

NASA Contractor Report 178024

ICASE REPORT NO. 85-55

NASA-CR-178024
19860010476

ICASE

FOR REFERENCE
NOT TO BE TAKEN FROM THIS ROOM

A PARTITIONING STRATEGY FOR NON-UNIFORM
PROBLEMS ON MULTIPROCESSORS

Marsha J. Berger
Shahid Bokhari

Contract No. NAS1-17070
November 1985

LIBRARY COPY
FEB 11 1986
LANGLEY RESEARCH CENTER
LIBRARY, NASA
HAMPTON, VIRGINIA

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

Operated by the Universities Space Research Association

NASA
National Aeronautics and
Space Administration
Langley Research Center
Hampton, Virginia 23665

ENTER:

18 1 1 RN/NASA-CR-178024

DISPLAY 18/6/1

86N19947** ISSUE 10 PAGE 1642 CATEGORY 59 RPT#: NASA-CR-178024

ICASE-85-55 NAS 1.26:178024 CNT#: NAS1-17070 DE-AC02-76ER-03077-V

85/11/00 30 PAGES UNCLASSIFIED DOCUMENT

UTTL: A partitioning strategy for nonuniform problems on multiprocessors TLSP:
Final Report

AUTH: A/BERGER, M. J.; B/BOKHARI, S. PAA: A/(New York Univ., N.Y.)

CORP: National Aeronautics and Space Administration. Langley Research Center,
Hampton, Va. AVAIL.NTIS

SAP: HC A03/MF A01

COI: UNITED STATES

MAJS: /*DATA PROCESSING EQUIPMENT/*MULTIPROCESSING (COMPUTERS)/*PARTITIONS
(MATHEMATICS)/*PROBLEM SOLVING

MINS: / COMPUTATIONAL GRIDS/ MAPPING/ PARTIAL DIFFERENTIAL EQUATIONS/ TREES
(MATHEMATICS)

ABA: Author

A Partitioning Strategy for Non-Uniform Problems on Multiprocessors

*Marsha J. Berger**

Courant Institute of Mathematical Sciences
New York University
251 Mercer St.
New York, NY 10012

*Shahid H. Bokhari***

Institute for Computer Applications in Science and Engineering
and
University of Engineering and Technology
Lahore-31, Pakistan

ABSTRACT

We consider the partitioning of a problem on a domain with unequal work estimates in different subdomains in a way that balances the work load across multiple processors. Such a problem arises for example in solving partial differential equations using an adaptive method that places extra grid points in certain subregions of the domain. We use a binary decomposition of the domain to partition it into rectangles requiring equal computational effort. We then study the communication costs of mapping this partitioning onto different multiprocessors: a mesh-connected array, a tree machine and a hypercube. The communication cost expressions can be used to determine the optimal depth of the above partitioning.

*Supported in part by Department of Energy Contract No. DEAC0276ER03077-V, and by the National Aeronautics and Space Administration under NASA Contract No. NAS1-17070, while the author was in residence at the Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Hampton, VA 23665.

**Supported by NASA Contract No. NAS1-17070 while the author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center.

NS80-19947#1

1. Introduction

We consider the partitioning of a problem on a domain with unequal computational work estimates in different subdomains, in a way that balances the work load across multiple processors. Such a problem arises, for example, in solving hyperbolic partial differential equations using an adaptive method that places extra grid points in certain subregions of the domain (see e.g. [5] and [6]). Such an approach has also been proposed in the multigrid literature ([2],[3],[12],[15]). At a given instant of time a typical computational mesh for either of these problems might look like Fig. 1.1.

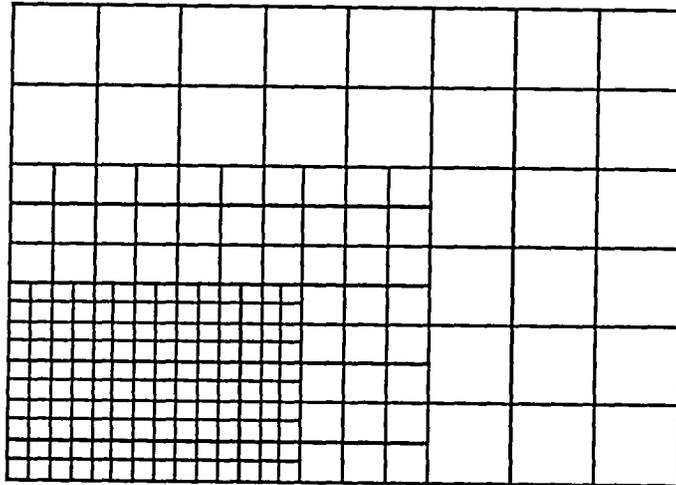


Fig. 1.1 Increased resolutions is obtained by superimposing fine grid patches over an underlying global coarse grid.

Hence, any simple partitioning scheme must account for the unequal amount of work to be done in the left half versus the right half of the domain. If Fig. 1.1 represents the grid for a time dependent problem, the work load in the left and right halves is significantly different, since the finest grids in space typically need a smaller step in time (if an explicit finite difference scheme is used to integrate the solution), and so many time steps are taken on the fine grid for every one on any coarser grid.

Other numerical examples which give rise to a problem with unequal work estimates might come from solving a pde with different equation sets in different regions. For example, in calculating transonic flow around airfoils, the Navier Stokes equations may be used in a boundary layer around the airfoil, and the Euler equations or even the potential equations can be used in the farfield. These different sets of equations have very different costs associated with their corresponding difference schemes. Another possibility is that the work estimates come from different physics in different parts of the domain, for example in weather

calculations, depending on whether a region is over water or over land. Other examples of unequal work include an iteration scheme such as SOR where a certain percentage of the domain is relaxed a second time before iterating on the entire domain again. This ad hoc procedure, applied to say the 10% of the grid with the largest residual can greatly reduce the cost of convergence.

In addition to the above *static* domains, we are interested in investigating the possibilities of solving adaptive mesh refinement problems on a multiprocessor system. A major difficulty is that, no matter how portions of the mesh are initially assigned to processors, a change in the mesh refinement will ultimately cause the computational load on the processors to become unbalanced. Attempts at rebalancing are complicated by the need to keep the interprocessor communication overhead at a minimum. Since the adaptive mesh refinement strategy in [6] is already based on a partitioning of the domain into rectangular grid patches, we can derive an approach presented here which is simple and tractable. Other approaches are given in [9], [17].

Most partitioning strategies use some type of domain decomposition to balance the work load over many processors. Typically, these uniform mesh problems can be divided into boxes (Fig. 1.2a) or strips (Fig. 1.2b).

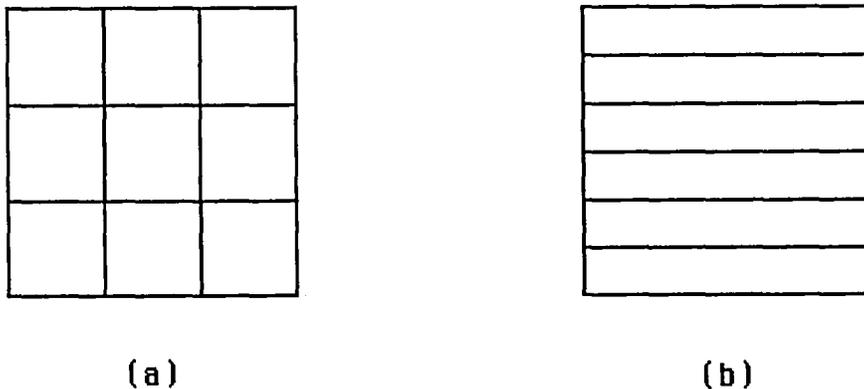


Fig. 1.2 Two common partitioning strategies for rectangular mesh problems.

The benefits of one over the other depend on the cost of transferring information around the perimeter of a box to the neighboring partition/processor (which depends on the machine architecture), and the order in which computations on such configurations can proceed. Papadimitriou and Ullman [13] discuss communication/time tradeoffs for such partitionings, and obtain lower bounds on the rate such computations can proceed.

A different kind of partitioning is evident in the work of Adams and Jordan [1]. By partitioning down to the grid point level, using a multi-color SOR iteration scheme, simultaneous updates can proceed for any given color grid point throughout the entire mesh. This can be useful on processor arrays as well as vector computers.

