NASA Contractor Report 178282

ICASE REPORT NO. 87-7

ICASE

PROBLEM SIZE, PARALLEL ARCHITECTURE, AND OPTIMAL SPEEDUP

David M. Nicol

Frank H. Willard

Contract No. NAS1-18107

April 1987

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

Problem Size, Parallel Architecture, and Optimal Speedup

David M. Nicol and Frank H. Willard

Institute for Computer Applications in Science and Engineering
and
The College of William and Mary

ABSTRACT

The communication and synchronization overhead inherent in parallel processing can lead to situations where adding processors to the solution method actually increases execution time. Problem type, problem size, and architecture type all affect the optimal number of processors to employ. In this paper we examine the numerical solution of an elliptic partial differential equation in order to study the relationship between problem size and architecture. The equation's domain is discretized into n^2 grid points which are divided into partitions and mapped onto the individual processor memories. We analytically quantify the relationships between grid size, stencil type, partitioning strategy, processor execution time, and communication network type. In doing so, we determine the optimal number of processors to assign to the solution (and hence the optimal speedup), and identify (1) the smallest grid size which fully benefits from using all available processors, (2) the leverage on performance given by increasing processor speed or communication network speed, (3) the suitability of various architectures for large numerical problems.

This research was supported by the National Aeronautics and Space Administration under NASA Contract Number NAS1-18107 while the author was in residence at ICASE, NASA Langley Research Center, Hampton, VA 23665.

1. Introduction

A numerical solution to an elliptic partial differential equation (PDE) is usually constructed by modeling the continuous domain of the equation's variables with a grid of discrete points. The partial derivatives are approximated using some differencing scheme, and a linear set of equations is constructed whose unknowns are the values of the solution function at each of the grid points. During an iterative solution of these equations (e.g. point Jacobi) the value at a point is approximated by a function of values at nearby points. The amount of computational work associated with updating an interior grid point is the same throughout the grid. Furthermore, during a single iteration grid points can be updated in parallel. This high degree of regularity and potential parallelism has made the solution of PDEs a very attractive problem area for the application of parallel processing.

An elliptic PDE problem may be solved in parallel by decomposing the grid into partitions, and mapping partitions to processors. During an iteration a processor updates its grid points, and then exchanges with other processors information necessary to compute the next iteration. As pointed out in [12], a large number of factors affect the performance of the resulting parallel computation: discretization stencil, partition shape, and parallel architecture. The analysis in [12] quantifies these relationships for a wide variety of stencils, shapes, and architectures. Their work throughout assumes that all processors in a parallel system are employed. This paper uses their framework to determine the largest possible speedup for a given problem, and to consider the behavior of that optimal speedup as a function of problem size when the number of available processors is not limited. These issues are important when we consider that users of large scientific codes will always want to solve a larger problem than the current technology supports. By focusing on the best possible speedup we are better able to access the suitability of various architectures for scaling up to larger problems, and the effects that various problem parameters and architecture parameters have on that suitability.

We will consider both strip and square partitions; although it is well known that squares have a higher computation to communication ratio, situations exist where the use of strips yields better performance than squares [13]. Other authors have employed strips [7] when the number of available processors is not a power of 4 (to avoid this last problem, we show that "nearly square" partitions perform within a few percentage points of true squares).

It is a folk theorem among the parallel scientific processing community that good speedup can be achieved simply by increasing the size of the problem. In fact, our analysis shows that this is indeed true for several different types of architectures, *provided that the maximal number of processors is fixed.* However, by allowing the number of processors (and supporting communication network) to grow along with the problem size, it becomes clear that some architectures are better suited for large problems than others. Architectures with hypercube or grid communication networks are shown to give linear optimal speedup in the grid size n^2, while bus-oriented networks are shown to give optimal speedup which increases at best in the cube root of n^2. The effect of the relationship between fixed communication overhead costs and bus bandwidth is shown to be important. We show that banyan type switching networks give optimal speedup which is $O(n^2/\log(n))$. From these results it is clear that bus networks are unsuited for large numerical problems of the type we consider. While hypercubes give better asymptotic optimal speedup than banyan networks, the true difference for grid sizes used in practice will not depend on the banyan network's log factor, but on the relative speeds of the communication networks.

2. Previous Work

Partition geometry plays a key role in determining communication costs, consequently much of the literature related to domain decomposition concerns the partition's geometric shape. Strips, squares, triangles, and hexagons have been considered in [4,12,16] on both message-passing and shared memory architectures. Reed, Adams and Patrick [12] have done a careful analysis of the relationships between discretization stencils, partition shape, parallel architecture, and data structure management. Their model determines which stencil/partition/architectures trios are best suited for each other. We will introduce their model in the main partion of the paper. Neither the analysis in [12] nor other work concerning partition shapes has explicitly focused on optimizing the number of processors used, or on the behavior of optimal speedup as the problem size increases.

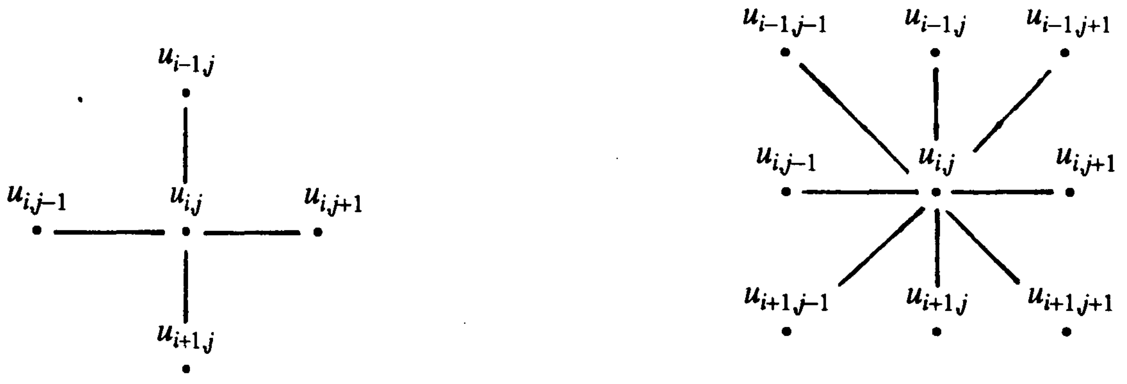
An analytic study of a conjugate gradient algorithm on the Finite Element Machine (FEM) is found in [1]. Their approach to modeling the computation is similar to ours, but is focused entirely on the FEM. The difference between the algorithm they study and the class of algorithms we study led to different conclusions concerning asymptotic performance.

Other related work uses a more abstract model of a parallel computation. In [6], Indurkya, Stone, and Cheng consider the module assignment problem under the assumptions of random module execution times and random communication patterns. They explicitly set out to determine the optimal number of processors to use. Convenient approximations were made to make the overall execution time more tractable; some of these approximations were removed by Nicol in [9], where it is shown that Indurkya's conclusions are basically sound despite the approximations (all of Indurkya's conclusions hold rigorously if module execution times are constant). The cost function studied in that work was the sum of execution time with the expected communication overhead. Their somewhat surprising conclusion is that the optimal assignment of modules to processor is extremal: either all modules are assigned to one processor, or the modules are distributed as evenly as possible across all available processors.

