition for the 5–point and a 2–partition with respect to the 9–point cross and 13–point stencils.

Moreover, the value of k can be a fraction. The hexagon, for example, is a $1\frac{2}{6}$ partition for the 7–point stencil and a $1\frac{4}{6}$ partition with the 9–point star stencil. Why? Because only some sides of the hexagon are involved in multiple data transfers. This categorization of partitions with respect to stencils provides a ranking mechanism for stencil/partition pairs. Hence, one can determine those stencils where l–partition hexagons are preferable to k–partition squares.

When communication from a partition to each of its neighboring partitions is done serially, the communicating perimeter for square k–partitions is nearly $\frac{4kn}{\sqrt{p}}$, and the corresponding ratio of computation to serial communication is $\frac{n}{4k\sqrt{p}}$. The communicating perimeter for hexagonal l–partitions is approximately $\frac{3ln}{\sqrt{p}}$, and the corresponding ratio of computation to serial communication is $\frac{n}{3l\sqrt{p}}$. Clearly, an l–partition hexagon yields a higher ratio when

$$\frac{n}{3l\sqrt{p}} > \frac{n}{4k\sqrt{p}}$$

or when

$$k > \frac{3}{4}l. \tag{3.1}$$

If one adopts parallel rather than serial communication, the communicating perimeter for square k-partitions is, except for a small constant, $\frac{kn}{\sqrt{p}}$, and the ratio of computation to parallel communication is $\frac{n}{k\sqrt{p}}$. Similarly, the communicating perimeter for hexagonal l-partitions is $\frac{ln}{2\sqrt{p}}$ and the corresponding ratio of computation to parallel communication is $\frac{2n}{3l\sqrt{p}}$. With parallel communication, l-hexagons are preferable to k-squares when

$$\frac{2n}{l\sqrt{p}} > \frac{n}{k\sqrt{p}}$$

or

$$k > \frac{l}{2}. \tag{3.2}$$

Using inequalities (3.1) and (3.2), Table 4 shows optimal stencil/partition pairs, based on the maximum ratio of computation to communication. Table 4 shows that square partitions are better than hexagons in only one of the 10 cases. Note that the k and l-values for parallel communication in Table 4 were obtained by rounding the fractional values for serial communication up to the next largest integer (i.e., a parallel communication of $1\frac{2}{6}$ perimeters requires two transmissions). Based solely on Table 4, hexagonal partitions are superior to square partitions because they minimize the interpartition data transfer.[3] Similarly, triangles are clearly inferior.

[3]As we shall see, the underlying parallel architecture also influences the choice of partition shape.

Table 4 Comparison of Square and Hexagonal Partitions

Stencil	Square k–value (serial, parallel)	Hexagon l–value (serial, parallel)	Optimal partition serial: $k > \frac{3l}{4}$	Optimal partition parallel: $k > \frac{l}{2}$
5-point	(1,1)	(1,1)	hexagon $(1 > \frac{3}{4})$	hexagon$(1 > \frac{1}{2})$
7-point	(1,1)	$(1\frac{2}{6},2)$	equal $(1 = \frac{3}{4}\cdot\frac{4}{3})$	equal $(1 = \frac{1}{2}2)$
9-point star	(1,1)	$(1\frac{4}{6},2)$	square$(1 < \frac{3}{4}\cdot\frac{10}{6})$	equal $(1 = \frac{1}{2}2)$
9-point cross	(2,2)	(2,2)	hexagon $(2 > \frac{3}{4}\cdot 2)$	hexagon $(2 > 1)$
13-point	(2,2)	(2,2)	hexagon $(2 > \frac{3}{4}\cdot 2)$	hexagon $(2 > 1)$

4. Architecture and the Performance of Stencil/Partition Pairs

Our previous analysis did not include architectural considerations, save for the inclusion of results for both serial and parallel communication. However, the stencil and grid partition cannot be divorced from the processor connectivity of a message passing architecture (e.g., square or hexagonal grid) or the storage schema used in a shared memory multiprocessor. Optimal performance can be achieved only via judicious selection of a trio: stencil, partitioning, and architecture.

Deriving expressions for parallel execution times and speed–ups for a stencil/partition/architecture trio requires a model of execution. Our parallel execution time model is a variation of one we developed earlier [Reed85] and is similar to the one used by Vrsalovic, *et al.* [Vrsa85]. In this model, the parallel iteration time for evaluating one partition of grid points is

$$t_{cycle}^{p-processor} = t_{comp} + t_a + t_w$$

where t_{comp} is the iteration computation time, t_a is the data access/transfer time, and t_w is the waiting/synchronization time.

The computation time t_{comp} depends on the partition size and stencil, and is independent of the architecture except for the time, T_{fp}, to execute a floating point operation. Formally, t_{comp} is

$$t_{comp} = \frac{E(S)n^2}{p} T_{fp}$$

where $E(S)$ is the number of floating point operations required to update the value of a grid point, given a stencil S, $\frac{n^2}{p}$ is the number of grid points in a partition, and T_{fp} is the time for a single floating point operation.

The *speedup* obtained using parallel iterations is simply

$$S_p = \frac{t_{cycle}^{uniprocessor}}{t_{cycle}^{p-processor}}, \tag{4.1}$$

where the single processor iteration time is just

$$t_{cycle}^{uniprocessor} = E(S)n^2 T_{fp}.$$

Specific values for the speedup depend not only on the trio of stencil, partition, and network chosen, but also on the technology constants (e.g., floating point operation time and packet transmission time).

The other components of the execution time model, t_a and t_w, depend on the particular combination of partitioning, stencil, and architecture and are analyzed below.

4.1. Message Passing Architectures

Among the competing classes of parallel machines, message passing architectures occupy an important niche. The recent emergence of commercial message passing machines (.e.g., the Intel hypercube [Ratt85]) has stimulated great interest in this area.

Each processor in a message passing machine contains a local memory and is connected to a (necessarily) small number of other processors. Access to data contained in another processor's memory requires transfer of that data via the interconnection network. Clearly, the performance of a stencil/partition pair depends heavily on the performance of the interconnection network of the multiprocessor system. Although a plethora of interconnection networks have been proposed [Reed83, Witt81], Figure 4.1 shows those networks (meshes) that are directly relevant to iterative solution of elliptic partial differential equations. Each interconnection network has an associated "natural" partition (e.g., square partitions on a square mesh).

Consider an interior processor in one of the partition/mesh pairs. During each iteration (cycle), two groups of data must cross each communications link, one in each direction from neighboring processors. There are several possible interleavings of computation and remote data access. These range from a separate request for each communicating "perimeter" grid point when it is needed to a request for an entire "side" of the communicating "perimeter" of the partition. These requests can, in turn, be either overlapped or non–overlapped with computation. Similarly, the hardware support for interprocessor communication must be specified. A simple hardware design allows only one link connected to each processor to be active at any time, increasing the data transfer time. With additional hardware, each processor link can be simultaneously active.