This paper is organized as follows. We describe the binary decomposition used to partition the work load in section 2, and discuss some of its properties in section 3. In sections 4, 5 and 6, we study the communication costs of mapping this partitioning onto different types of multiprocessors: a nearest neighbor array, a tree machine, and a hypercube. We derive expressions for the communication versus computation costs which can be used to determine an optimal depth for the above partitioning. Section 7 summarizes the results.

2. Binary Decomposition of the Domain

In this section we describe the strategy used to partition a domain into subunits requiring equal computational effort. In this presentation we will assume that the number of available processors is a power of 2, although many of our results generalize. Another underlying assumption is that the number of grid points $N \gg p$, the number of processors. We will concentrate on the static case, and only say a few words about adaptively rebalancing the decomposition later in the section.

Suppose that work estimates on a given domain have already been obtained, through a priori knowledge, or from an initial computation on a uniform mesh using a partitioning as in Fig. 1.2. Given these work estimates, we can now make a vertical cut through the domain so that the left and right segments each contain half the work (or as near as possible given the constraint that the line is vertical, and the number of grid points in each segment increases by a finite amount on shifting the location of the cut by one column). If there are four processors available, the two segments are each partitioned using two different line segments of a horizontal cut line next, into a total of four equally balanced work loads. This procedure continues by recursively partitioning using first vertical then horizontal cut line segments, so that the length of the longest side of any subregion is reduced every other step. A typical decomposition for the grids in Fig. 1.1 using 16 processors is shown schematically in Fig. 2.1. The idea for this decomposition was inspired by the similar looking rectangular regions used by Bentley [4] in answering two dimensional point domination questions.

We emphasize that the computational work of an iteration on any rectangular region in Fig. 2.1 is identical. However, the communication requirements across the perimeters are not. In particular, if the source of the problem is a grid such as Fig. 1.1, then the grid point density along any given line segment varies, depending on whether a cut line intersects a finer grid or not. In general, therefore, we will only obtain upper bounds for communication

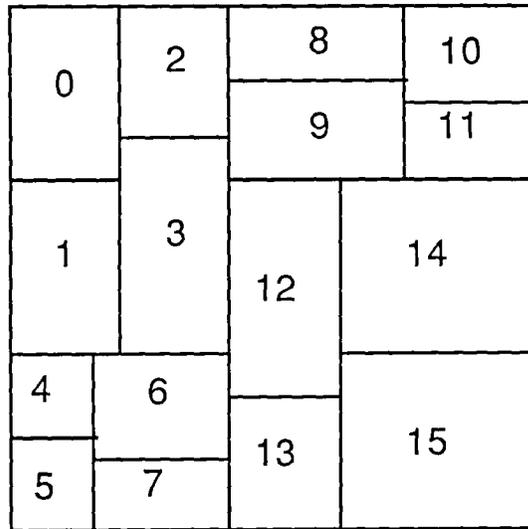


Fig. 2.1 A binary decomposition using 16 processors.

costs assuming the worst case grid point distribution. Instead, therefore, we will now assume that the problem gives rise to different work requirements in different regions, but is based on an underlying grid of N^2 points which are uniformly distributed.

There are several points to note about the decomposition depicted in Fig. 2.1. First, by restricting the subunits to be rectangular blocks, we avoid a messy problem with data structures. If more general L-shaped regions or diagonal lines were used, the specification of a region would be more difficult. All that this approach requires to specify each block is the four corners of the rectangle (2 will do). This is sufficiently low overhead that every processor can keep a map of the entire domain with each processor/rectangle pair. A tree data structure can easily be traversed for any neighbor information that is needed, for example, for a non-shared memory machine.

Secondly, this approach does not suffer the drawbacks that other decompositions suggested for this problem have. For example, Fig. 2.2 indicates a grid configuration with four fine grids superimposed on a global coarse grid, and 2 further refined grids nested in 2 of the four. It is tempting to use these grids, which already form one type of decomposition of the domain (and are each regular with a simple data structure) as the basis for assignment of work to a processor. However, there is no attempt at load balancing in this approach. In addition, if the mesh is later changed so that there are 8 subgrids, for example, either the computation must request 8 processors, or some of them were idle beforehand.

We mention that the partitioning itself is easily accomplished. In principle, the partitioning can be done by summing the numbers of grid points (or work per grid point times the number of grid points) first rowwise, then columnwise. The partitioning cut is then made in

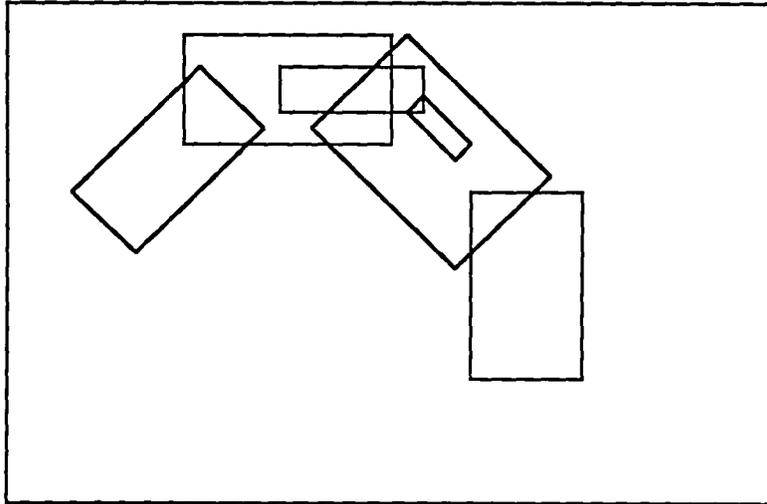


Fig. 2.2 A domain decomposition with 4 subgrids, and 2 sub-subgrids.

the middle.

We point out one final advantage of the binary decomposition. As the computation proceeds, if it turns out that one region gets more work (say a finer mesh is introduced), a local rebalancing can be done without necessarily redoing the entire partitioning. For example, in Fig. 2.1 if region 14's work load increases by some amount δ , the last cut line in the partition, which separates regions 14 and 15, can be adjusted so that both regions have an imbalance of only $\delta/2$. If the k previous cut lines are adjusted, the imbalance is reduced to $\delta/2^k$. The cost of rebalancing (which might include time to send data to the new governing partition, for example), can be traded off against the lost time of having an imbalance in the work load. This can determine how high in the tree (the number k above) to rebalance.

3. Analysis of Partitionings

In this section we present several definitions related to our partitionings and analyze some of their important properties. These are essential to our discussion of mappings of subregions onto various processor architectures, which we present in subsequent sections.

3.1. Definitions

The *depth* of a partitioning is the number of times the domain has been partitioned. This equals the depth of the corresponding binary tree, with root node corresponding to the entire domain, and leaf nodes corresponding to each rectangle in the final partitioning.

Each partition line is divided into a number of *segments* by the incidence of other partition lines.

The *total number of segments* of a partitioning is the sum of all such segments.

The *depth* of a segment is the depth of the partition line to which it belongs.

For example, Fig. 3.1 shows a partitioning of depth 2. The depth 1 partition line a-b is divided into 3 segments by the incidence of the two depth 2 partition lines c-d and e-f. The total number of segments in this partitioning are 5.

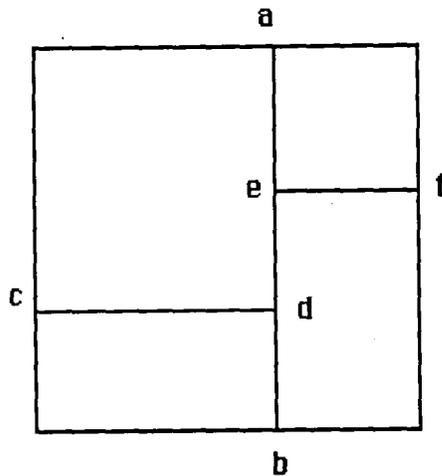


Fig. 3.1 A depth 2 partitioning with 5 segments.

We observe that if two adjacent regions of a partitioning are assigned to different processors then the segment between them represents a communication requirement between the processors. The following definition makes this easier to appreciate.