The cost model studied by Indurkya et al. and Nicol fails to capture the potential overlap of communication and computation in some architectures. Stone [15] also realized this, and gives a thorough analysis of a number of simple cost and communication models for the module assignment problem. Several of these models allow situations where adding processors increases execution time, so that the optimal assignment need not be extremal. For computations captured by these models, finding the optimal number of processors becomes an important issue. Stone uses a parallel solution of the Poisson equation to illustrate the relationship of these models to a real problem. His discussion does not treat the relative merits of partition geometries and stencils, although he does consider partitioning domain rows into pieces. A similar abstract view of this problem is given by Cvetanovic [3]. In contrast, our goal in this paper is to show how to optimize the size of a given partition shape for a given PDE on a given architecture. We then use the optimal size to characterize the suitability of the architecture for large numerical problems.

3. Model Description

A square physical domain is discretized into an $n \times n$ grid of points, and constant boundary values are assumed. Depending on the algorithm used, the value at a grid point $u_{i,j}$ is updated according to a *discretization stencil.* For example, figure 1 shows a 5-point stencil and a higher order 9-point stencil



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

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

Figure 1 5-point and 9-point stencils with update equations for the Laplace equation, solved using point Jacobi iteration. The equations clearly show that the stencil has a direct impact on the amount of computation performed. A grid partitioned into squares is shown in figure 2. From the equations in figure 1, we see that a grid point on the partition boundary of one square needs the values of one or more grid points in adjacent squares. Consequently the chosen stencil also affects the amount of communication. Since every boundary point must be communicated, the perimeter of a partition's shape affects communication volume. For example, a rectangular strip with $r \cdot n$ points has $2(r + n)$ boundary points, while a square partition with $r \cdot n$ points has $4\sqrt{r \cdot n}$ points; $2(r + n) \geq 4\sqrt{r \cdot n}$. Furthermore, some stencils require the communication of more than just one perimeter boundary; for example, see figure 3. Partitions are categorized in [12] with respect to a given stencil by the number of "perimeters" that must be communicated when the stencil is used. Following this

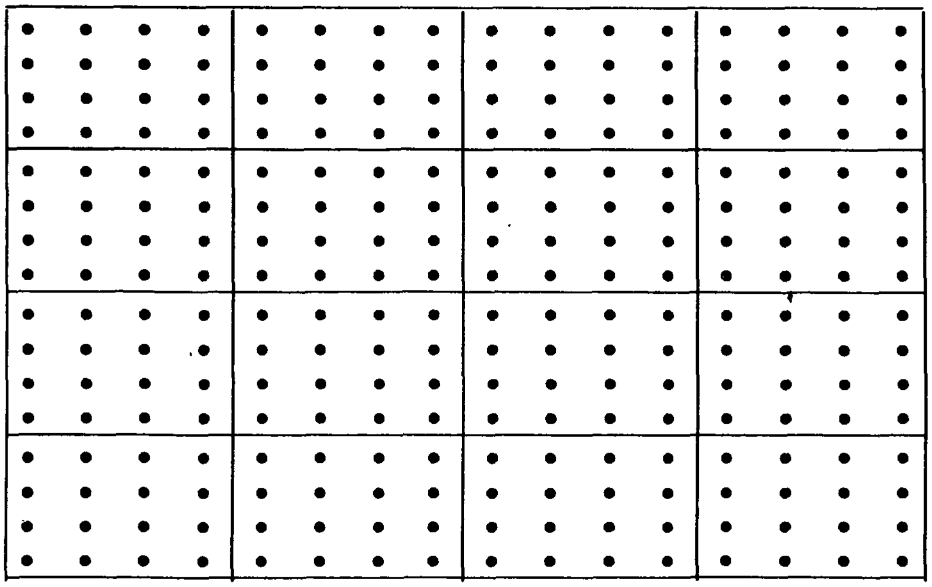
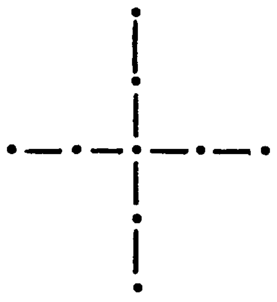


Figure 2 Square partitions on grid


9-cross stencil

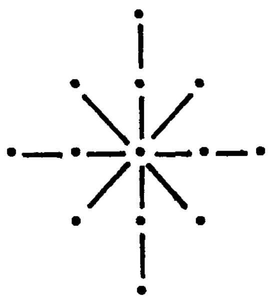

13-point stencil

Figure 3 Stencils requiring more than one perimeter communicated

idea, we define $k(P, S)$ to be the number of perimeters communicated by partition P using stencil S. Some values of $k(P, S)$ are given below.

Partition	*Stencil*	*k(Partition, Stencil)*
Strip	*5 point*	*1*
Square	*5 point*	*1*
Strip	*9 cross*	*2*
Square	*9 cross*	*2*

Assuming that one iteration cannot begin until the last iteration has ended, it is reasonable to model the iteration execution time (or cycle time) by

$$t_{cycle} = t_{comp} + t_a \tag{1}$$

where t_{comp} is the computation time of a single partition, t_a is the data access/transfer and synchronization time of a single partition. This model is essentially identical to that in [12] and [16] (although we have coalesced communication and synchronization times). t_a depends on the number of processors used and the underlying communication architecture. We will develop specific forms for t_a as needed.

The computation time t_{comp} depends on the stencil, the solution algorithm, the time to perform a floating point operation, and the number of grid points in a partition [1]:

$$t_{comp} = E(S) \cdot A \cdot T_{fp}.$$

Here $E(S)$ is the number of floating point operations per grid point employed by the algorithm (assumed to be constant), A is the number of grid points in a partition, and T_{fp} is the time for a floating point operation.

With A grid points per partition the number of processors used is $\frac{n^2}{A}$. For a given architecture we will optimize the number of processors by choosing the value of A which minimizes t_{cycle}, subject to

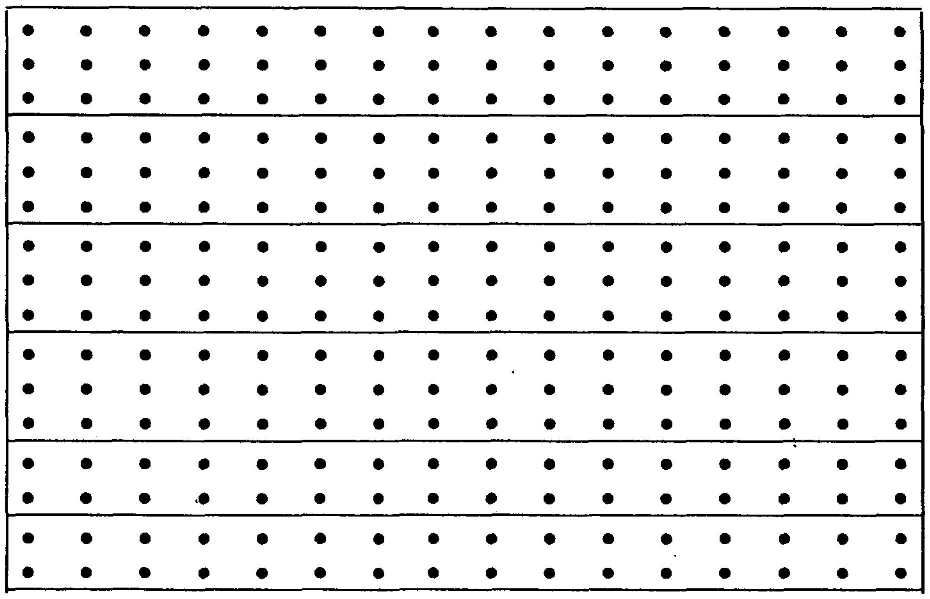


Figure 4 Strip partitioning of domain

[1]We implicitly assume that the costs of floating point operations strongly dominate the cost of a grid point update. Other overhead (such as address calculation and loop indexing) can be added to the model as needed.

memory constraints and processor availability constraints. Other constraints concern the partition's shape. Square partitions only admit values of A which are perfect squares, thereby reducing substantially the number of feasible domain decompositions (and hence freedom in choosing the number of processors). Furthermore, it is possible to assign exactly equal work to each processor only if the number of processors divides the number of grid points evenly. We will therefore relax the requirements that each partition have *exactly* the same number of points, and when using square partitions relax the requirement that partitions be *exactly* square.