Each combination of data access patterns and hardware design alternatives leads to an implementation with different performance characteristics. Rather than cursorily examine a wide variety of alternatives, we have chosen to examine a smaller set in detail. Specifically, we assume

- communication links are half–duplex (i.e., data can flow along links in only one direction at a time) and

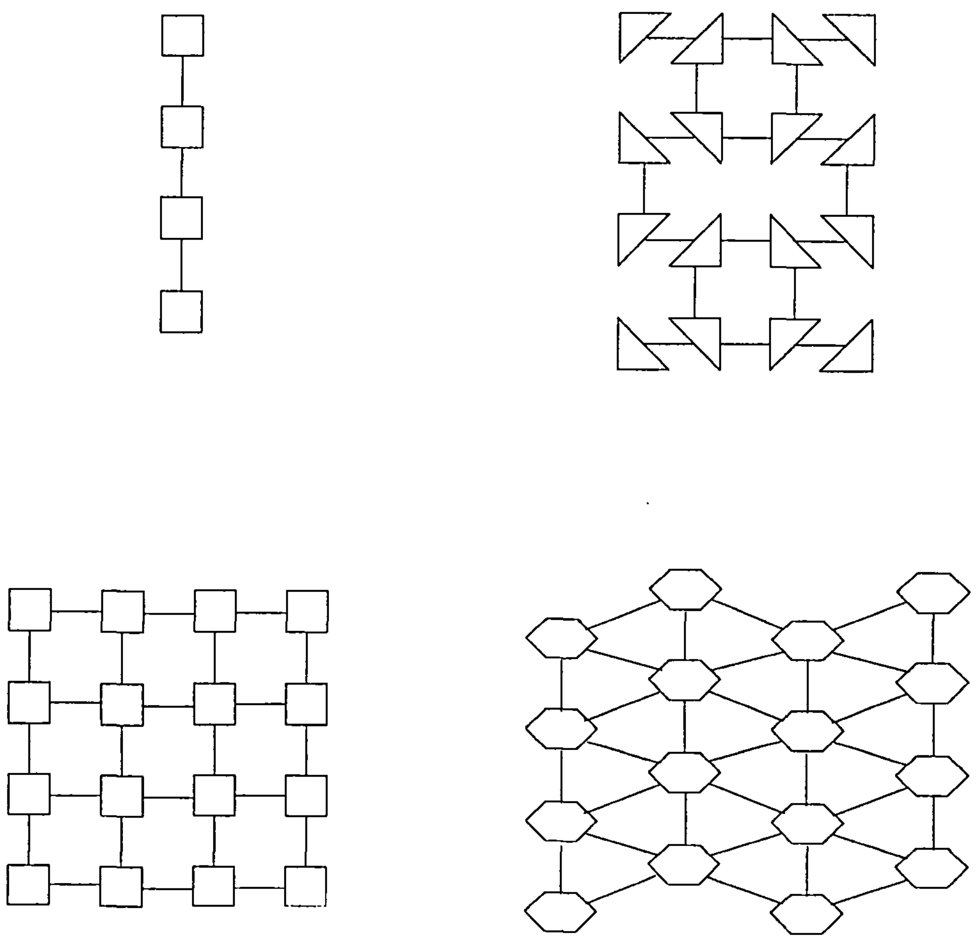


Figure 4.1 Selected interconnection networks

- processors request and wait for all perimeter values before starting computation.

Currently, these assumptions correspond to all commercial hypercube implementations [Ratt85].

Whether the communication is serial or parallel, some processor P_i in the interior of the network will need data from another processor P_j that is some number of links l_{ij} away. (See Table 5 for notation.) The amount of data to be transmitted, $d_{ij}(S, P)$, depends on both the stencil S and the grid partitioning P. Ignoring synchronization and queueing delays, the time to transmit data from P_i to P_j, crossing l_{ij} links, is

Table 5 Execution time model notation

Quantity	*Definition*
$d_{ij}(S,P)$	amount of data sent from i to j
l_{ij}	number of links between i and j
P	partition
P_i	processor i
Ps	packet size
S	discretization stencil
S_p	speedup
T_{fp}	time for a single floating point operation
$t_a^{parallel}$	parallel access time
t_a^{serial}	serial access time
t_{comm}	time to send a packet across one communication link
t_{cycle}	time for one iteration
$t_{forward}$	time (possibly zero) to interrupt an intermediate processor and forward a message
$t_{send}(i,j)$	data transmission time from processor i to j
$t_{startup}$	overhead for preparing a communication
t_w^{serial}	serial waiting time

$$t_{send}(i,j) = t_{startup} + \left\lceil \frac{d_{ij}(S,P)}{Ps} \right\rceil l_{ij} t_{comm} + (l_{ij} - 1) t_{forward}, \tag{4.2}$$

where $t_{startup}$ is the fixed overhead for sending data, t_{comm} is the packet transmission time, and $t_{forward}$ is the message forwarding overhead incurred at intermediate processors. The ceiling function reflects the redundant communication due to the fixed packet size Ps.

In general, data destined for other processors *will* encounter queueing delays, both at their origin and at intermediate nodes. The latter is expected, but the former is counter-intuitive. As an example of this phenomenon, consider the mapping of hexagonal partitions onto either a square or hexagonal mesh. On a square mesh, data from the six sides of the hexagons must exit via only four connecting links. Even with all links simultaneously active, some data will be delayed.

With hexagonal partitions on a hexagonal mesh, each partition edge is directly connected to its six neighboring partitions. However, each pair of neighbors must *exchange* data. Thus, two transmission delays are needed on each of the six links before exchanges are complete. If all links can simultaneously be active, two transmission delays will suffice to exchange boundary values. Conversely, twelve delays will be needed if only one link per processor can be active at any time.

There are two general approaches to managing the interpartition communication problem. The first relegates management of message passing and the associated queueing of messages for available links to system software residing in each processor. With this approach, each partition simply passes the data to be delivered to other partitions to the system software. No consideration is given to the pattern of communication in time. As an example, each partition might successively send boundary values on each of its links, then await receipt of boundary values from neighboring partitions. Although this approach is attractive from a programming standpoint, it hides the performance issues and may lead to increased contention for communication links.

The second approach requires programming the exchange of partition boundaries in a series of *phases*, each phase corresponding to a particular pattern of communication. In the example of hexagonal partitions on a hexagonal mesh, discussed above, the communication pattern of neighboring partitions would be alternating sends and receives. Sender and receiver would cooperate, each expecting the action of the other. This *pseudo-SIMD* mode of communication leads to regular communication patterns with minimal delays. Application of this approach is the subject of the next section.

4.2. Message Passing Analysis

Because the range of partition and network possibilities is so large, we have opted to present only the analysis of the 5–point stencil with square and hexagonal partitions on square and hexagonal interconnection networks. The triangular partitions were omitted because, as a cursory examination of Figure 2.3 shows, they require data transmission to four adjacent partitions. Because square partitions also transmit to only four adjacent partitions and have a higher ratio of computation to communication, they are always preferable to triangles. The analysis for 9–point stencils is similar to that presented below; only the case analysis is more complex.

When partitions are mapped onto an interconnection network, the processors may permit communication on only one link or on all. In the following we consider only the serial case; similar analysis applies to simultaneous communication on all links.