The *graph of a partitioning* is the dual graph obtained in the usual way [8] by representing each region by a node and connecting two nodes if and only if the corresponding regions are adjacent (share a segment on their perimeter). Each edge in the graph of a partitioning represents a communication requirement between the two regions represented by the nodes at its end points. Fig. 3.2 shows a partitioning of depth 4 along with its graph. The assignment of regions of a partitioning to the processors of a multiple computer system is now equivalent to the mapping of the graph of a partitioning onto the graph of a multiple computer system [7].

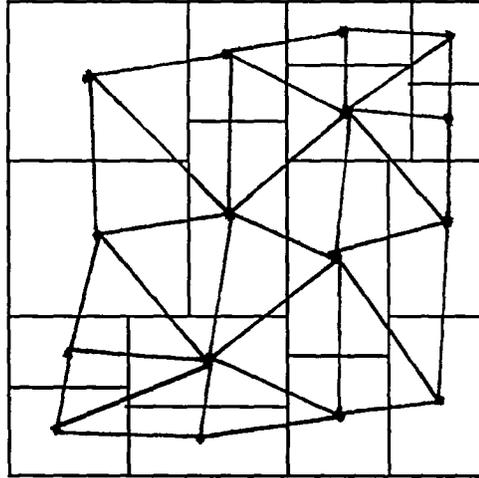


Fig. 3.2 A depth 4 partitioning and its dual graph.

3.2. Properties

Clearly, the total number of segments of a partitioning is an important number in our analysis, since it represents the total number of different communication paths required. A lower bound on this number occurs when the domain has uniform computational density in which case the partitioning is made up of continuous horizontal and vertical lines extending from one side of the region to the other. It is easy to show that in this case the total number of segments T_L is

$$T_L(k) = 2^{k+1} - 2^{\frac{k}{2}+1}, \quad k \text{ even} \quad (3.1)$$

$$T_L(k) = 2^{k+1} - 3 \cdot 2^{\frac{k-1}{2}}, \quad k \text{ odd},$$

for a depth k partitioning for 2^k processors.

To obtain an upper bound $T_U(k)$ on the number of segments, we need to investigate further properties of partitionings.

3.2.1. Property 1

Note that the graph of a partitioning remains unchanged no matter how much the vertical (horizontal) partitioning lines are displaced as long as the sequence of the vertical (horizontal) coordinates of these lines are not disturbed. Because each edge of this graph corresponds to a segment, this property implies that the total number of segments is unchanged under such displacements.

3.2.2. Property 2

A partitioning of depth k may be considered to be made up of the juxtaposition of two partitionings of depth $k-1$. The line at which the constituent depth $k-1$ partitionings abut each other becomes the new depth 1 line, and the depth of the remaining lines increases by one. If k is even, the constituent depth $k-1$ partitionings will have $2^{\frac{k}{2}}$ by $2^{\frac{k}{2}-1}$ sides; in this case the sides with the larger number of regions will face each other. If k is odd, the constituent regions have $2^{\frac{k-1}{2}}$ regions along either side.

Because of Property 1 above, we can displace the segments perpendicular to the interface of the constituent regions to any extent as long as we do not disturb their sequence. This allows us to distort the constituent partitions so that no two depth $k-1$ segments on either side of the interface are collinear, and thus obtain the maximum number of depth 1 segments, which turns out to be

$$S_{\max}(k) = 2^{\frac{k}{2}+1} - 1, \quad k \text{ even} \quad (3.2)$$

$$S_{\max}(k) = 2^{\frac{k+1}{2}} - 1, \quad k \text{ odd.}$$

It is possible for a partitioning of depth k to have the maximum possible number of depth 1 segments, independent of its constituent depth $k-1$ partitionings.

3.2.3. Property 3

It thus follows that the depth k partitioning with maximum total segments is made up of two depth $k-1$ partitionings with maximum total segments. This leads to the following recurrence for maximum total number of segments,

$$T_U(k) = 2^{\frac{k}{2}+1} - 1 + 2 T_U(k-1), \quad k \text{ even} \quad (3.3)$$

$$T_U(k) = 2^{\frac{k+1}{2}} - 1 + 2 T_U(k-1), \quad k \text{ odd}$$

The solutions to these recurrences are

$$T_U(k) = 2^{k+2} - 2^k - 2^{\frac{k}{2}+2} + 1, \quad k \text{ even} \quad (3.4)$$

$$T_U(k) = 2^{k+2} - 2^k - 3 \cdot 2^{\frac{k+1}{2}} + 1, \quad k \text{ odd}$$

3.2.4. Property 4

Another property of great interest is the maximum degree of any node in the graph of a partitioning. A simple counting argument shows that for 2^k processors, $k \geq 4$, the maximum degree is

$$2^{\frac{k}{2}} + 2^{\frac{k}{2}-1} + 3, \quad k \text{ even} \tag{3.5}$$

$$2^{\frac{k+1}{2}} + 3, \quad k \text{ odd.}$$

The node with this maximum degree borders on the depth 1 partitioning line (see Fig. 3.3).

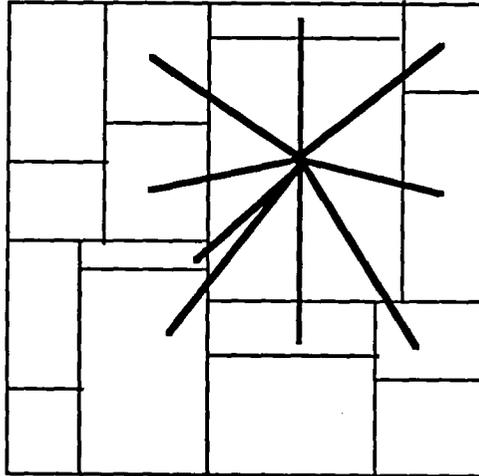


Fig. 3.3 Maximum degree possible for a 2^k processor decomposition, $k=4$.

In fact, a simple counting procedure shows the following. For 2^k processors, k even, the partition consists of $k/2$ segments of vertical cut lines interlaced with $k/2$ horizontal cut line segments. If we number the cut lines from $j = 1$ to k in total, then the number of incident edges a segment from a cut line of depth j can have is at most

$$e_k(j) = 2^{\frac{k}{2} + 1} - \left\lfloor \frac{j}{2} \right\rfloor - 1, \quad k \text{ even} \tag{3.6}$$

$$e_k(j) = 2^{\frac{k+1}{2} + 1} - \left\lfloor \frac{j+1}{2} \right\rfloor - 1, \quad k \text{ odd.}$$

Using the fact that 2^{j-1} line segments make up the j^{th} cut line, the expressions in (3.6) can be summed to provide an alternate derivation of (3.4) for the maximum number of edges in the

dual graph.

4. Mapping onto Nearest Neighbor Arrays.

In this section we investigate how the binary partitionings can be mapped onto nearest neighbor arrays. By nearest neighbor arrays we mean multicomputer systems in which the processors may be thought of as points on an integer lattice, where each processor has communication links to its 4, 8 or more nearest neighbors. For example, the Illiac machine (if we ignore its wraparound connections) is an example of a 4 nearest neighbor array (4nn array). The FEM [11] is an example of an 8nn array.

In this analysis we will consider rectangular arrays of size 2^k . For such arrays, there exists what we call *natural* mappings of depth k partitioned regions onto processors, defined as follows. When partitioning the domain into two regions, partition the processor array into two equal halves. Assign the left subdomain to the left half and the right subdomain to the right half. Repeat recursively until the processor partitions have exactly one processor in them. At this point every partitioned region has been assigned to a processor.

4.1. Cardinality of Natural Mappings

One way to measure the quality of a mapping is to compute its *cardinality* [7], defined as the number of edges of the problem graph that fall on edges of the processor graph divided by the total number of edges in the problem graph. Mappings with a cardinality of one have minimum interprocessor communication overhead since all processes that need to communicate lie on processors that are adjacent to each other. One such extreme case occurs when the domain being partitioned is uniform (as described in Section 3.2), and the graph of the partitioning matches the graph of a 4nn array perfectly.