It is easy to decompose the domain into strips for P processors: if $n = k \cdot P + r$ with $0 \leq r < P$ then r processors receive $\lceil \frac{n}{P} \rceil + 1$ contiguous rows, and the remaining processors each receive $\lceil \frac{n}{P} \rceil$ contiguous rows. As illustrated by figure 4, the number of communicating boundaries is the same as if all the partitions have equal work. Square partitions raise harder problems. We will approximate square partitions with nearly square rectangles which cover the domain in a nice way. The rectangles are arranged in a grid fashion as illustrated in figure 5. The domain is first divided into strips as before; then into rectangles by defining a border every *mth* column. We require that m divide n evenly, and call these *legal rectangles*. For tractability our analysis treats partition execution and communication costs as though the partitions are squares. However, empirical studies described below show that the error introduced by this assumption is small.

For a given n it is easy to calculate the area of each legal rectangular partition. For each calculated area A we determine the legal rectangle with area A whose perimeter is minimized (several different legal rectangles may have the same area). If its perimeter is within 5% of $4\sqrt{A}$ (the perimeter of a square with area A), we retain the rectangle and discard all other rectangles with area A. Otherwise we discard **all** legal rectangles with area A, since none are sufficiently square-like. Each remaining rectangle is a *working rectangle*. Not every area A will have a working rectangle with area A. Now suppose we analytically determine that squares with area A optimize performance. We need to find a

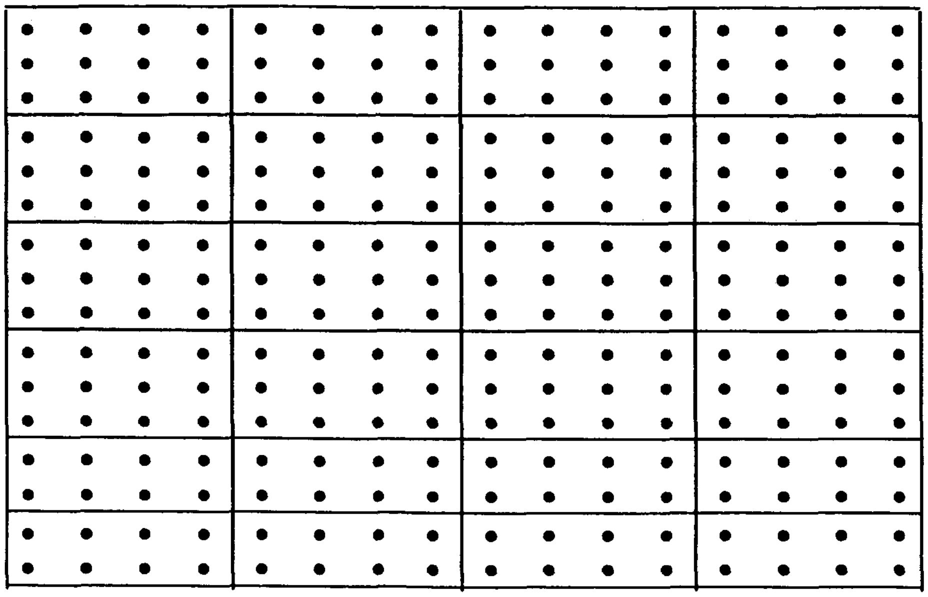
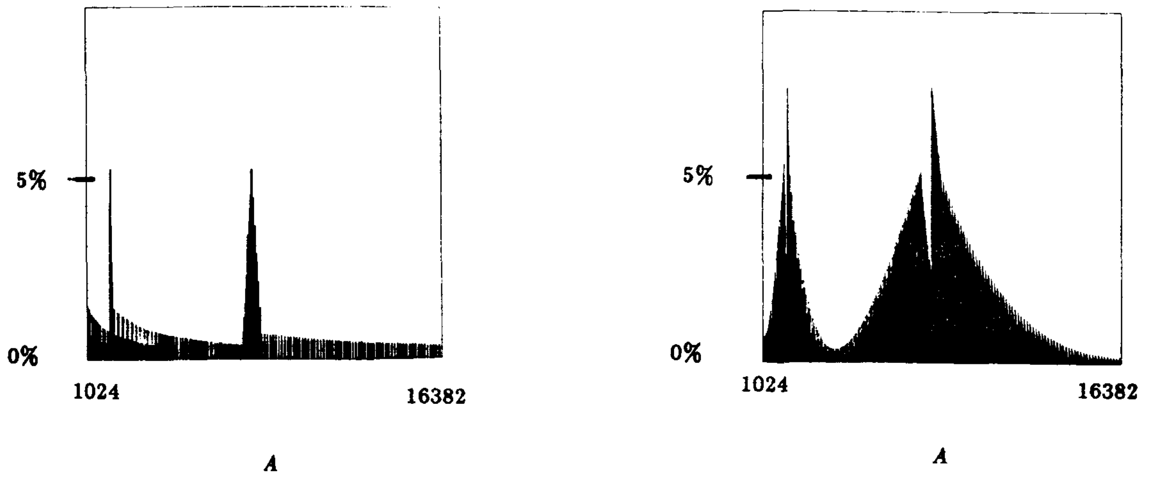


Figure 5 Rectangular partition of domain

working rectangle which closely approximates a square with area A. Figure 6a shows the relative approximation error in area for a 256 ×256 grid when we choose the working rectangle with area closest to A; Figure 6b shows the relative approximation error in perimeter. A ranges from 1024 to 16384 (every even value of A is plotted), reflecting decompositions using 4 to 64 processors. We see that the error introduced by this approximation is quite small, usually less than 3% for area and less than 6% for perimeter. Similar results were obtained for 128×128, 512×512, and 1024×1024 size grids. We can consequently optimize partition area as though partitions are exactly square with the assurance that the costs obtained are not far different from costs that are truly achievable. We next consider this optimization for various architecture types.