We begin with the simplest case: square partitions on a square interconnection network. Each partition must exchange $\frac{n}{\sqrt{p}}$ values with each of its four neighbors. Because only one link per processor can be active, we expect the data exchange to require four phases (i.e., time proportional to $\frac{4n}{\sqrt{p}}$). However, this would require *all* processors to simultaneously send and receive. At any given instant, only half the processors can send; the other half must receive. Hence, eight phases are needed, and the total time for data exchange is

$$t_a^{serial} + t_w^{serial} = 4t_{startup} + 8\left\lceil \frac{n}{Ps\sqrt{p}} \right\rceil t_{comm}.$$

Four startup costs are needed to initiate message transmissions to neighboring processors. Because square partitions map directly onto the square mesh, no intermediate node forwarding costs arise. Because the square mesh can be directly embedded in the hexagonal mesh, the data exchange delay for square partitions on a hexagonal mesh is identical to that for the square mesh.[4]

Like square partitions on a square mesh, hexagonal partitions map directly onto a hexagonal mesh. Recalling that the north and south sides of a hexagon contain $\frac{n}{2\sqrt{p}} + 1$ points, and the other four sides contain $\frac{n}{2\sqrt{p}}$ points each, the data exchange delay is

$$t_a^{serial} + t_w^{serial} = 6t_{startup} + 4\left\lceil \frac{\left\lceil \frac{n}{2\sqrt{p}} + 1 \right\rceil}{Ps} \right\rceil t_{comm} + 8\left\lceil \frac{\frac{n}{2\sqrt{p}}}{Ps} \right\rceil t_{comm}.$$

The first ceiling term corresponds to the north/south exchange and requires four phases. Similarly, the second term represents the exchange of data along the four diagonal connections and requires eight phases.

Finally, hexagonal partitions can also be mapped onto a square mesh. Unlike the other mappings, this one requires data exchange between non-adjacent processors. In this case, we assume that rows of hexagons are mapped onto corresponding rows of the square mesh. With this mapping, north/south connections and half the diagonal connections are realized directly. The remaining diagonal connections require traversal of two links to "turn the corner" in the square mesh. Hence, the total communication delay due to data exchange is

$$t_a^{serial} + t_w^{serial} =$$

$$6t_{startup} + 4\left\lceil \frac{\left\lceil \frac{n}{2\sqrt{p}} + 1 \right\rceil}{Ps} \right\rceil t_{comm} + 4\left\lceil \frac{\frac{n}{2\sqrt{p}}}{Ps} \right\rceil t_{comm} + 8\left\lceil \frac{\frac{n}{2\sqrt{p}}}{Ps} \right\rceil t_{comm} + 4t_{forward}.$$

[4]This is only true for the 5-point stencil. With the 9-point and other stencils, the distinction between square and hexagonal meshes is important.

The first ceiling term corresponds to the north/south connections and the second to the directly connected diagonals, each with four phases. The third term represents indirectly connected diagonals, requiring eight phases. Half these phases require forwarding through intermediate nodes, hence the four forwarding costs.

As noted earlier, similar analysis can be applied to other meshes and stencils. Table 6 shows the number of other partitions with which each partition must communicate (i.e., the number of transmission startups). In addition, transmission delays are shown as a sum of terms. Each term is a product of the amount of data exchanged between logically adjacent partitions and the number of phases necessary to accomplish the exchange. In the table, the potential effects of packet size on transmission delay are ignored, as are the times for startup and forwarding. Table 6 suggests that hexagonal partitions are preferable for 5–point stencils, and square partitions are more appropriate for 9–point stencils, confirming our earlier, mesh independent analysis. As we shall see, however, both the number of message startups *and* amount of data must be considered when estimating the performance of a stencil/partition/mesh trio.

4.3. An Evaluation of Stencils, Partitions and Meshes

Equation (4.2), the delay to send data, includes parameters for startup, forwarding cost, packet size, and packet transmission time. Because our primary interest is the effect of transmission time, we have ignored the effects of startup and forwarding (i.e., we have assumed those parameters are zero). When evaluating the relative performance of stencil/partition/mesh trios, we have attempted to use values for packet size and packet transmission time based on those for commercial message passing machines. For example, the Intel iPSC [Ratt85] sends 1K byte packets with a measured transmission time of between 6 and 7 milliseconds.

Table 6 Message passing data exchange

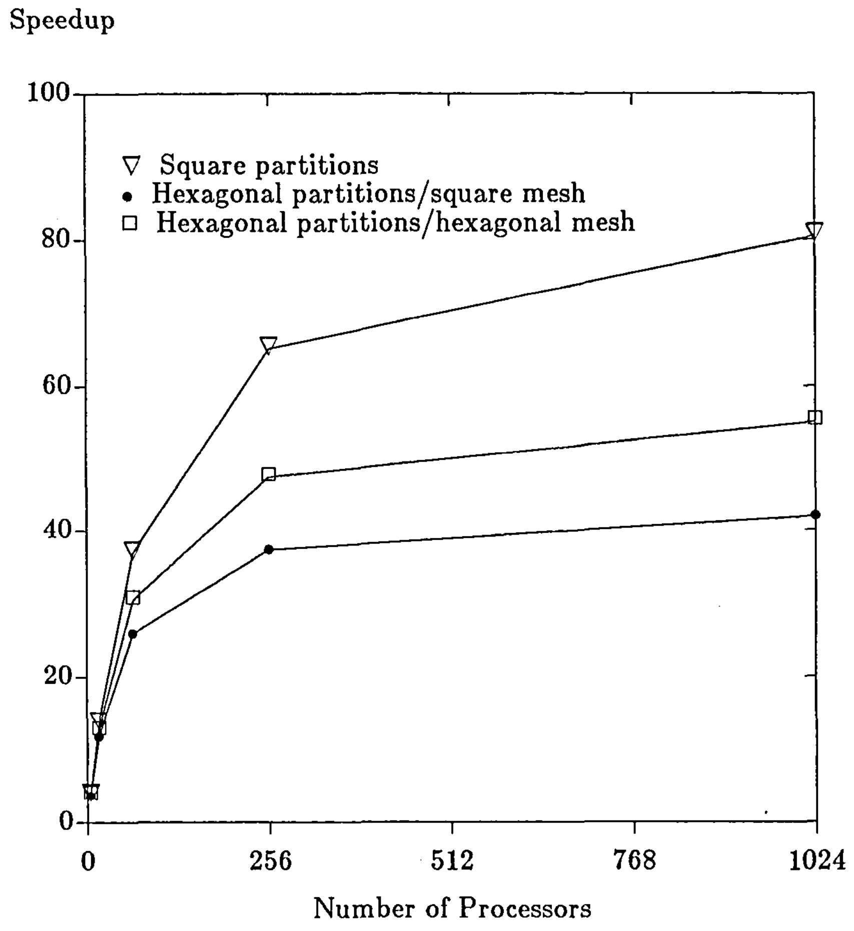
Partition	*Mesh*	*Stencil*	*Number of Communicating Partitions*	*Phase-Data Transmission Products*	*O(Expected Delay)*
Square	Square	5-point	4	$\frac{8n}{\sqrt{p}}$	$\frac{8n}{\sqrt{p}}$
Square	Square	9-point	8	$\frac{8n}{\sqrt{p}} + 16$	$\frac{8n}{\sqrt{p}}$
Square	Hexagon	5-point	4	$\frac{8n}{\sqrt{p}}$	$\frac{8n}{\sqrt{p}}$
Square	Hexagon	9-point	8	$\frac{8n}{\sqrt{p}} + 12$	$\frac{8n}{\sqrt{p}}$
Hexagon	Square	5-point	6	$4\left[\frac{n}{2\sqrt{p}} + 1\right] + \frac{4n}{2\sqrt{p}} + \frac{8n}{2\sqrt{p}}$	$\frac{8n}{\sqrt{p}}$
Hexagon	Square	9-point	6	$4\left[\frac{n}{2\sqrt{p}} + 1\right] + \frac{4n}{\sqrt{p}} + \frac{8n}{\sqrt{p}}$	$\frac{14n}{\sqrt{p}}$
Hexagon	Hexagon	5-point	6	$4\left[\frac{n}{2\sqrt{p}} + 1\right] + \frac{8n}{2\sqrt{p}}$	$\frac{6n}{\sqrt{p}}$
Hexagon	Hexagon	9-point	6	$4\left[\frac{n}{2\sqrt{p}} + 1\right] + \frac{8n}{\sqrt{p}}$	$\frac{10n}{\sqrt{p}}$