At the other extreme, decompositions of the type illustrated in Fig. 3.3 have graphs in which some nodes have exponential degree. We could not possibly accommodate the edges incident on such nodes using any fixed degree nearest neighbor array. The question then is how low the cardinality can be over all possible decompositions.

A simple but important observation towards this goal is the following. In a natural mapping of the graph of a partitioning onto a 4nn array, the edges on the perimeter of any subdomain are all mapped on the corresponding edges of the mesh. This can be seen in Fig. 2.1, where the 12 edges on the perimeter of the complete domain fall on the perimeter edges

of the array. At the same time, the 8 perimeter edges in the left and right hand subdomains also fall on perimeter edges of the two halves of the array and so on.

It follows that the edges of the partitioning graph that *fail* to fall on edges of the array graph cannot exceed the number of edges that extend across partitions and are not perimeter edges. The number of such 'misses' that extend across the depth 1 partition is precisely the number of depth 1 segments (eq. (3.2)) less 2 (the perimeter edges), giving the following recurrences,

$$M(k) = 2^{\frac{k+1}{2}} - 3 + 2M(k-1), \quad k \text{ even} \quad (4.1)$$

$$M(k) = 2^{\frac{k+1}{2}} - 3 + 2M(k-1), \quad k \text{ odd}, \quad k > 1.$$

It is important to appreciate that for $k=1$, $M(k)=0$.

Restricting to the case of k even, these recurrences can be solved to yield

$$M(k) = 3 \cdot 2^{k-1} - 2^{\frac{k+2}{2}} + 3. \quad (4.2)$$

Combining (4.2) with the expression for the total number of edges yields the cardinality

$$\begin{aligned} C(k) &= \frac{T_U(k) - M(k)}{T_U(k)} \\ &= \frac{3 \cdot 2^{k-1} - 2}{3 \cdot 2^k - 2^{\frac{k+2}{2}} + 1} \end{aligned} \quad (4.3)$$

with a similar expression for k odd. Clearly, when $k=1$, $C(k)=1$, and it drops gradually as k increases. It can be seen, however, that as k becomes large the cardinality converges to 0.5 and does not drop further. For example, for $k=4$, $C(k)=.67$, but for $k=10$, $C(k)=.52$. A similar analysis for the $8nn$ array gives the recurrence relation $M(k) = 2^{\frac{k+1}{2}} - 5 + 2M(k-1)$, which reveals that $C(k)$ converges to .79.

One question to consider is whether these natural mappings are near optimal, or whether there exists other mappings of regions to processors with higher cardinality. A worst-case partitioning can have T_U edges, where T_U is given by eq. (3.4). A $4nn$ array has only T_L edges, where T_L is given by eq. (3.1). An optimal mapping is one which uses all $4nn$ edges, and thus has cardinality of at most $\frac{T_U - T_L}{T_U} = 75\%$. In the worst case, there are

natural mappings that use all $4nn$ edges, and have the maximum number of additional edges for a total of T_U , and so by this measure, natural mappings are within $2/3$ of optimal. We conjecture that in fact, no other mapping can do better for a $4nn$ array. A similar analysis cannot be applied to $8nn$ arrays, since even in the worst case, the number of edges in these arrays exceeds the number of edges in the partitioning graph. An $8nn$ array is not well utilized by our partitionings, since it has so many unused edges.

4.2. Communication Requirements

The cardinality expressions of the previous section are interesting, but do not give precise expressions for the communication overhead. In this section, we obtain upper bounds for the total cost of communication when running our dissections on $4nn$ meshes. We assume that the solution method is as follows. All processes compute in parallel and, by construction, take the same amount of time. The communication step proceeds as follows. First the information to be communicated across vertical boundary segments is transmitted (first left then right) by each processor. Then this step is repeated for horizontal segments (top then bottom). Inspection of Fig. 3.2 reveals that when the dual graph of a binary dissection is naturally mapped onto a $4nn$ mesh, adjacent nodes of the dual graph are mapped onto nodes of the mesh that lie at least in adjacent rows or adjacent columns. This means that each communication step above has two phases. In the first phase, data makes at most one horizontal (vertical) movement to get to the correct column (row). In the second phase, it travels zero or more steps in the vertical (horizontal) direction to get to the correct row (column). The hardware communication mechanism at each node is assumed to be such that a unidirectional transmission of data can be performed on one communication link in one time step. Thus a two way exchange of data over a single communication line takes two time steps.

By definition all processors have equal amounts of computation. It remains to evaluate how much communication overhead is incurred. Notice that in the best case of a simple uniform partition, the length of the side of any square subdomain is

$$L_{uniform}(k,N) = \frac{N}{2^{k/2}},$$

for a total communication time

$$T_{uniform}(k,N) = 8 \cdot \frac{N}{2^{k/2}}. \tag{4.4}$$

for a problem with N points on a side mapped on to a square mesh with $2^{\frac{k}{2}}$ processors on a

side. In the case of non-uniform regions, the degree of distortion of a partitioning determines the time required during the second phase of a communication step. This time is slight for mildly distorted partitions but can be a major factor in partitions with large distortion.

4.2.1. Skewness of a partitioning

To quantify the amount of distortion in a partitioning, we introduce the following concept.

The *x-skewness* (S_x) of a given partitioning with $N \times N$ points and depth k is the ratio of the length of the longest horizontal side of any subdomain in that partitioning to the length of the side of the square in the corresponding uniform partitioning.

The *y-skewness* (S_y) is similarly defined for vertical sides.

For example in Fig. 3.3, S_x is about 2.5 and S_y , about 4.

We work under the assumption that there is always at least one point per processor.

This constrains the skewness to lie between 1 and $2^{\frac{k}{2}}(1 - \frac{(2^{\frac{k}{2}} - 1)}{N})$.

4.2.2. Dilation of edges in a dual graph.

Skewness itself does not completely determine the communication overhead. A partitioning can be highly skewed yet have a dual graph that precisely matches a $4nn$ array. On the other hand, a partitioning can have this same skewness, but with a large mismatch between the graph of the partitioning and a $4nn$ mesh.

To more accurately describe this mismatch we define the *dilation* of an edge in a dual graph that has been naturally mapped onto a $4nn$ mesh to be the number of edges that data passes through between two communicating processors *during the second phase* of communications.

The *x-dilation*, d_x (*y-dilation* d_y) of a partitioning is the maximum dilation over all edges in the x (y) direction.

In uniform partitionings (with $S_x=S_y=1$), the maximum dilation is zero. As skewness increases, the maximum possible dilation also increases. The biggest change occurs as the

skewness increases from 1 to 2, since the maximum y -dilation increases from 0 to $\frac{2^{\frac{k}{2}}}{2}$. The

general expression for the worst case dilation is

$$d_y = 2^{k/2} - \left\lfloor \frac{2^{k/2}}{S_y} \right\rfloor.$$

The maximum possible y-dilation in a partitioning of depth k is $2^{\frac{k}{2}} - 1$. Here we assume that the first cut we make in our partitioning is always a vertical line. However it is still possible for a partitioning with very high skewness to have zero dilation.

4.2.3. Data Transmitted Per Step

The x and y dilations and skewnesses allow us to compute the time required for communication. Each data point can be transmitted to its destination row or column in one time step in phase 1 of the communication. Assume the hardware first takes care of all data that is to move in the x direction and then all data to be moved in the y direction. For a non-uniform partitioning, the maximum number of data points transmitted per communication step is $2S_x L_{uniform}(k, N)$ in the x direction and $2S_y L_{uniform}(k, N)$ in the y direction, since each region has two vertical and two horizontal sides. The time required for phase one is then no more than

$$T_1(k, N) = 2(S_x + S_y) \frac{N}{2^{k/2}}.$$

Phase 2 of communication is complicated by the fact that there may be several overlapping communication paths in a single row or column. For example if the first processor in a row is transmitting to the 4th, then the 2nd might be transmitting to the 5th at the same time, causing congestion. Furthermore, the total time for communication is influenced by the x and y -dilations.

4.2.4. Communication Strategies

We propose two communication strategies for this troublesome phase two of communication. These are the *permutation* strategy and the *pipelined* strategy. Both strategies are useful over the range of values of problem size N , depth of partitioning k and skewness S .