(a) Relative magnitude error in area **(b)** Relative magnitude error in perimeter

Figure 6 Bar graphs of approximation errors

4. Hypercube

Due to its commercial availability and interesting topological properties, a hypercube architecture such as the Intel iPSC[11] is a natural candidate for PDE solutions. The hypercube's rich communication topology allows the mapping of adjacent strips (or square) partitions onto processors in such a way that logically adjacent partitions are mapped onto physically adjacent processors (at least with stencils having no diagonals). This property is very important, because it implies that there is no contention for communication resources between non-logically adjacent partitions. The cost of sending a packet of data from one partition to another is independent of the total amount of communication on the system. We may model the communication delay of a V byte message from one processor to an adjacent processor as

$$t_a = \alpha \cdot \left\lceil \frac{V}{packetsize} \right\rceil + \beta$$

where α is the per packet transmission cost, and β is a startup cost. We assume that the problem size is fixed at n^2, and that if N processors are used, each partition gets n^2/N points. Thus, as we allow N to increase for a fixed value of n^2, the number of points in a partition *decreases*, so that both the execution cost (t_{comp}) and the communication/synchronization cost (t_a) for the partition decreases. This implies that t_{cycle} as defined in equation (1) is a decreasing function of N over the interval $[2, n^2]$. If only one processor is used then no communication costs are suffered; if the one processor execution cost is still greater than the two processor cost, then using all processors is optimal. If the one processor cost is less than the two processor cost, but greater than the cost of using all processors, then using all processors is again optimal. The last possibility is that the communication costs are so high that the one processor cost is less than the cost of using all processors. In this case, using only one processor is optimal. Thus we see that t_{cycle} is minimized by either spreading the computation out over as many processors as possible, or by placing the whole domain into one processor. If memory limitations prohibit the latter option, then the computation should be spread maximally.

Assume that the grid is spread across all available processors as squares, and consider the effect of increasing *both* n^2 and N in such a way that the number of grid points per processor remains constant (say F points per processor) as n^2 increases. This implies that the optimal cycle time is the constant [2] $C = E(S) \cdot F \cdot T_{fp} + 8(\lceil \frac{4\sqrt{F}}{packetsize} \rceil \alpha + \beta)$. The optimal speedup is then $\frac{E(S) \cdot n^2 \cdot T_{fp}}{C}$, which is linear in n^2.

If the number of processors is fixed at N, the cycle time of a processor is then

$$t_{cycle} = \frac{E(S) \cdot n^2 \cdot T_{fp}}{N} + 8(\lceil \frac{V(n^2)}{packetsize} \rceil \alpha + \beta).$$

where $V(n^2)$ denotes the volume of a partition's communication. $V(n^2) = 2n$ for strips and

[2]This expression assumes that only one communication port can be active at a time in a processor, and that the communication link is half duplex.

$V(n^2) = \sqrt{n^2/N}$ for squares; it is easily checked that speedup for both squares and strips approaches N as $n^2 \rightarrow \infty$.

The quick analysis above fails to consider the very important activity of convergence checking. A convergence check requires that every updated grid point value be compared with its last value. Depending on the convergence criterion employed, another iteration is called for if the updated solution is too "different" from the last estimate. Every partition determines whether its subgrid is converged and produces either a convergence flag, or a number (e.g. sum of squared update differences over subgrid) which must be disseminated throughout the entire network. For small stencils like 5-point, the additional computation required to do a convergence check can be 50% of the grid update computation. Furthermore, communication during the dissemination stage is not local, and the delay due to this stage increases in the number of processors used. Saltz, Naik, and Nicol examine this problem in [13], and note that the communication cost for convergence checking is extremely high due to message packaging and handling costs. They then give algorithms for *scheduling* convergence checks; measurements taken on an Intel iPSC show that despite the potentially very high cost of convergence checking, these algorithms reduce that cost to an insignificant amount. For the sizes of hypercubes currently available, we may safely ignore convergence checking costs in hypercubes.

5. Grid Architectures

Parallel architectures have been designed with nearest neighbor communication, e.g. the Illiac IV [5], and NASA's Finite Element Machine (FEM)[1]. The observations made for hypercubes apply equally well: the communication costs increase as the partition size increases, implying that the work should be spread as evenly as possible or lumped onto one machine (which makes little sense on the fore-mentioned machines). This type of machine often provides a global bus, and additional hardware for functions such as convergence checking. Provided that such additional hardware exists, the communication overhead of convergence checking does not appear to be as significant a concern as it is with hypercubes (although the additional computational cost may still be significant).

Adams and Crockett [1] analyze a conjugate gradient code on the FEM. Each iteration of this code requires every processor to send every other processor a number, and a processor adds together all such numbers. Eventually adding more processors to a fixed size problem causes this communication and addition to dominate performance. The result is that increasing the number of processor past a certain threshold increases the algorithm execution time. This highlights the fact that the monotonicity we claim for hypercube and grid machines depends very much on the exclusively nearest neighbor communication pattern. In the next section we will see that in bus architectures the communication cost can actually decrease in increasing partition size, making for a more interesting optimization problem.

6. Bus Architectures

Shared memory bus architectures are another important class of commercially available parallel processors. Currently, several vendors offer a few tens of processors on a common bus; we denote the maximum number of processors available by N. We suppose that the architecture supports local memory *and* global memory, with global memory access being several times slower than local memory access (several of the commercial machines do not support this model; they do support caches which, if sufficiently large could be viewed as local memory). We will consider both synchronous and asynchronous busses: a synchronous bus requires a processor requesting service to wait until that service is completed; an asynchronous bus admits overlapping computation and data writes to the global memory. We will see that in both cases contention for global memory via the bus can degrade performance to the point where adding processors decreases execution time.

Reed et al. [12] also observe that a processor's management of boundary values makes an important difference in performance. Following their advice, we will assume that each processor copies its neighbors' boundary points into local memory at the start of an iteration, and writes its own boundary points out to memory at the iteration's end. In our experience on the FLEX/32 [8], the cost of transferring a word to or from common memory is best modeled (ignoring contention) as $c + b$, where c is a fixed overhead cost due to address calculation and any overhead for accessing the bus, and b is the bus cycle time. Because all communication is serialized by a bus, the relative importance of different types of communication can be compared by their volume. The cost of communicating convergence checking information on bus architectures is insignificant because it involves only one number from each processor, and is hence ignored here.

6.1. Synchronous Bus

We model a synchronous bus and the contention it imposes by assuming that if P processors are simultaneously requesting service, the effective delay seen by each processor is $c + bP$ time per floating point number [3]. The transfer time t_a depends on the partition and on P. For strips with area A, each partition has $2n$ boundary points, and $\frac{n^2}{A}$ processors simultaneously require bus access. t_a for strips is consequently given by

$$t_a^{strip} = 4 \cdot n \cdot k(strip,S)(c + b\left\lceil \frac{n^2}{A} \right\rceil) = \frac{4 \cdot n^3 \cdot b \cdot k(strip,S)}{A} + 4 \cdot n \cdot c \cdot k(strip,S).$$

The cycle time is then

$$t_{cycle}^{strip} = E(S) \cdot A \cdot T_{fp} + \frac{4 \cdot n^3 \cdot b \cdot k(strip,S)}{A} + 4 \cdot n \cdot c \cdot k(strip,S). \qquad (2)$$

Note that the communication costs expressed by equation (2) are *decreasing* in A, making (2) the sum of a convex increasing term and a convex decreasing term. Equation (2) is consequently a convex function of A, so that the $\hat{A}$ minimizing (2) is easily found using calculus. If the $\hat{A}$ so determined falls outside of bounds placed by memory or processor limitations then either the least or the largest admissible value of A optimizes performance. $\hat{A}$ is given by