Figure 4.2 shows the speedup, obtained using (4.1), of square and hexagonal partitions on both square and hexagonal meshes, using a 5-point stencil. In the figure, 1K byte packets are used. We see that *square* partitions yield significantly larger speedup than hexagonal partitions, regardless of the underlying mesh. This is counter-intuitive and would seem to contradict Table 6. Careful inspection of (4.2), however, shows that packet size is crucial. The term

$$\left\lceil \frac{d_{ij}(S, P)}{Ps} \right\rceil l_{ij}\, t_{comm}$$

in (4.2) accounts for the discretization overhead caused by packets. If Ps, the packet size, is large, the number of partitions that must receive data from each partition is much more important than the total amount of data to be sent. For example, sending 4 bytes to 6 partitions

Figure 4.2
Speedup for 5-point stencil
(1024 X 1024 grid with 1024 byte packets)



Parameter	Value
Packet size	1024
Startup	0.0
Forwarding	0.0
Packet transmission	6×10^{-3} *sec*
Floating point operation	1×10^{-6} *sec*

is much more expensive than sending 6 bytes to 4 partitions if the packet size is 1024 bytes. The former requires 6 packet transmissions, the latter only 4 transmissions. Square partitions, because they require communication with only four neighboring partitions, are preferable to hexagonal partitions with six neighboring partitions, *even though more data must be transmitted with square partitions.*

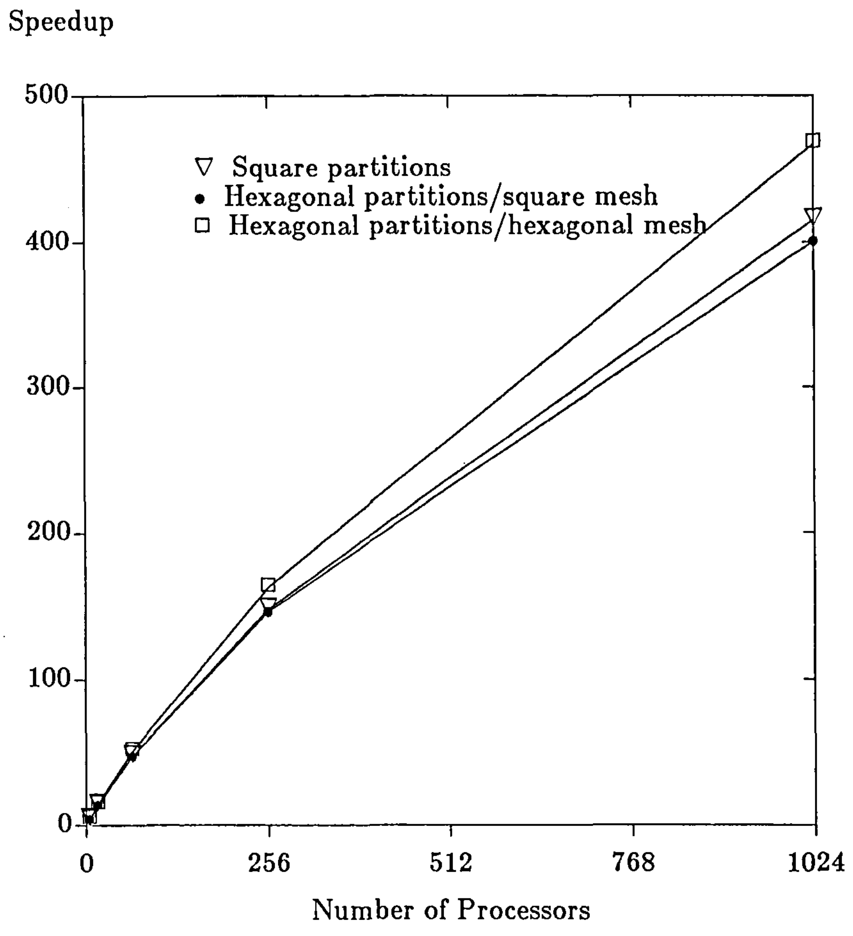
With a 4 byte floating point representation, a 1024×1024 grid, 1024 byte packets, and square partitions (the assumptions of Figure 4.2), using more than 16 partitions will not decrease the communication delay because, beyond this point, the total number of packet transmissions does not change. Instead, the ratio of useful computation to communication begins to degrade.

As the packet size decreases, we would expect the differential in amount of transmitted data to become more important. For small enough packets, the total amount of data accurately reflects the delay. Figure 4.3 shows just this result. For smaller 16 byte packets used in the figure, hexagonal partitions are preferred over square partitions.

Comparing Figures 4.2 and 4.3, we also see the effects of varying the number of processors. For a small number of processors, the iteration is *compute bound.* As the processors (and partitions) increase, the distinction between differing partition shapes becomes apparent. With 1024 processors, only one grid point resides in each partition, and the effects of packet size on performance are striking.

Figures 4.4 and 4.5 illustrate phenomena similar to those in Figures 4.2 and 4.3. For large packet sizes, Figure 4.4, hexagonal partitions are preferable to square partitions because the hexagons communicate with only six other hexagons, rather than eight other squares. However, the square partitions require less interpartition data transfer. Only when the packet size becomes small, Figure 4.5, does the potential advantage of square partitions become apparent.