4.2.5. The permutation strategy.

We may view phase 2 of communication as the permutation of a set of data on a chain of processors. Each processor sends a data value to a processor at most d_x (d_y) processors away in the x (y) directions. All processors can send *one* data value out to its destination in $2*d_x$ or $2*d_y$ time steps. The constant 2 arises because it takes 2 time units for a processor to receive and transmit one data point. The constant is not 4, which it would be for arbitrary permutations, because planarity insures no processor both transmits and receives in the same direction. The time for phase 2 of communications is thus

$$T_{perm} = 2 * (Max\ points\ per\ side) * (Max\ dilation).$$

for each side. This works out to be

$$T_{perm,x}(k,N) = 2 * S_x \frac{N}{2^{k/2}} * d_x$$

$$T_{perm,y}(k,N) = 2 * S_y \frac{N}{2^{k/2}} * d_y$$

The total time for phase 2 of communications is the sum of the two expressions multiplied by 2, since each region has two vertical and two horizontal sides,

$$T_{II,perm} = 4 * (S_x * d_x + S_y * d_y) \frac{N}{2^{k/2}}. \quad (4.5)$$

4.2.6. The pipelining strategy

Instead of viewing phase 2 as a sequence of permutations, we can think of it as a sequence of data transfers in which each processor transmits all of its data points to all processors to which it needs to transmit in a pipelined fashion. That is, if processor 3 needs to send data to processors 5, 6 and 7, it pipelines this transfer so that as soon as it finishes sending off data intended for processor 7, it starts sending data for processor 6 etc. In this case it is impossible for, say, processor 5 to send data to processor 8 (should it need to do so) until processor 3 has finished. This situation can arise from the partitioning in Fig. 4.1a, where the configuration gives rise to two separate, overlapping chains of communication. A *chain* is a contiguous sequence of processors in a column (row) that all receive data from a single processor in an adjacent column (row). Fig. 4.1b shows the chains from the column in Fig. 4.1a.

Thus there exists the problem of *congestion* which we define to be the number of overlapping chains of communication in a given row or column. As might be expected,

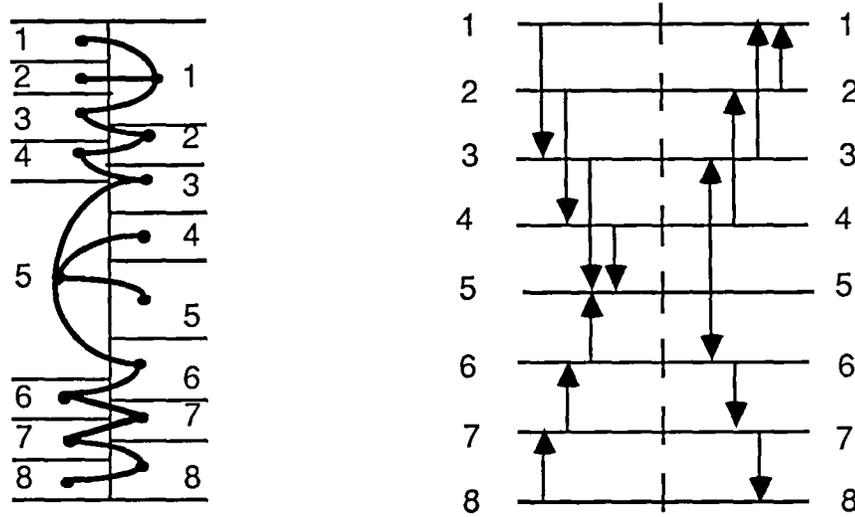


Fig. 4.1 Overlapping chains of communication cause congestion.

congestion varies with dilation. The form of this relationship is somewhat unexpected. As dilation increases from 1, the congestion increases but then reaches a maximum at half the maximum dilation and starts decreasing again. The exact expressions are

$$G_x(k) = \begin{cases} d_x & d_x \leq 2^{\frac{k}{2}-2} \\ 2^{\frac{k}{2}-1} - d_x & d_x > 2^{\frac{k}{2}-2} \end{cases} \quad (4.6)$$

$$G_y(k) = \begin{cases} d_y & d_y \leq 2^{\frac{k}{2}-1} \\ 2^{\frac{k}{2}} - d_y & d_y > 2^{\frac{k}{2}-1} \end{cases}$$

The amount of time required to complete phase 2 using the pipelined strategy is the maximum time for one region to send out its data multiplied by the maximum congestion.

The time required is thus

$$T_{pipe} = 2 * (\text{Max points/side} + \text{Max dilation}) * \text{congestion, i.e.}$$

$$T_{pipe,x}(k,N) = 2 * (S_x \frac{N}{2^{k/2}} + d_x) * G_x(k)$$

$$T_{pipe,y}(k,N) = 2 * (S_y \frac{N}{2^{k/2}} + d_y) * G_y(k)$$

The total time for phase 2 of communications is again the sum of the two expressions multiplied by 2, giving

$$T_{II,pipe} = 4(S_x \frac{N}{2^{k/2}} + d_x) * G_x(k) + 4(S_y \frac{N}{2^{k/2}} + d_y) * G_y(k). \quad (4.7)$$

Comparison of (4.5) and (4.7) shows that the permutation strategy is always preferable for partitions with low skewness, when $S \leq 2$. For $S > 2$ and low to moderate depth of partitioning, pipelining is better, since in this case the congestion G is small. Fig. 4.2 shows graphically the ratio of the respective costs of these communication strategies, for a problem where $N = 4096$ is the number of points on a side.

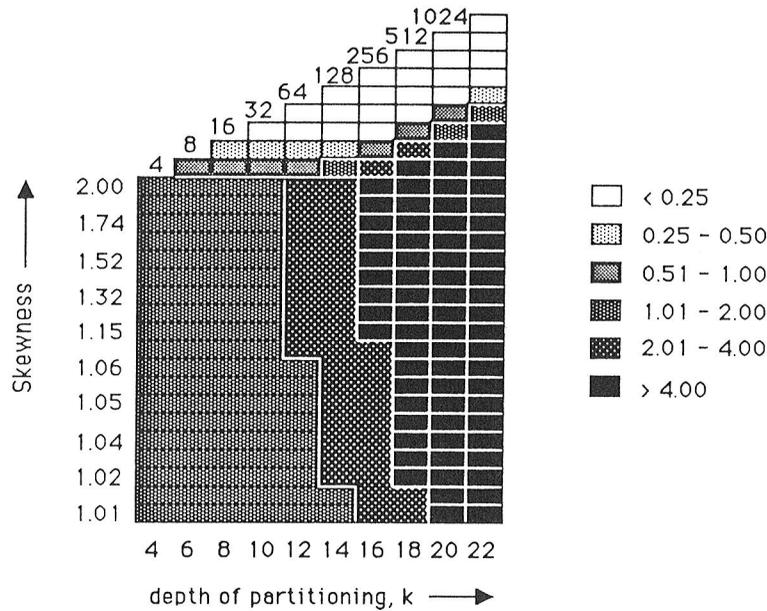


Fig. 4.2 Ratio of cost of pipeline to permutation strategy for communicating processors.

5. Mapping onto Trees

In this section we consider the costs of mapping the partitions onto binary trees. At first sight, tree structured multiprocessors would appear unsuitable for grid problems, because of potential traffic bottlenecks at the root. However, they are natural to consider in this case since we use a binary decomposition of the domain. Our results show that in the worst case, with 2^k processors, the performance of binary trees is within a factor of a constant times k of the mesh performance.

In our model, the leaf nodes do all the computation, and the rest of the nodes are used only for communication. Fig. 5.1 indicates how the partitions are matched with the leaf nodes. Regions that are separated by the last depth k partitioning cut are mapped to adjacent leaf nodes. Regions separated by the first partitioning cut are in different halves of the tree.