[3]For our problem, this assumption yields the same performance as if every processor were able to retain the bus for its entire transmission. This follows since one processor will be last to receive the bus; its effective communication time is $c + bP$ per floating point number. This model also implicitly assumes that available processors which are not participating in the computation do not significantly interfere with bus service.

$$\hat{A} = \left[\frac{4 \cdot n^3 \cdot b \cdot k(strip,S)}{E(S) \cdot T_{fp}}\right]^{1/2}. \tag{3}$$

It is important to note that $\hat{A}$ depends on most of the problem and architectural parameters assumed by the model (the overhead cost c does not affect $\hat{A}$). When $\hat{A} > \frac{n^2}{N}$ is not a multiple of n, we calculate $A_l = n\lfloor \frac{\hat{A}}{n} \rfloor$, and $A_h = A_l + n$. Between these two we choose the area yielding a smaller cycle time; the convexity of (2) ensures that this time is optimal among strips. Substitution of $\hat{A}$ into (2) gives the optimized cycle time when arbitrarily many processors are available,

$$t_{cycle}^{strip} = \left[E(S) \cdot T_{fp} \cdot n^3 \cdot b \cdot k(strip,S)\right]^{1/2} + \left[E(S) \cdot T_{fp} \cdot n^3 \cdot b \cdot k(strip,S)\right]^{1/2} + 4 \cdot n \cdot c \cdot k(strip,S).$$

Here we see that for sufficiently large n (or sufficiently small c) the computation time and the communication time are essentially identical. Then this expression shows what leverage we have in improving performance by improving hardware. For example, suppose that we have optimized performance for one set of architectural parameters, and wish to increase processor or bus speed. If we double the speed of the bus, the minimized cycle time decreases by a factor of $1/\sqrt{2}$; the same improvement is achieved by doubling the speed of a floating point operation. Since the original configuration was optimized, these factors bound from above performance gain we can achieve by doubling processor or bus speed on *any* subsequent partitioning of the domain. On the other hand if c is large relative to expected problem sizes, then the overhead cost $4 \cdot n \cdot c \cdot k(strip,S)$ will dominate the communication cost so that any speed increase in the bus will not significantly improve performance; on the other hand, decreasing c has a linear impact on t_{cycle}^{strip}.

Fewer than N processors should be used if $\frac{n^2}{\hat{A}} < N$. By (3), this is equivalent to

$$\frac{N^2 b}{T_{fp}} > \frac{E(S) \cdot n}{4} \cdot k(strip,S). \tag{4}$$

Inequality (4) gives a simple expression relating hardware characteristics to problem characteristics. If

$\hat{A} < \frac{n^2}{N}$, then the grid should be distributed across all N processors, giving a cycle time of

$$t_{cycle}^{strip} = \frac{E(S)\cdot n^2\cdot T_{fp}}{N} + 4\cdot n\cdot b\cdot N\cdot k(strip,S) + 4\cdot n\cdot c\cdot k(strips,S).$$

Using this expression we calculate speedup

$$Speedup_N^{strip} = \frac{N\cdot E(S)\cdot T_{fp}}{E(S)\cdot T_{fp} + \frac{(4\cdot b\cdot N^2 + 4\cdot c)\cdot k(strip,S)}{n}} \tag{5}$$

which is seen to approach N as $n^2 \rightarrow \infty$.

Square partitions are handled similarly. The communication time for a square partition with s points per side is [4]

$$t_a^{square} = 8\cdot s\cdot k(square,S)(c + b\left\lceil\frac{n^2}{s^2}\right\rceil) = 8\cdot k(square,S)\cdot b\cdot\frac{n^2}{s} + 8\cdot s\cdot c\cdot k(strips,S),$$

a quantity which is always smaller than the corresponding cost for strips with s^2 points. Also note that the increasing or decreasing behavior of this cost in s is strongly dependent on the relative values of b and c. The cycle time using squares with s points per side is

$$t_{cycle}^{square} = E(S)\cdot s^2\cdot T_{fp} + 8\cdot k(square,S)\cdot b\cdot\frac{n^2}{s} + 8\cdot s\cdot c\cdot k(strips,S).$$

The importance of the relationship between c and b on optimal allocation of processors is illustrated by considering necessary conditions under which fewer than all processors are optimally used. Differentiating t_{cycle}^{square} with respect to s and setting equal to zero yields the equation

$$E(S)\cdot T_{fp}\cdot s^3 + 4\cdot k(square,S)\left[c\cdot s^2 - b\cdot n^2\right] = 0.$$

Now suppose that t_{cycle}^{square} is minimized by $\hat{s}^2 = \frac{n^2}{P}$, $2 \leq P \leq N$. Then P processors are employed, and

[4]This expression assumes that the number of boundary points a partition writes to global memory is the same as the number read in. This is not rigorously true for any stencil which uses diagonals: our expression does not count diagonal elements required by the 4 corner points. However, when the number of partition points is large relative to the number of processors, this approximation is reasonable.

$\hat{s}$ is a root of the equation above. Substituting this $\hat{s}^2$ back into the equation above, we find that a necessary condition on P is that $c/b \le P$. Recalling that bus architectures typically have fewer than 30 processors, we see that this inequality tightly constrains values of b and c. Measurements taken on the FLEX/32 suggest that $c/b \approx 1000$, implying that numerical problems run on that machine should use all processors. Care in allocating processors is apparently needed more when c is less than b. Consequently, we now consider the extreme case of $c = 0$, and the optimal speedups that are achievable under that assumption. Note that any speedups so derived serve as upper bounds on speedups gained when $c \neq 0$.

If there is no overhead associated with accessing the bus, the optimal square partition size is easily shown to be

$$\hat{s}^2 = \left[\frac{4 \cdot n^2 \cdot b \cdot k(square,S)}{E(S) \cdot T_{fp}}\right]^{2/3}.$$

The cycle time using $\hat{s}^2$ points per partition is

$$t_{cycle}^{square} = (E(S) \cdot T_{fp})^{1/3}(4 \cdot n^2 \cdot b \cdot k(square,S))^{2/3} + 2(E(S) \cdot T_{fp})^{1/3}(4 \cdot n^2 \cdot b \cdot k(square,S))^{2/3},$$

which shows that the communication cost is twice that of the computation cost. This expression also shows that we have more leverage by improving communication speed than we do computation speed: doubling the speed of the bus gives an cycle time which is 63% of the original; doubling the speed of a floating point computation gives an cycle time which is 79% of the original. As with strips, simple algebra shows that fewer than N processors should be used if

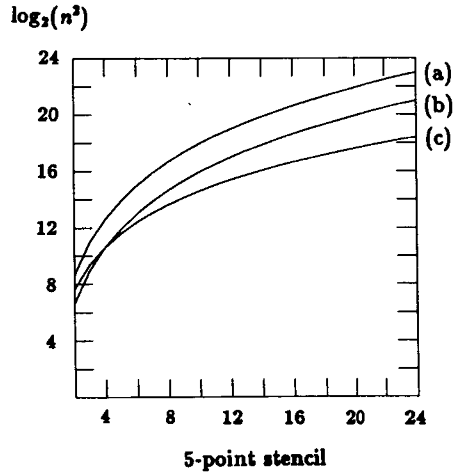
$$\frac{N^{3/2} \cdot b}{T_{fp}} > \frac{E(S) \cdot n}{4 \cdot k(square,S)}. \qquad (6)$$

Inequalities (4) and (6) show that a strip decomposition of a given problem will always call for fewer (or equal) processors than a square decomposition (provided that $k(square,S) = k(strip,S)$). The minimal problem size which uses all N processors is found by treating (6) as an equality, and solving for n. Figure 7 plots the the log (base 2) of the minimal problem size n^2 which gainfully uses all N processors, as a function of N. For the parameter values considered we see that a 256×256 grid with square partitions and a 5-point stencil should be solved on 1 to 14 processors; the same grid with a 9-point stencil should use 1 to 22 processors. The higher computation to communication ratio of the 9-point stencil allows more parallelism in computation for the same amount of communication.