Figure 4.3
Speedup for 5-point stencil
(1024 X 1024 grid with 16 byte packets)



Parameter	**Value**
Packet size	16
Startup	0.0
Forwarding	0.0
Packet transmission	9.375×10^{-5} *sec*
Floating point operation	1×10^{-6} *sec*

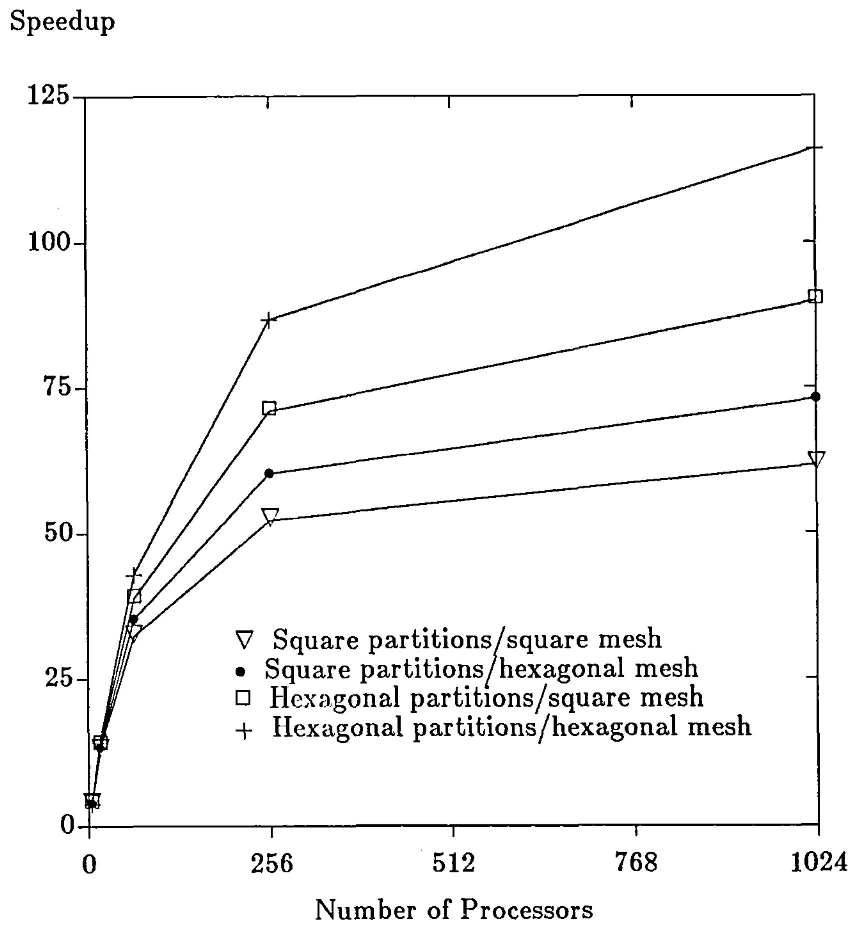
Stencil, partition, mesh, and hardware parameters interact in non-intuitive ways. For 5-point stencils, square partitions, with their smaller number of communicating neighbors, are appropriate for large packet sizes. Likewise, hexagonal partitions, with smaller interpartition data transfer, are appropriate for small packet sizes. The reverse is true for 9-point stencils: hexagonal partitions are appropriate for large packet sizes (even though they are $1\frac{4}{6}$ partitions for the 9-point stencil), and square partitions are best for small packet sizes. The interaction of parameters cannot be ignored when considering the performance of an algorithm on a particular architecture.

Recognizing the interdependence of parameters, Saltz *et. al.* [Salt86] recently evaluated the Intel iPSC for solution of the heat equation using Successive Over Relaxation (SOR). They observed that performance on the iPSC, with its high transmission startup cost and large packets, varied greatly with the size of the grid and the shape of the grid partitions. For small grids, horizontal strips, although requiring more interpartition data transfer, were preferable to square partitions. Only when the grid became large did the advantage of square partitions become apparent. The reasons are precisely those observed in Figures 4.2 and 4.3: amount of data versus number of communicating partitions. This validation of our analytic techniques suggests that they can effectively be used to determine the appropriate combination of partition shape and size given the architectural parameters of the underlying parallel machine.

4.4. Shared Memory Architectures

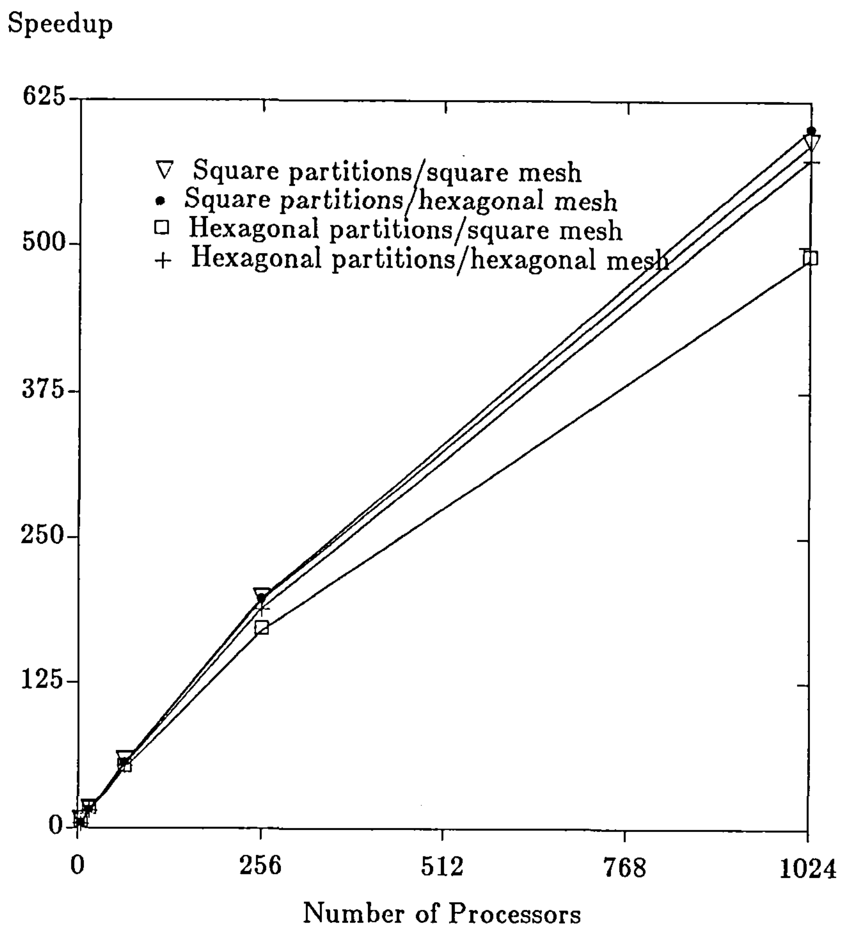
Unlike a message passing architecture where partitions exchange values via explicit messages, a shared memory implementation stores all partition values that must be exchanged in global, shared memory. The values associated with all other grid points are kept in memories local to each processor.