The solution algorithm starts with all leaf nodes computing on their respective sub-domains. The communication step can be thought of as having k phases. In the first phase, the leaf nodes send up all data that must pass through the root node. This includes all nodes that border the first, depth 1 partition cut, and takes time proportional to the length of that line. In the second phase, it sends data that rises no higher than the two children of the root node. This communication is between nodes sharing one of the two depth 2 boundary segments, and takes time proportional to the length of the depth 2 segment. In the k^{th} and last phase, leaf nodes sharing a depth k boundary segment swap data. If each phase proceeds to completion before the next phase starts, the communication time for phase j is proportional to the length of the maximum depth j segment. The total communication time is then proportional to the sum of these, and has latency k^2 through the tree. Instead, a leaf node can start the next phase of communication as soon as it is ready. This pipelining gives both a smaller communication time and a smaller latency through the tree of $2k-1$. We define a *hyperperimeter*, (in analogy with the perimeter estimates for the meshes), as $H(k) = \sum_{j=1}^k l_j$, where $l_j =$ any one line segment of depth j . Fig. 5.2 shows the maximum length hyperperimeter in the given partitioning. If the communication is pipelined, the hyperperimeter is a connected series of line segments. If instead, each communication phase proceeds to completion before the next phase starts, the maximum hyperperimeter need not be connected. Instead of the maximum of the sums, we would take the sum of the maximum length segments for each depth j .

To have a fair comparison with the performance of meshes, we assume that each non-leaf node can only receive or transmit in one direction over one link in one unit of time. Thus, a node can receive 3 values on its input lines and transmit 3 values on its output lines in 6 units of time. In the worst case therefore, buffers of size no greater than $2N$ are required. The total communication cost is then $6 \cdot \text{hyperperimeter} + \text{latency}$. For example, for a uniform domain decomposition the maximum hyperperimeter is

$$N + \frac{N}{2} + \frac{N}{2} + \dots + \frac{N}{2^{\frac{k}{2}-1}} + \frac{N}{2^{\frac{k}{2}-1}} + \frac{N}{2^{\frac{k}{2}}} = N \left[3 - \frac{3}{2^{\frac{k}{2}}} \right].$$

Thus the total time

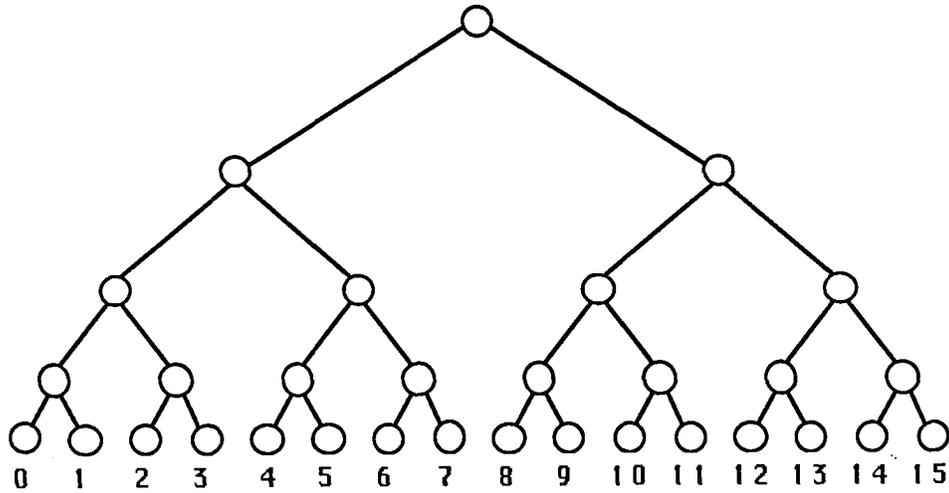


Fig. 5.1 A binary decomposition mapped onto a tree of processors.

$$T_{uniform}(k,N) = 18N (1-2^{-\frac{k}{2}}) + 2k-1. \tag{5.1}$$

Clearly, this is much worse than the corresponding expression (4.4) for a mesh.

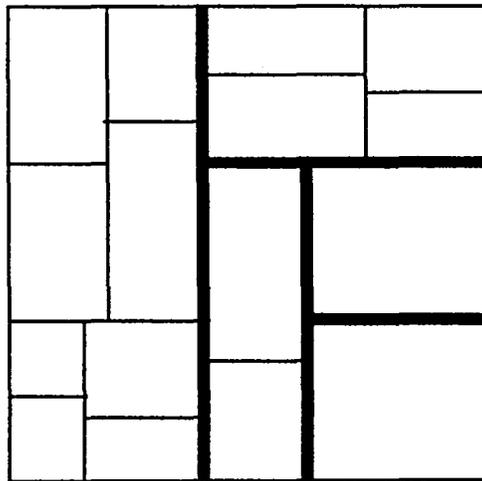


Fig. 5.2 Darkened hyperperimeter shows the maximum communication requirement in the tree.

For a non-uniform partition, in the worst case the hyperperimeter can grow like $N+(N-1)+(N-2) \dots$, except there must be at least 1 point per region. For $k = 4$, for example, the hyperperimeter is $N+(N-2)+(N-2)+(N-3)$. In general, the expression is $H(k) = kN - 2^{\frac{k}{2}}(k-3) - 3$ for a total time of

$$T(k,N) = 6kn - 3 \cdot 2^{\frac{k}{2}}(k-3) - 9 + (2k-1) \quad (5.2)$$

Notice that the latency is essentially irrelevant. In comparison with eq. (4.7) for nearest neighbor meshes, where the leading term is $4N$, the trees are a factor $\frac{3k}{2}$ worse. However, a naive analysis of trees gives a worst case bound of $6k^2N$, which is avoided here by using the extremely ordered properties of the binary decomposition.

These communication estimates can be used to determine the optimal number of partitions to use to solve a given sized problem, for a given efficiency. We do a sample calculation for the uniformly partitioned case. For an N by N square grid, the amount of computation to be done using 1 processor is $W_1 = C_1 \cdot N^2$. For a rather simple method, C_1 might include 20 floating point operations per point. For 2^k processors, the amount of work per processor is

$$W_{2^k} = C_1 \cdot N^2 / 2^k + C_2 \cdot 18N,$$

where we have dropped the lower order terms in the communication cost. The speedup is

$$S = \frac{W_1}{W_{2^k}}$$

and the efficiency is

$$E = \frac{S}{2^k} = \frac{1}{1 + \frac{C_2}{C_1} \cdot \frac{18 \cdot 2^k}{N}}$$

For example, if the communication time for one item C_2 takes approximately the same time as one floating point operation, then if $N = 10^2$, and $k = 6$, the efficiency is only 63%. For an efficiency of 87%, $k = 4$, or 16 processors should be used.

6. Communication Cost Analysis for Hypercubes.

An analysis similar to the one for nearest neighbor arrays can be performed for hypercubes. A nearest neighbor array of size $2^{\frac{k}{2}}$ by $2^{\frac{k}{2}}$ can be easily embedded in a hypercube of dimension k using the well known Gray code mapping method. Fig. 6.1 shows a 16 x 16 processor array that has been mapped onto a hypercube of dimension 8. All the edges of the mesh but only some edges of the hypercube have been shown. Specifically, only the edges of the hypercube that connect nodes that lie along one row and column of the 4nn mesh are shown. These edges demonstrate the richer interconnection of the hypercube in relation to the mesh. We would expect this richer interconnection to reduce the communication overhead when running our dissections.

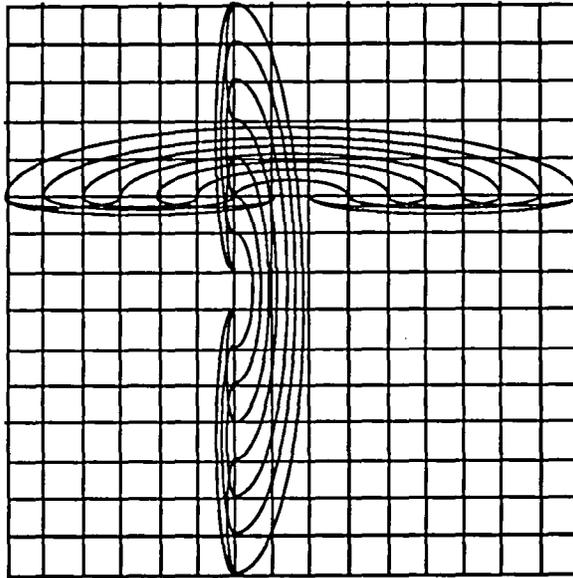


Fig. 6.1 A 16 by 16 mesh mapped onto a hypercube of dimension 8. Only some hypercube edges are shown.