For sufficiently large n^2 all N processors should be employed. The speedup achieved is

$$Speedup_N^{square} = \frac{N \cdot E(S) \cdot T_{fp}}{E(S) \cdot T_{fp} + \frac{2 \cdot b \cdot N^{3/2} \cdot k(strip,S)}{n}}$$

which also approaches N as $n^2 \rightarrow \infty$. Comparison of this speedup with speedup for strips (equation (5) with $c = 0$) shows the clear superiority of squares using realistic parameter values and large problems. Supposing that $E(S) \cdot T_{fp} = b$, $N = 16$, $k(strip,S) = k(square,S) = 1$, and $n = 256$ the speedup for strips is $\frac{16}{(1 + 512/n)} = 4$, while the speedup for squares is $\frac{16}{(1 + 128/n)} \approx 10.6$. Increasing the grid to



(a) Synchronous, Strip
(b) Asynchronous, Strip
(c) Synchronous, Square

Parameter	Value
T_{fp}	1×10^{-6}
b	5×10^{-6}

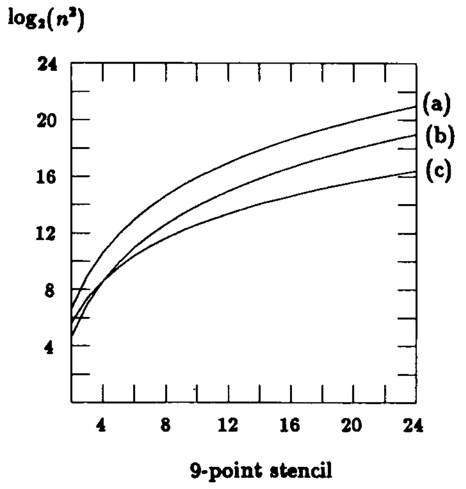


Figure 7 Minimal problem size as function of processors

1024×1024 raises the strip speedup to 10.6 and the square speedup to 14.2.

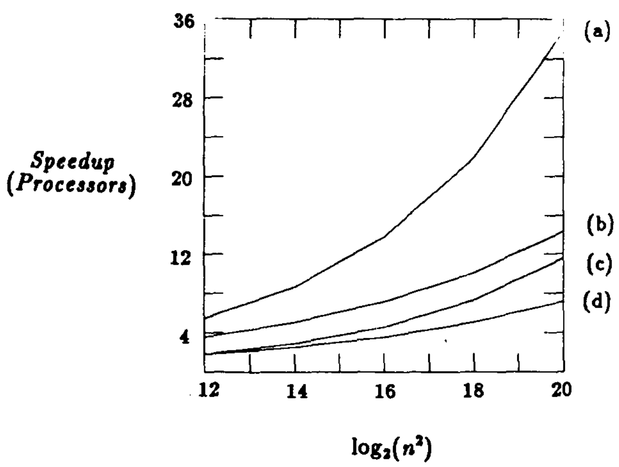
It is interesting (and straightforward) to calculate the optimal speedup when processors are not limited to N. For strips we obtain

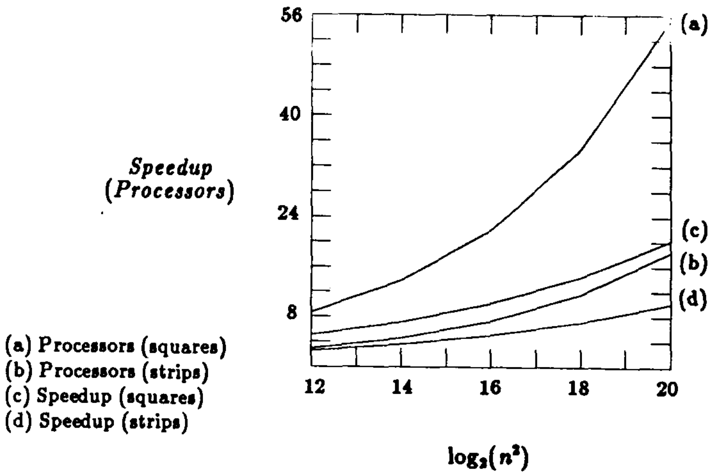
$$Speedup_{opt}^{strip} = \frac{n^{1/2}}{4}\left[\frac{E(S)\cdot T_{fp}}{b\cdot k(strip,S)}\right]^{1/2}.$$

This speedup is proportional to $(n^2)^{1/4}$, a rather disheartening figure. With squares we fair only somewhat better. Optimal speedup is

$$Speedup_{opt}^{square} = \frac{n^{2/3}}{3}\left[\frac{E(S)\cdot T_{fp}}{4\cdot b\cdot k(square,S)}\right]^{2/3};$$

a figure proportional to $(n^2)^{1/3}$. Figure 8 gives speedup curves and processor counts as a function of $\log(n^2)$ for the same problem parameters as addressed by figure 7. These unremarkable speedups support the common wisdom that bus architectures do not scale up. This does not negate the utility of





(a) 5-point Stencil **(b)** 9-point stencil

Figure 8 Speedup and processors required to achieve speedup

these machines: the speedups we calculated for a 16 processor machine on large grids were acceptable. However, significantly larger speedups for this same problem are possible using a (larger) hypercube. If minimizing the computation's execution time remains the prime objective then other architectures should be considered.

6.2. Asynchronous Bus

Better performance can be expected if we are able to overlap communication and computation. We next consider an architecture which allows asynchronous writes to global memory, but requires processors to wait for completion of their read requests. We then view an iteration as a reading phase, followed by a computation phase. During the computation phase, we assume that a boundary value is written to global memory as soon as it is updated. To maximize performance, we also assume that boundary values are updated before any other points.

The time required to read the boundary points is exactly half of t_a derived in the previous section. During the computation phase, a boundary point is updated every $E(S)\cdot T_{fp}$ units of time until all boundary points have been updated. The time required to update all A points in a partition is $E(S)\cdot A\cdot T_{fp}$. If at this time the bus has managed to complete all requested writes, then the iteration is finished. Otherwise, the iteration does not terminate until the bus services its backlog of boundary value writes. If a backlog exists after all points are updated and P processors are in use, then clearly the bus is unable to service P boundary value writes in time $E(S)T_{fp}$. Consequently, if a backlog exists, the bus has been fully utilized during the entire computation phase. We may therefore write

$$t_{cycle} = t_{read} + \max\{E(S)\cdot A\cdot T_{fp},\ b\cdot B_{total}\} \qquad (7)$$

where $t_{read} = t_a/2$ and B_{total} is the total load (summed over all processors) offered to the bus during the iteration.