Figure 4.4
Speedup for 9–point stencil
(1024 X 1024 grid with 1024 byte packets)



Parameter	Value
Packet size	1024
Startup	0.0
Forwarding	0.0
Packet transmission	6×10^{-3} *sec*
Floating point operation	1×10^{-6} *sec*

Figure 4.5
Speedup for 9-point stencil
(1024 X 1024 grid with 16 byte packets)



Parameter	**Value**
Packet size	16
Startup	0.0
Forwarding	0.0
Packet transmission	9.375×10^{-5} *sec*
Floating point operation	1×10^{-6} *sec*

Just as for message passing, the iteration time for evaluating one partition of grid points is

$$t_{cycle}^{p-processor} = t_{comp} + t_a + t_w, \tag{4.3}$$

only the interpretation of the access (t_a) and waiting (t_w) times differ. With a message passing architecture, these times depend on the contention for communication links. Analogously, shared memory delays arise from memory contention. Vrsalovic, *et al.* [Vrsa85] observed that the expected waiting time for memory access takes the form

$$t_w = \begin{cases} \left[\frac{p}{C} - 1\right] t_a & \textit{synchronous} \\ max\,\{0, \left[\frac{p}{C} - 1\right] t_a - t_{comp}\} & \textit{asynchronous} \end{cases} \tag{4.4}$$

where C is the number of processors that can access shared memory simultaneously, and t_a is the memory access time. The synchronous case, where all processors simultaneously attempt to access global memory, forces one processor to wait until all others have accessed memory. The length of this delay depends on the number of simultaneous memory accesses supported. If the processors operate asynchronously, allowing overlap of computation and memory access, the level of memory contention is reduced. In the simplest case, a set of global memory modules connected to a shared bus, the number of concurrent memory accesses C is just 1. If a multistage switching network connects processors and memories, (4.4) can be replaced with a waiting time function [Krus83]. Whatever the interconnection network, t_w reflects the effects of memory contention. We will return to this later, but first we consider the expected amount of data transferred to/from shared memory.

When considering a shared memory implementation of an iteration technique, two choices arise: local copies of partition boundaries or only global storage. In the first case, each partition not only retains a copy of its boundary values after writing them to global memory but also copies into local memory all boundaries needed from other partitions. With only global storage, the boundary values are accessed in global memory each time they are needed. The performance of these two implementations differs considerably based on the stencil, partition, and memory access technique. Hence, we consider local copies and global access for both 5-point and 9-point stencils with rectangular strips, square, and hexagonal partitions.

For notational convenience, we let $g_a^{local\ copies}$ be the number of global memory accesses for copying boundary values to and from a partition (assuming local copies), $g_a^{no\ copies}$ be the number of global memory accesses (assuming no local copies), t_g be the time to access one value from global memory, and t_l be the processor overhead associated with copying one boundary value. With this notation, the cycle time for one iteration is

$$t_{cycle}^{local\ copies} = \frac{E(S)n^2}{p} T_{fp} + g_a^{local\ copies} \cdot (t_g + t_l) + t_w \tag{4.5}$$

or

$$t_{cycle}^{no\ copies} = E(S)\frac{n^2}{p} T_{fp} + g_a^{no\ copies} \cdot t_g + t_w, \tag{4.6}$$

where the three terms correspond to those in (4.3). As with message passing implementations, $E(S)$ is the number of floating point operations required to update each grid point given stencil S, p is the number of partitions, and T_{fp} is the time needed for one floating point operation. When local copies are used, some processor overhead may be required to maintain the copies (e.g., copying from system buffers to user memory); this overhead is reflected by t_l. Finally, t_g and t_l are hardware parameters; only g_a depends on the choice of local copies or global access. Thus, we concentrate on derivations of $g_a^{local\ copies}$ and $g_a^{no\ copies}$ for selected combinations of stencils and partitions. The results, derived below, are summarized in Table 7.

Table 7 Shared memory data transfers

Stencil	*Rectangular Strip*	*Square*	*Hexagon*
5–point			
g_a^{local}	$4(n-2)$	$\frac{8n}{\sqrt{p}} - 4$	$\frac{6n}{\sqrt{p}}$
g_a^{global}	$12(n-2)$	$24\left[\frac{n}{\sqrt{p}} - 1\right]$	$\frac{18n}{\sqrt{p}} - 12$
9–point			
g_a^{local}	$4(n-2)$	$\frac{8n}{\sqrt{p}}$	$\frac{10n}{\sqrt{p}} - 4$
g_a^{global}	$20(n-2)$	$\frac{40n}{\sqrt{p}} + 8$	$\frac{48n}{\sqrt{p}} - 44$

The values of $g_a^{local\ copies}$ for both the 5–point and 9–point stencils can be easily determined from Figures 2.2, 2.4, and 2.7. For example, Figure 2.2(b) (with $r = \sqrt{p}$) shows that a square partition reads $\frac{n}{\sqrt{p}}$ data global memory values from each of its four neighbors and writes its own four boundaries back to global memory.[5] The total number of global memory accesses is then

$$\frac{4n}{\sqrt{p}} + 4\left[\frac{n}{\sqrt{p}} - 1\right] = \frac{8n}{\sqrt{p}} - 4.$$

The 9–point stencil is similar, requiring four extra boundary values, one from each of the diagonally adjacent partitions.

[5]The four corner points are written only once.

The situation changes dramatically if no local copies of boundary data are maintained. Boundary values are often used multiple times during an iteration. With a 5-point stencil and square partitions, updating a single element on the boundary generally requires access to three values on the partition's boundary and one access to another partition's boundary. The updated boundary element must then be rewritten. Hence, five memory accesses are required if no copies are maintained.

The penalty for not maintaining local copies is even more striking for 9-point stencils. Figure 4.6 shows the number of global memory accesses for each point in an interior square partition when the 9-point stencil is used and no local copies are maintained. The numbers are

---

Figure 4.6
Global memory accesses for 9-point stencil
(square partitions with no local copies)

9	8	7	7	7	7	8	9
8	5	3	3	3	3	5	8
7	3	0	0	0	0	3	7
7	3	0	0	0	0	3	7
7	3	0	0	0	0	3	7
7	3	0	0	0	0	3	7
8	5	3	3	3	3	5	8
9	8	7	7	7	7	8	9

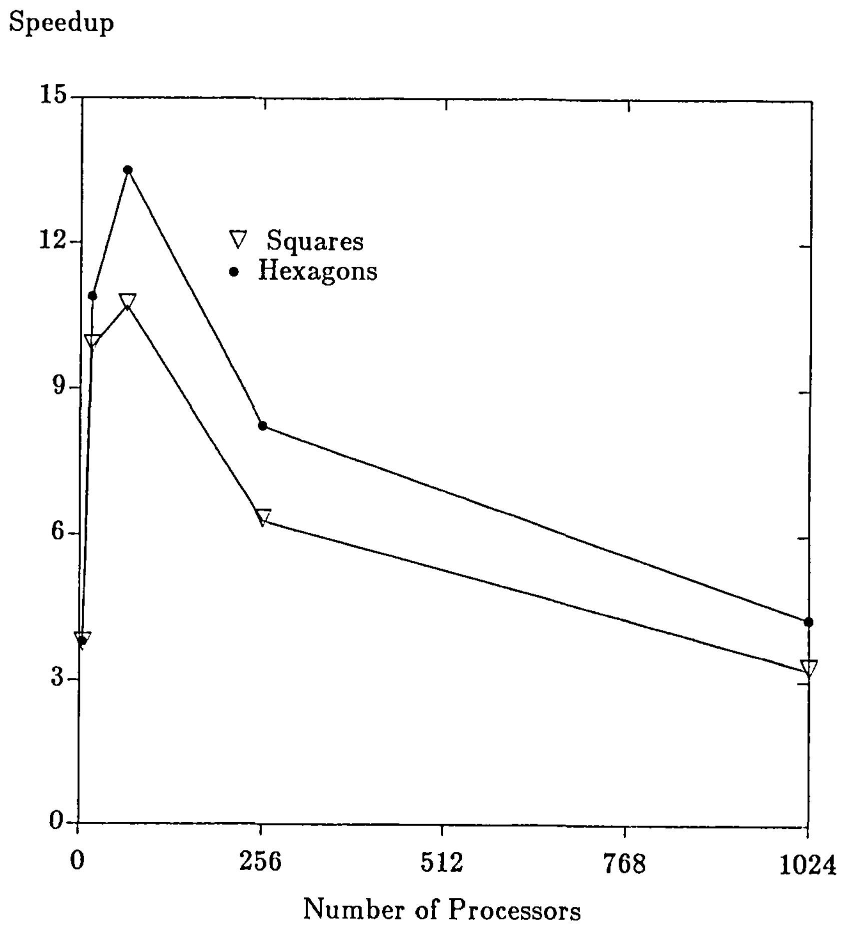
determined by counting boundary grid points needed to update the value at each grid point. If the grid point lies on the boundary, the count is increased by two: the old value must be be read from global memory and the new value written. Obtaining a general formula for the number of memory accesses is straightforward, given diagrams such at Figure 4.6. For a 9–point stencil with square partitions, $\frac{40n}{\sqrt{p}} + 8$ memory accesses are needed, a five–fold increase over that when local copies are maintained.