6.1. Mapping onto Hypercubes.

To map the dual graph of a partitioning onto a hypercube, first map the dual graph onto a mesh, and then use the Gray codes to embed the mesh in the hypercube. The cardinality of the mapping in this case is no better than that obtained when a 4nn mesh is used. However, the richer interconnection of the hypercube leads to a smaller communication overhead when a detailed analysis is done.

6.2. Communication strategies.

As before, we assume a processor can send or receive only one item on one link at a time. We believe this is an accurate model of a scalable multiprocessor. A hypercube that can transmit on all links at one time is realizable only for a fixed dimension, but cannot be extended to higher dimensions. We again use two different communication strategies. In the case of the permutation strategy we take advantage of the logarithmic time to permute data in a hypercube. For the pipelining strategy we exploit the logarithmic diameter of the hypercube.

6.2.1. Permutation Strategy.

A hypercube of dimension k can perform any permutation of 2^k elements, one per processor, in $4k-1$ time, under our communication assumptions [16]. We can do better than this for our problem by noting that the edges connecting all the nodes in a single row or column of a mesh embedded in a hypercube (see Fig. 6.1) form a sub-hypercube of dimension $k/2$. Recall that data flows only along columns or rows during phase 2 of the communication step. Thus each permutation step takes at most $2k-1$ time. This is a worst case analysis, however, and may not be optimal, since it does not depend on the dilation of a given partitioning. We obtain the following expressions,

$$T_{perm,x}(k,N) = S_x \frac{N}{2^{k/2}} * (2k - 1)$$

$$T_{perm,y}(k,N) = S_y \frac{N}{2^{k/2}} * (2k - 1)$$

for a total phase II time of

$$T_{II,perm} = 2 * (2k-1) * (S_x + S_y) \frac{N}{2^{k/2}}. \quad (6.1)$$

These correspond to (4.5) for meshes.

6.2.2. Pipelining Strategy

In the case of hypercubes, the pipelining strategy utilizes the logarithmic diameter of hypercubes to improve communication times. If we examine a contiguous subchain of d_x nodes in a row or column, it is easy to verify that there exists a tree of diameter no more than $1 + \lceil \log_2(d_x) \rceil$ rooted at every node, that reaches every other node in the chain.. This tree may be found by doing a breadth first search outwards from the desired node, staying within the graph induced by the contiguous subchain. Thus the requirement of the pipelining strategy that one node transmits to several other nodes in a subchain takes no more than $2 + \lceil 2\log(d_x) \rceil$ time. The problem of congestion remains. The amount of congestion is precisely the same as with 4nn meshes. This is because the problem and not the multicomputer architecture determines the congestion. In the hypercube, a number of trees are pumping data out towards their leaves in parallel, instead of a number of chains linearly pumping data out towards their end points, as was the case for meshes. The time required is

$$T_{pipe,x}(k,N) = (S_x \frac{N}{2^{k/2}} + 2\log_2(d_x) + 2) * G_x(k)$$

$$T_{pipe,y}(k,N) = (S_y \frac{N}{2^{k/2}} + 2\log_2(d_y) + 2) * G_y(k)$$

for a total time of

$$T_{\Pi pipe} = 2 \cdot (S_x \frac{N}{2^{k/2}} + 2 \cdot \log(d_x) + 2) G_x + 2 \cdot (S_y \frac{N}{2^{k/2}} + 2 \cdot \log(d_y) + 2) G_y. \quad (6.2)$$

Fig. 6.2 shows the ratio of the pipelining to permutation communication costs in a hypercube. For any fixed skewness, the pipelining strategy is initially cheaper, but as with meshes, the permutation strategy eventually wins with increasing depth of partitioning.

We compare the communication costs in a nearest neighbor array versus a hypercube in Fig. 6.3. For each type of machine, the cost used is the cheaper of the permutation and pipelining costs, for the given parameters. For a large range of skewness and low depth of partitioning, the mesh is not much worse than a hypercube. For high depth of partitioning and moderate skewness, the hypercube is much better, but its performance approaches that of a mesh for very low and very high skewness. For large skewness, there is a region with a large perimeter which takes a long time to transmit. The cost of this dominates both the square root and logarithmic communication latencies of the mesh and hypercube respectively.

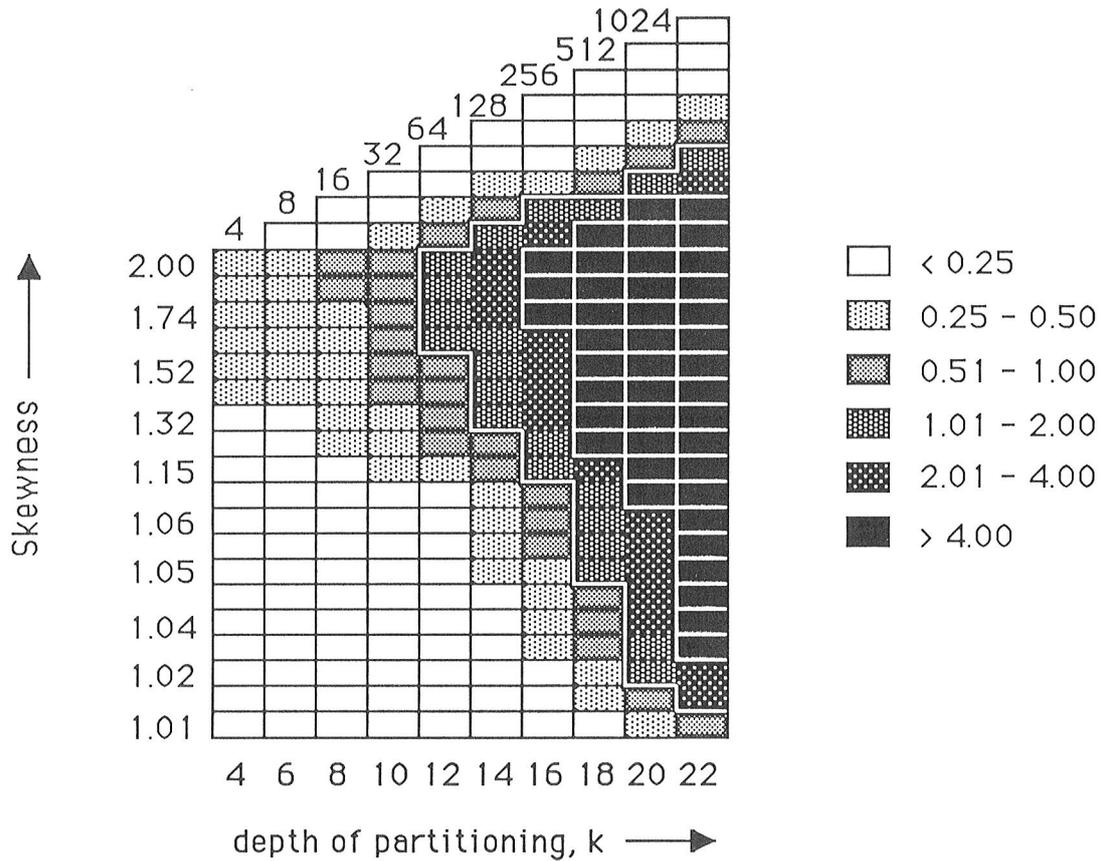


Fig. 6.2 Ratio of cost of pipeline to permutation strategy for a hypercube.

7. Conclusions

We have shown how domains with non-uniform workloads can be partitioned to equidistribute the computational load. For these types of grid problems, our binary decompositions can be mapped in a natural way onto trees, nearest neighbor meshes and hypercubes.

For $4n$ arrays, it is interesting that the map from the problem graph onto the array graph has at least a 50% hit rate of edges. For an $8n$ array this is 79%.