For strips with area A, the cycle time is

$$t_{cycle}^{strips} = \frac{2 \cdot n^3 \cdot b \cdot k(strip,S)}{A} + \max\{E(S) \cdot A \cdot T_{fp}, \frac{2 \cdot n^3 \cdot b \cdot k(strip,S)}{A}\}.$$

Again, this function is convex in A, with its minimum precisely where the arguments to the max function are equal:

$$\hat{A} = \left[\frac{2 \cdot n^3 \cdot b \cdot k(strip,S)}{E(S) \cdot T_{fp}}\right]^{1/2}. \tag{8}$$

The corresponding area given by equation (3) for a synchronous bus is exactly a factor of $\sqrt{2}$ larger. As before, it is easy to show that fewer than N processors should be used if

$$\frac{N^2 \cdot b}{T_{fp}} > \frac{E(S) \cdot n}{2k(strip,S)}.$$

The optimal speedup is given by

$$Speedup_{optimal}^{strip} = \frac{n^{1/2}}{2\sqrt{2}}\left[\frac{E(S) \cdot T_{fp}}{b \cdot k(strip,S)}\right]^{1/2}.$$

Comparison with the synchronous bus speedup shows that the asynchronous bus speedup is a factor of $\sqrt{2}$ better.

The cycle time for a square partition with s^2 points is

$$t_{cycle}^{square} = \frac{4k(square,S) \cdot b \cdot n^2}{s} + \max\{E(S) \cdot s^2 \cdot T_{fp}, \frac{4k(square,S) \cdot b \cdot n^2}{s}\}.$$

This is a convex function of s which is minimized when the arguments of the max function are equal:

$$\hat{s}^2 = \left[\frac{4b \cdot n^2 \cdot k(square,S)}{E(S) \cdot T_{fp}}\right]^{2/3}.$$

This area is identical to that calculated for the synchronous bus case. The asynchronous bus optimal speedup is

$$Speedup_{optimal}^{square} = \frac{n^{2/3}}{2}\left[\frac{E(S) \cdot T_{fp}}{4 \cdot b \cdot k(square,S)}\right]^{2/3}$$

which is 150% larger than the synchronous bus speedup.

The most interesting thing to note about our asynchronous bus results is their relationship to the synchronous bus results. For both strip and square partitions we observe that optimal asynchronous bus performance is a constant (albeit substantial) factor better than synchronous bus performance. Constant factor improvement remains even if we relax the requirement that global memory reads are synchronous (in this case we assume that half the grid points are updated in parallel with the initial read requests, the other half in parallel with the boundary writes; this gives an additional 126% improvement in speedup). The inevitable contention for communication resources, even when conducted in parallel with computation and even when fixed overhead is ignored, constrains the optimal speedup to be $O((n^2)^{1/4})$ for strips and $O((n^2)^{1/3})$ for squares.

7. Switching Networks

An important class of parallel machines are those which communicate over a banyan type switching network (e.g. IBM RP3 [10], BBN Butterfly[2]). For a fixed sized network it is messy to do an exact analysis of the communication delay suffered by a partition as a function of processors used. To simplify things we make the following assumptions:

(1) The number of global memory modules is equal to the number of processors;

(2) Each processor has local memory, and only boundary values are stored in global memory;

(3) The network switches are 2 by 2;

(4) The network is sufficiently fast so that we can ignore contention while boundary values are asynchronously written to global memory.

Item (1) does not make any assumptions about the location of the global memory modules. They may be resident in processors (as with the BBN Butterfly) or not. Assumption (2) is used because the study in [12] shows that performance can be much better if local memory is employed. Assumption (3) allows us to avoid switch contention under certain circumstances. Assumption (4) is reasonable, since we may also *schedule* the times at which processors write to memory to further avoid contention. It is convenient to assume that all of the boundary values a partition *reads* are stored in the same global memory module, different from any other partition's. When a processor writes its boundary values, it writes them to the different modules of processors which use those values. Then it is possible to assign these modules to partitions in such a way that no contention at switches is ever incurred by any boundary value read (presuming all partitions read concurrently). Under these assumptions the global memory access time for a read is

$$t_a = 2 \cdot w \cdot \log_2(N)$$

where w is the speed of a switch, and the factor of two reflects two trips across the network. An iteration consists of a phase of reading boundary values, followed by a computing phase. During the computing phase the boundary points are written asynchronously back to global memory. The cycle time for strip partitions with A points is given by

$$t_{cycle}^{strip} = 4 \cdot n \cdot k(strip,S) \cdot w \cdot \log_2(N) + E(S) \cdot A \cdot T_{fp}.$$

As a function of A, the cycle time is minimized when A is minimized, meaning that all available processors are employed. Similarly, the cycle time for square partitions with s^2 points is

$$t_{cycle}^{square} = 8 \cdot s \cdot w \cdot \log_2(N) + E(S) \cdot s^2 \cdot T_{fp}.$$

This latter time is increasing in s, and so is minimized when s is minimized. Like the hypercube, we see that problems mapped onto inter-connection networks ought to be lumped onto one processor, or distributed as completely as possible across all processors .

We now allow the size of the parallel system to increase with increasing problem size. For square partitions we fix F points per processor, making the cycle time

$$t_{cycle}^{square} = 8 \cdot \sqrt{F} \cdot k(square,S) \cdot w \cdot \log_2 \left[\frac{n^2}{F} \right] + E(S) \cdot F \cdot T_{fp},$$

giving $O\left[\frac{n^2}{\log(n)}\right]$ speedup, which is nearly linear in the problem size. Strip partitions force an increasing number of points per processor, and have $O\left[\frac{n}{\log(n)}\right]$ optimal speedup.

These switching network speedups differ from the hypercube speedups only by a factor of $1/\log(n)$; a factor which arises from the growing number of stages of the switching network as the problem grows. For the size of problems treatable in the near future, this log factor will not be as significant in determining performance as is switching network speed (for banyan networks), and message packaging costs (for hypercubes and grids).

8. Conclusions

A number of factors influence the performance of an elliptic PDE solution on a parallel architecture. Reed et al. [12] detail the interactions of stencil, partition, and architecture; we use their framework to look at issues in processor allocation, and maximum possible speedup. For various types of architectures we developed equations describing execution time; invariably these functions turned out to be convex in the number of grid points assigned to a processor. This convexity shows that the best assignment of grid points to processors either (1) uses as few processors as possible, (2) uses as many processors as possible, or (3) there is a unique preferred assignment which does not use all available processors, and is easily determined using calculus. We show that for any collection of model parameter values, optimal performance on hypercubes, grid-like, and switching network types of architectures is achieved *either* by spreading the problem grid across all processors, or by forcing the grid into as few processors as possible. This result depends heavily on the fact that communication for the algorithm studied is strictly nearest neighbor; existing studies [1] provide counter-examples for other communication patterns. For our problem, both synchronous and asynchronous bus architectures allow for optimal assignments which do not use all processors. However, we showed that in order for this situation to arise, the fixed overhead cost of communicating a word on the bus must be nearly as small as the bus cycle time. Our formulas predict the smallest grid size which needs all available processors to perform optimally; they also give upper bounds on the optimal speedup possible. We noted that bus architectures can achieve acceptable speedup on reasonably sized grids, despite the potential for relatively high contention for global memory. Also, by looking at optimized execution times on bus architectures, we identify the leverage on performance given by increasing processor or network communication speed.