4.5. Shared Memory Analysis

Given an analysis of the memory traffic required for maintaining local copies or always accessing shared memory, only hardware parameters and some assumption about the underlying interconnection network are needed to predict performance. As noted earlier, the memory contention function t_w can reflect a variety of interconnection strategies ranging from a single global memory bus to a multistage interconnection network. Because the importance of local copies is most striking when memory contention is severe, we have concentrated on the worst case: a single bus connecting all global memories and processors.

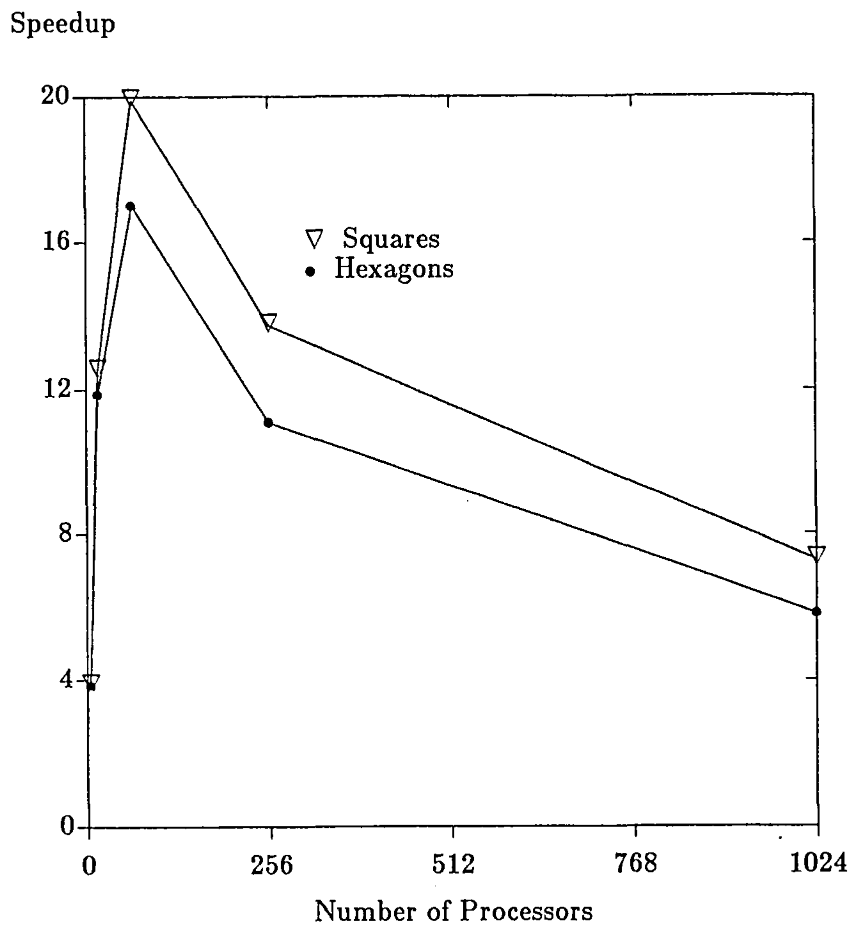
Figure 4.7 illustrates the speedup obtained for square and hexagonal partitions on a 5–point stencil with varying numbers of processors and local copies of partition boundaries. In the figure, access to global memory is assumed to require five times that for a single floating point operation. The figure confirms the analysis of Table 6 and 7: hexagons are the preferred partition type. For small numbers of processors, computation time dominates, and there is little distinction between the partition types. However, as the number of processors increases, the smaller interpartition data transfer required by hexagons makes their use attractive. Speedup increases with the number of processors until the global memory bus becomes a bottleneck; at that point speedup begins to decline.

Figure 4.7
Shared memory speedup for 5–point stencil
(local copies with grid size 1024 X 1024)



Parameter	Value
Global memory access time	5×10^{-6} *sec*
Local processing overhead	0.25×10^{-6} *sec*
Floating point operation	1×10^{-6} *sec*

Figure 4.8
Shared memory speedup for 9-point stencil
(local copies with grid size 1024 X 1024)



Parameter	**Value**
Global memory access time	5×10^{-6} *sec*
Local processing overhead	0.25×10^{-6} *sec*
Floating point operation	1×10^{-6} *sec*

Figure 4.8 shows a result similar to Figure 4.7, except for 9–point stencils. As Table 7 suggests, square partitions are preferred. A comparison of Figures 4.7 and 4.8 shows that the 9–point stencil gives larger absolute speedup. The reason is intuitive: the greater computation cost at each grid point more than offsets the increased communication cost for the 9–point stencil. Hence, an equal number of processors translates into a greater speedup.

