SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 3, September 1984

008

ORDERING SCHEMES FOR PARALLEL PROCESSING OF CERTAIN MESH PROBLEMS*

DIANNE P. O'LEARY†

Abstract. In this work, some ordering schemes for mesh points are presented which enable algorithms such as the Gauss–Seidel or SOR iteration to be performed efficiently for the nine-point operator finite difference method on computers consisting of a two-dimensional grid of processors. Convergence results are presented for the discretization of $u_{xx}+u_{yy}$ on a uniform mesh over a square, showing that the spectral radius of the iteration for these orderings is no worse than that for the standard row by row ordering of mesh points. Further applications of these mesh point orderings to network problems, more general finite difference operators, and picture processing problems are noted.

Key words. parallel computation, mesh problems, finite difference methods, finite element methods, nine-point operator

1. Introduction. Consider the standard uniform mesh finite difference approximation to the equation

$$u_{xx}+u_{yy}=f(x, y)$$

in a square domain with appropriate boundary conditions. The equation for each mesh point involves data at that point and at its north, south, east, and west neighbors. The Jacobi iterative method for this problem converges, and the iteration matrix has a spectral radius of $\cos(\pi/n)=1-O(1/n^2)$, when the mesh is $n\times n$. The successive overrelaxation method (SOR) with optimal choice of the relaxation parameter gives an iteration an order of magnitude faster, with spectral radius $[1-\sin(\pi/n)]/[1+\sin(\pi/n)]=1-O(1/n)$. Thus, for this problem SOR is preferred over the Jacobi method for standard computers, since both take time proportional to n^2 per iteration. These standard results can be found, for example, in [23].

However, the Jacobi method has undergone a renaissance recently with the development of computers with parallel design. On a computer with n^2 processors connected in a two-dimensional grid with local communication only, one iteration of Jacobi can be completed in time independent of n, while SOR still requires $O(n)$ for the first iteration if the mesh points are ordered row by row. (However, successive iterations can overlap the first, and be completed in time independent of n.) More details on these implementations will be given in § 2.

SOR can be speeded for parallel computation by reordering the mesh points. For example, using the checkerboard ordering (Fig. 5a: all even numbered mesh points ordered after all odd points), the time per iteration using n^2 processors is again independent of n and the convergence rate is unchanged. The mesh can also be ordered by lines into a block scheme, so that all new values on a line are determined at once. If k lines are grouped together, the spectral radius is $1-O(k/n)$, but iteration time increases with n [18].

The checkerboard ordering does not work so well for more complicated elliptic equations or alternate approximation strategies. Whenever a finite difference mesh point (or finite element unknown) is linked to one of its diagonal neighbors, the checkerboard trick fails. The line methods are often still useful.

* Received by the editors December 27, 1982, and in revised form November 19, 1983. This work was supported by the Air Force Office of Scientific Research under Grant AFOSR-82-0078.

† Computer Science Department, University of Maryland, College Park, Maryland 20742.

One purpose of this work is to develop ordering strategies for use in solving certain discretizations of elliptic equations on parallel processors. These strategies are applicable to any problem in which the equation for each mesh point involves data at that point and any subset of its north, south, east, west, northeast, northwest, southeast, and southwest neighbors. The goal is to make iteration time independent of n without sacrificing convergence speed.

A second purpose is to note that these orderings are also useful in several classes of problems unrelated to partial differential equations.

This work is related to work performed independently by Adams [1]. In that paper, the four-color ordering of Fig. 5a is presented for the nine-point finite difference operator, and some multicolored orderings for other couplings of mesh points are also given. No theoretical results concerning rate of convergence are given, but numerical experiments on elliptic partial differential equations are reported.

There has also been other work on parallel iterative methods (see, for example, [8]). Most recent work (see [24] for an exception) has centered around implementation of the conjugate gradient algorithm and appropriate preconditionings. Sameh [22] discusses preconditioning partial differential equation problems by block Jacobi with line red/black ordering. Kowalik, Kumar, and Lord [10] discuss block Jacobi, and Kumar in her thesis [12] considers other preconditionings and examples. Lichnewsky [16] discusses preconditioning with an incomplete Cholesky factorization under the nested dissection ordering. Parter and Steuerwalt [19], [20] discuss convergence properties of various preconditionings based on block iterative methods.

Another aspect of the problem is the mapping of irregular mesh problems onto regular arrays of processors. One heuristic approach is given in [3]. The measure of success is taken to be maximizing the number of problem edges that match processor connections. In [7], the mapping problem is studied for adaptive local refinements of regular meshes.

In § 2 we present some background on parallel computation and mesh problems. In § 3 we present orderings for mesh points and discuss convergence rates for the system of equations corresponding to the nine-point finite difference approximation to the operator $u_{xx} + u_{yy}$. In § 4 we discuss implications for more complicated problems, including nonlinear systems of equations and constrained optimization problems.

2. Parallel computation of mesh problems. In this section, we consider sources of mesh problems and the implementation of the Jacobi, Gauss–Seidel, and conjugate gradient algorithms on parallel processors.