We further evaluated the performance by analyzing the traffic through these networks. For an N by N problem using 2^k processors in the worst cases, the communication overhead using the pipelined strategy, was approximately $4N$ for hypercubes, $8N$ for meshes, and $6kN$ for trees. The performance of trees was found to be better than a naive analysis would suggest. While these results are encouraging, certainly a better approach is to partition using a weighted sum of computational effort and communication costs. In addition, the more difficult problem of adaptive load balancing will have to confront the problems of modifying data

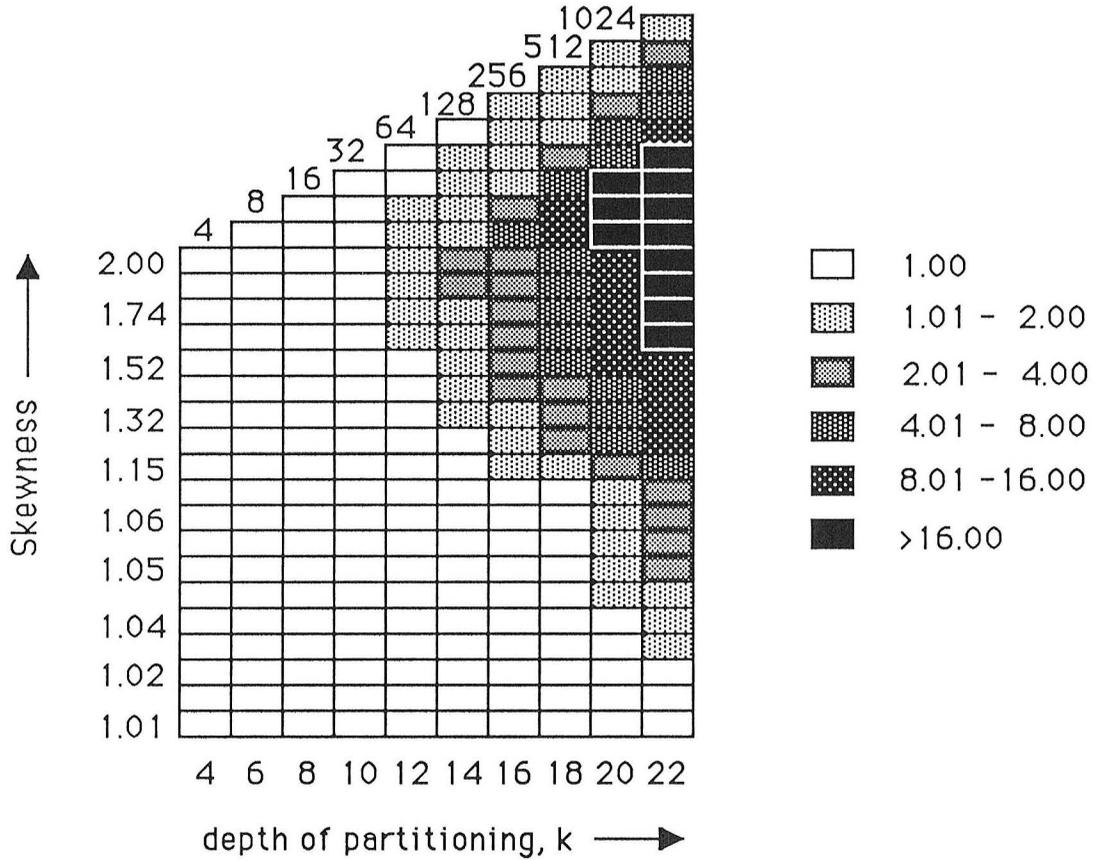


Fig. 6.3 Ratio of the communication cost of a nearest neighbor array and a hypercube.

structures within each processor. It will clearly be more efficient to tolerate small amounts of load imbalance than to change partitions with every perturbation.

We believe the decomposition technique presented here would be beneficial even on shared memory machines such as the Ultracomputer [10] or the IBM RP3 [14]. Our partitionings allow efficient load balancing across processors, without the overhead of a fine-grained queueing mechanism that would otherwise be necessary. They would also reduce memory traffic and increase the cache hit rate.

Acknowledgements

It is a pleasure to acknowledge several helpful discussions with Bob Voigt, and to thank Vijay Naik for pointing out reference [16].

References

- [1] L. Adams and H. Jordan, "Is SOR Color-Blind?", ICASE Report No. 84-14, May, 1984.
- [2] D. Bai and A. Brandt, "Local Mesh Refinement Multilevel Techniques", Weizmann Institute of Science Report, 1984.
- [3] R. Bank and A. Sherman, "Algorithmic Aspects of the Multi-level Solution of Finite Element Equations", CNA-144, Center for Numerical Analysis, University of Texas at Austin, October 1978.
- [4] J. Bentley, "Multidimensional Divide-and-Conquer", *Comm. ACM* 23, (1980).
- [5] M. Berger and A. Jameson, "Automatic Adaptive Mesh Refinement for the Euler Equations", *AIAA Journal* 23, (1985).
- [6] M. Berger and J. Olinger, "Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations", *J. Comp. Phys.* 53, (1984).
- [7] S. Bokhari, "On the Mapping Problem", *IEEE Trans. Computers* C-30,3 (1981).
- [8] N. Deo, *Graph Theory with Applications to Engineering and Computer Science*, Prentice-Hall, 1974.
- [9] D. Gannon and J. Van Rosendale, "Parallel Architectures for Iterative Methods on Adaptive Block Structured Grids", in *Elliptic Problem Solvers II*, G. Birkhoff and A. Schoenstadt, editors, Academic Press. 1984.
- [10] A. Gottlieb, R. Grishman, C. Kruskal, K. McAuliffe, L. Rudolph and M. Snir, "The NYU Ultracomputer - Designing an MIMD Shared Memory Parallel Machine", *IEEE Trans. Computers* C-32,2 (1983).

- [11] H. Jordan, "A Special Purpose Architecture for Finite Element Analysis", in *Proc. 1978 Conf. on Parallel Processing*, Aug. 1978.
- [12] S. McCormick and J. Thomas, "The Fast Adaptive Composite Grid Method for Elliptic Equations". To appear in *Math. Comp.*, 1986.
- [13] C. Papadimitriou and J. Ullman, "A Communication-Time Tradeoff", in *IEEE Proc. 25th Annual Symp. on Foundations of Computer Science*, (1984).
- [14] G. Pfister, et al, "The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture", *Proc. 1985 Intl. Conf. Parallel Proc.*, (1985).
- [15] J. Van Rosendale, "Rapid Solution of Finite Element Equations on Locally Refined Grids by Multi-level Methods", Ph.D. Thesis, University of Illinois UIUC, May 1980.
- [16] A. Waksman, "A Permutation Network", *J. ACM* 15,1, (1968).
- [17] P. Zave and W. Rheinboldt, "Design of an Adaptive, Parallel Finite-Element System", *ACM Trans. Math. Software* 5, (1979).

1. Report No. NASA CR-178024 ICASE Report No. 85-55		2. Government Accession No.		3. Recipient's Catalog No.	
4. Title and Subtitle A PARTITIONING STRATEGY FOR NON-UNIFORM PROBLEMS ON MULTIPROCESSORS				5. Report Date November 1985	
				6. Performing Organization Code	
7. Author(s) Marsha J. Berger and Shahid H. Bokhari				8. Performing Organization Report No. 85-55	
				10. Work Unit No.	
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				11. Contract or Grant No. NAS1-17070	
				13. Type of Report and Period Covered Contractor Report	
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D.C. 20546				14. Sponsoring Agency Code 505-31-83-01	
				15. Supplementary Notes Langley Technical Monitor: Submitted to IEEE Trans. Comput. J. C. South Jr. Final Report	
16. Abstract We consider the partitioning of a problem on a domain with unequal work estimates in different subdomains in a way that balances the work load across multiple processors. Such a problem arises for example in solving partial differential equations using an adaptive method that places extra grid points in certain subregions of the domain. We use a binary decomposition of the domain to partition it into rectangles requiring equal computational effort. We then study the communication costs of mapping this partitioning onto different multiprocessors: a mesh-connected array, a tree machine and a hypercube. The communication cost expressions can be used to determine the optimal depth of the above partitioning.					
17. Key Words (Suggested by Author(s)) partitioning problem, multi-processors, load balancing, hypercubes, trees, meshes			18. Distribution Statement 59 - Mathematical & Computer Sciences (General) 62 - Computer Systems Unclassified - Unlimited		
19. Security Classif. (of this report) Unclassified		20. Security Classif. (of this page) Unclassified		21. No. of Pages 30	22. Price A03