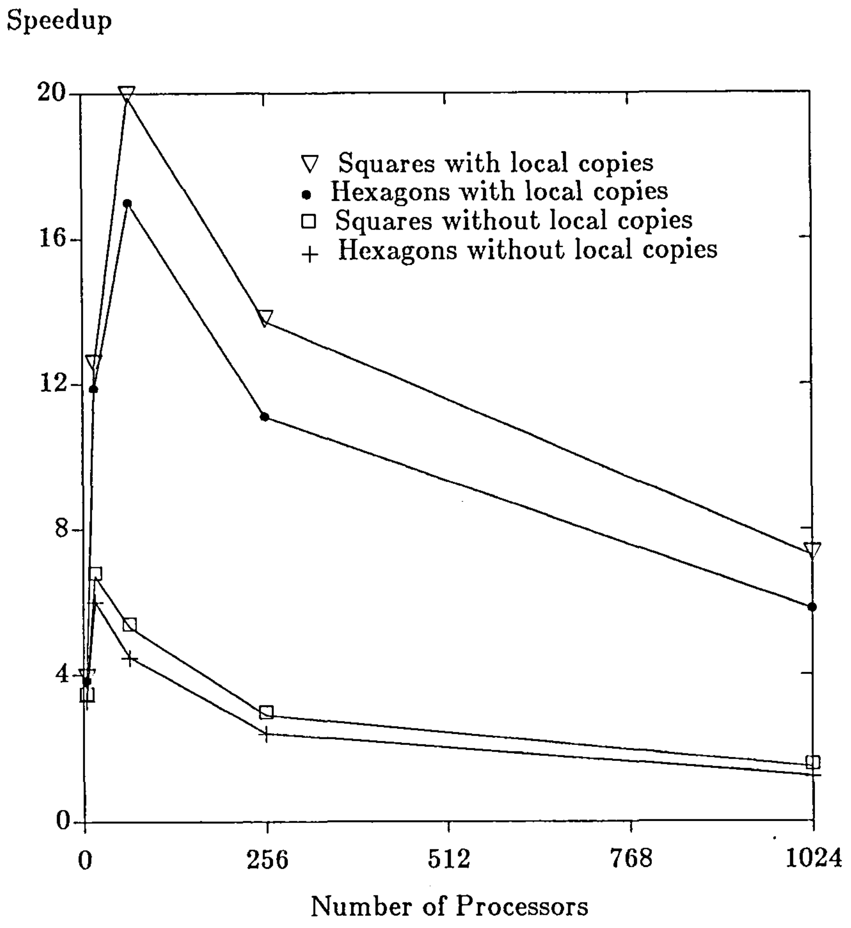
Finally, Figure 4.9 compares maintaining local copies of boundaries to continued access to global memory. This figure also confirms what Table 7 suggests; local copies are clearly advantageous. The argument for storing boundaries in local memories is compelling. Without such copies, the bandwidth of the global bus quickly saturates.

5. Conclusions

The trio of iteration stencil, grid partition shape, and underlying parallel architecture must be considered together when designing parallel algorithms for solution of elliptic partial differential equations. Isolated evaluation of one or even two components of the trio is likely to yield non–optimal algorithms.

We have seen, for example, that an abstract analysis of iteration stencil and partition shape suggests that hexagonal partitions are best for 5–point stencils, whereas square partitions are best for 9–point stencils. Further analysis shows that this is only true in a message passing implementation if small packets are supported. For large packets, the reverse is true (i.e., square partitions for 5–point stencils and hexagonal partitions for 9–point stencils). Likewise, the type of interconnection network is crucial. Mapping grid partitions onto a network that does not directly support the interpartition communication pattern markedly degrades performance. Finally, when considering shared memory implementation of the iterations, maintaining local copies of the partition boundaries is imperative. Without local copies, or an extremely fast

Figure 4.9
Speedup for 9–point stencil with grid size 1024 X 1024



Parameter	Value
Global memory access time	5×10^{-6} *sec*
Local processing overhead	0.25×10^{-6} *sec*
Floating point operation	1×10^{-6} *sec*

interconnection network, the observed speedups are extremely small. Consequently, only a small number of processors can be used effectively.

In summary, stencil, partition shape, and architecture must be considered in concert when designing an iterative solution algorithm. They interact in non-intuitive ways and ignoring one or more of the three almost certainly leads to sub-optimal performance.

References

[Fox84] G. C. Fox and S. W. Otto, "Algorithms for Concurrent Processors," *Physics Today,* Vol. 37, pp. 50-59, May 1984.

[Krus83] C. P. Kruskal, "The Performance of Multistage Interconnection Networks for Multiprocessors," *IEEE Transactions on Computers,* Vol. C-32, No. 12, pp. 1091-1098, December 1983.

[Orte85] J. Ortega and R. Voigt, "Solution of Partial Differential Equations on Vector and Parallel Computers," *SIAM Review,* Vol. 27, No. 2, pp. 149-240, June 1985.

[Ratt85] J. Rattner, "Concurrent Processing: A New Direction in Scientific Computing," *Conference Proceedings of the 1985 National Computer Conference,* AFIPS Press, Vol. 54, pp. 157-166, 1985.

[Reed83] D. A. Reed and H. D. Schwetman, "Cost-Performance Bounds on Multimicrocomputer Networks," *IEEE Transactions on Computers,* Vol. C-32, No. 1, pp. 85-93, January 1983.

[Reed85] D. A. Reed and M. L. Patrick, "Parallel, Iterative Solution of Sparse Linear Systems: Models and Architectures," *Parallel Computing,* Vol. 2, pp. 45-67, 1985.

[Salt86] J. H. Saltz, V. K. Naik, and D. M. Nicol, "Reduction of the Effects of the Communication Delays in Scientific Algorithms on Message Passing MIMD Architectures," *ICASE Report 86-4,* NASA Langley Research Center, to appear in the *SIAM Journal of Scientific and Statistical Computing.*

[Voig85] R. G. Voigt, "Where Are the Parallel Algorithms?" *Conference Proceedings of the 1985 National Computer Conference,* AFIPS Press, Vol. 54, pp. 329-334, 1985.

[Vrsa85] D. Vrsalovic, E. F. Gehringer, Z. Z. Segall, and D. P. Siewiorek, "The Influence of Parallel Decomposition Strategies on the Performance of Multiprocessor Systems," *Proceedings of the 12th Annual International Symposium on Computer Architecture,* ACM Sigarch Newsletter, Vol. 13, No. 3, pp. 396-405, June 1985.

[Witt81] L. D. Wittie, "Communication Structures for Large Multimicrocomputer Systems," *IEEE Transactions on Computers,* Vol. C-30, No. 4, pp. 264-273, April 1981.

1. Report No. NASA CR-178102 ICASE Report No. 86-24	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle STENCILS AND PROBLEM PARTITIONINGS: THEIR INFLUENCE ON THE PERFORMANCE OF MULTIPLE PROCESSOR SYSTEMS		5. Report Date May 1986
		6. Performing Organization Code
7. Author(s) Daniel A. Reed, Loyce M. Adams, and Merrell L. Patrick		8. Performing Organization Report No. 86-24
9. Performing Organization Name and Address Institute for Computer Applications in Science and Engineering Mail Stop 132C, NASA Langley Research Center Hampton, VA 23665-5225		10. Work Unit No.
		11. Contract or Grant No. NAS1-17070, NAS1-18107
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D.C. 20546		13. Type of Report and Period Covered Contractor Report
		14. Sponsoring Agency Code 505-31-83-01

15. Supplementary Notes

Langley Technical Monitor: J. C. South

Submitted to IEEE Trans. Comput.

Final Report

16. Abstract

Given a discretization stencil, partitioning the problem domain is an important first step for the efficient solution of partial differential equations on multiple processor systems. We derive partitions that minimize interprocessor communication when the number of processors is known a priori and each domain partition is assigned to a different processor. Our partitioning technique uses the stencil structure to select appropriate partition shapes. For square problem domains, we show that non-standard partitions (e.g., hexagons) are frequently preferable to the standard square partitions for a variety of commonly used stencils. We conclude with a formalization of the relationship between partition shape, stencil structure, and architecture, allowing selection of optimal partitions for a variety of parallel systems.

17. Key Words (Suggested by Authors(s)) discretization stencils, domain partitioning, parallel computing, elliptic equations	18. Distribution Statement 59 - Mathematical & Computer Sciences 62 - Computer Systems Unclassified - unlimited

19. Security Classif.(of this report) Unclassified	20. Security Classif.(of this page) Unclassified	21. No. of Pages 47	22. Price A03