We also examined the suitability of these architectures for solving increasingly large problems. It is seen that for any of the fore-mentioned architectures with N fixed processors, the speedup approaches N as the grid size increases. More interesting is the behavior of optimal speedup when we let the architecture grow with the problem size. There we find that square partitions are strongly preferred over strip partitions; that hypercube speedups grow linearly in n^2, switching network speedups grow proportionally to $n^2/\log(n)$, and that bus architecture speedups grow only as $(n^2)^{1/3}$, even if bus access is completely asynchronous. Table I summarizes the optimal speedup in n^2 as a function of architecture (square partitions are assumed, one point per processor when appropriate).

Most of our results come as no surprise, they merely substantiate what is commonly thought about each of these architectures. The implications of these results are simply that communication volume and contention should be avoided as much as possible. Consider that when processors are no constraint, strip partitions have a communication volume which is a square root of the computation

Architecture	*Optimal Speedup*
Hyper-cube	$\frac{E(S)\cdot n^2\cdot T_{fp}}{8(\beta + \alpha)}$
Synchronous Bus	$\frac{n^{2/3}}{3}\left[\frac{E(S)\cdot T_{fp}}{4\cdot b\cdot k(square,S)}\right]^{2/3}$
Asynchronous Bus	$\frac{n^{2/3}}{2}\left[\frac{E(S)\cdot T_{fp}}{4\cdot b\cdot k(square,S)}\right]^{2/3}$
Switching Network	$\frac{E(S)\cdot n^2\cdot T_{fp}}{16\cdot w\cdot k(square,S)\cdot \log_2(n) + E(S)\cdot T_{fp}}$

Table I Summary of Optimal Speedups

volume. At best, we can expect speedup to grow in the square root of the computation volume. Allow contention proportional to total communication volume (summed over all partitions), and the optimal speedup drops to the fourth root of n^2. Even for squares, allowance of such contention restricts speedup to a cube root of n^2. The clear implication is that contention eventually causes serious performance degradation; our analysis shows how bad that degradation can be. It is also interesting to note the rather limited leverage we have on improving bus architecture performance by increasing processor or communication network speed: reducing the floating point time by $1/k$ decreases optimal execution time only by $\frac{1}{k^{1/3}}$; a similar reduction in bus time reduces optimal execution time by $\frac{1}{k^{2/3}}$. On the other hand for strip partitions, reducing the fixed overhead cost of communication decreases optimal execution time linearly.

One possible means for reducing contention is to use clever scheduling to access communication resources. We have not yet explored this possibility, but suggest that it is important to do so given the significance of the degradation our analysis predicts. Future effort will be devoted to verifying our analysis empirically, and to investigate the fore-mentioned scheduling issues.

References

[1] L.M. Adams, T.W. Crockett, "Modeling Algorithm Execution Time on Processor Arrays", *Computer*, vol. 17, July 1984, 38-44.

[2] *Butterfly Parallel Processor Overview*, BBN Laboratories Incorp., 1985.

[3] Z. Cvetanovic, "The Effects of Problem Partitioning, Allocation, and Granularity on the Performance of Multiple-Processor Systems", *IEEE Trans. on Computers*, C-36, April 1987, 421-432.

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

[5] R.W. Hockney, C.R. Jesshope, *Parallel Computers*, Adam Hilger Ltd, Bristol, 1981.

[6] B. Indurkhya, H.S. Stone, L. Xi-Cheng, "Optimal Partitioning of Randomly Generated Distributed Programs", *IEEE Trans. on Software Eng.*, vol. SI-12, pp. 483-495, March 1986.

[7] D. Kamowitz, "SOR and MGR[v] Experiments on the Crystal Multicomputer", University of Wisconsin Computer Science Technical Report 623, January 1986 (to appear in *Parallel Computing*).

[8] N. Matelan, "The Flex/32 MultiComputer", *Proc. 12th International Symposium on Computer Architecture*", Computer Society Press, Los Alamitos, CA, pp. 209-213, June 1985.

[9] D.M. Nicol, "Optimal Partitioning of Random Programs Across Two Processors", ICASE Report 86-53, August 1986 (submitted to IEEE Trans. on Software Eng).

[10] G.F. Pfister, W.C. Brantley, D.A. George, S.L. Harvey, W.J. Kleinfelder, K.P. McAuliffe, E.a. Melton, V.A. Norton, J. Weiss, "The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture", *Proceedings of the 1985 International Conference on Parallel Processing*, pp. 764-771, August 1985.

[11] J. Rattner, "Concurrent Processing: A New Direction in Scientific Computing", *Conference Proceedings of the 1985 National Computer Conference, AFIPS Press, Vol. 54, pp. 159-166, 1985.*

[12] D.A. Reed, L.M. Adams, M.L. Patrick, "Stencils and Problem Partitionings: Their Influence on the Performance of Multiple Processor Systems", ICASE Report 86-24, May 1986 (to appear in IEEE Trans. on Computers).

[13] J.H. Saltz, V.K. Naik, D.M. Nicol, "Reduction of the Effects of the Communication Delays in Scientific Algorithms on Message Passing MIMD Architectures", *SIAM Journal of Scientific and Statistical Computing*, Vol. 8, No. 1, January 1987, s118-s134.

[14] P.B. Schneck, D. Austin, S.L. Squires, J. Lehmann, D. Mizell, K. Wallgren, "Parallel Processor Programs in the Federal Government", *Computer*, vol. 18, no. 6, pp. 43-55, June

1985.

[15] H.S. Stone, *High Performance Architecture*, Addison-Wesley, New York, 1987.

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

Standard Bibliographic Page

1. Report No. NASA CR-178282 ICASE Report No. 87-7	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle PROBLEM SIZE, PARALLEL ARCHITECTURE, AND OPTIMAL SPEEDUP		5. Report Date April 1987
		6. Performing Organization Code
7. Author(s) David M. Nicol and Frank H. Willard		8. Performing Organization Report No. 87-7
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. NAS1-18107
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 505-90-21-01

15. Supplementary Notes

Langley Technical Monitor: J. C. South

Submitted to the International Conference on Parallel Processing

Final Report

16. Abstract

The communication and synchronization overhead inherent in parallel processing can lead to situations where adding processors to the solution method actually increases execution time. Problem type, problem size, and architecture type all affect the optimal number of processors to employ. In this paper, we examine the numerical solution of an elliptic partial differetnial equation in order to study the relationship between problem size and architecture. The equation's domain is discretized into n^2 grid points which are divided into partitions and mapped onto the individual processor memories. We analytically quantify the relationships between grid size, stencil type, partitioning strategy, processor execution time, and communication network type. In doing so, we determine the optimal number of processors to assign to the solution (and hence the optimal speedup), and identify (1) the smallest grid size which fully benefits from using all available processors, (2) the leverage on performance given by increasing processor speed or communication network speed, (3) the suitability of various architectures for large numerical problems.

17. Key Words (Suggested by Authors(s)) parallel processing, processor allocation, scientific computing	18. Distribution Statement 64 - Numerical Analysis 65 - Statistics and Probability Unclassified - unlimited

19. Security Classif.(of this report) Unclassified	20. Security Classif.(of this page) Unclassified	21. No. of Pages 30	22. Price A03

For sale by the National Technical Information Service, Springfield, Virginia 22161

NASA Langley Form 63 (June 1985)