By a parallel computer system, we mean a set of processors, possessing some local memory, capable of performing some arithmetic operations and connected in some network so that each processor can communicate with "neighboring" processors and perhaps with common memory. Examples of parallel processors include the Denelcor HEP, the ILLIAC [2], DAP, BSP [11], FEM [9], the ZMOB [21], systolic arrays [13], wavefront array processors [14], [15], and plans for the Japanese Fifth Generation Computer System [17].

The examples we consider will assume that the processors are arranged in a two-dimensional grid. Each processor should have at least one connected neighbor in each adjacent row and column. This structure is of interest because in many sparse matrix problems, the graph of the matrix has the structure of a planar mesh. Such problems arise from diverse applications areas. Three are described below.

1) The discretization of elliptic partial differential equations imposes a regular or irregular grid on the region. In two dimensions, a finite difference method often results

in a rectangularly oriented grid in which each unknown is directly coupled to some subset of eight compass-point neighbors. Finite element methods over irregular regions produce less patterned grids, but are still characterized by local coupling only. Graded grids, which introduce refinement into some subregions, also occur commonly in such problems.

2) Network problems also exhibit a mesh structure, arising from limited connectivity between nodes of the network. Such problems are typical in electrical power system analysis, queuing theory models of communication networks, and geodesy.

3) Digital image processing problems also have mesh structure. In this case, the grid is usually quite regular, resulting from digital coding of a gray level or color level for each "pixel" or picture element. Typical pictures have 10,000–100,000 pixels. Key problems are noise smoothing, feature extraction (e.g., finding region boundaries), and scene analysis (e.g., determining the position of the light source). Often the problem is formulated as a constrained optimization problem

$$\min_{c \leqq u \leqq d} u^T A(u)u + u^T b.$$

In noise smoothing, for example, u is the vector of digitized color levels, c and d represent bounds on meaningful digitized colors, and A has the structure of a 9-point operator, since a color at one point is most closely coupled to the eight neighboring colors.

Jacobi-type iterative methods have been widely used for parallel computation of mesh problems. Application to partial differential equations and network problems seem to share a common heritage, but the developments for image processing were independent. Such iterations take the form

$$u_i^{(k+1)} = \Psi\{u_j^{(k)}: j \text{ is a mesh neighbor of } i\},$$

where Ψ is a function of several variables. Under a reasonable assignment of mesh values to processor nodes, the ith process can access neighboring values efficiently, update its own value, and be ready for the next iteration in time independent of the size of the mesh. An example is given in Fig. 1. Here we have a rectangular grid of processors, with nearest neighbor connections, applying the Jacobi iteration to a five-point operator. In many applications, however, these schemes are only slowly convergent, and more sophisticated methods are required.

Gauss–Seidel-like methods have also been considered. They take the form

$$u_i^{(k+1)} = \Psi[\{u_j^{(k)}: j \text{ is a mesh neighbor of } i \text{ and } j \geqq i\} \cup \{u_j^{(k+1)}: j \text{ is a mesh neighbor of } i \text{ and } j < i\}].$$

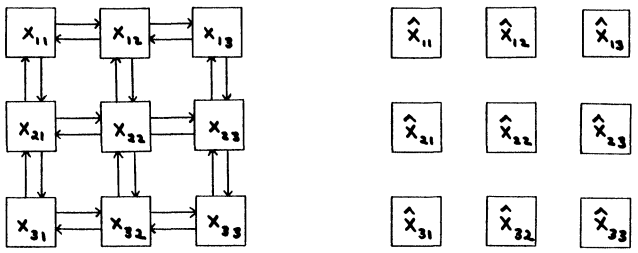

FIG. 1. *The Jacobi iteration for a five-point operator on a $n \times n$ grid of processors.* (a) *Step* 1: *Each processor passes its current mesh value to each of its neighbors.* (b) *Step* 2: *Each processor updates its mesh value.*

When expressed in this form, u_i cannot be updated until all of its neighbors with lower indices are updated. This method is illustrated in Fig. 2 for the same problem as Fig. 1 with row by row ordering of mesh points. The first iteration takes time proportional to n for an $n \times n$ grid, but each successive iteration takes only two more time units. Alternate orderings of mesh points are much better than this; we discuss them in the next section.

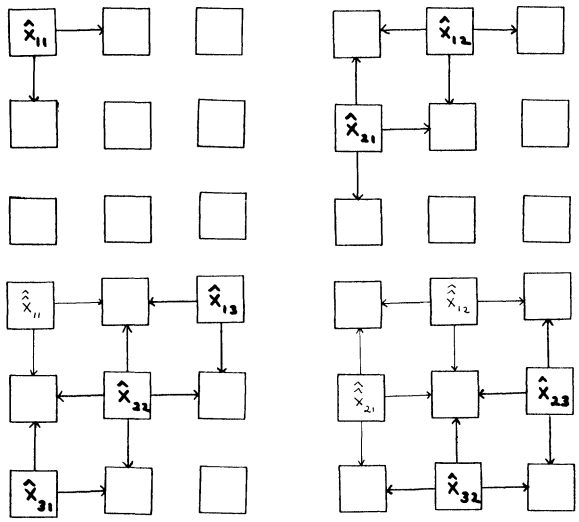

FIG. 2. *The Gauss–Seidel iteration for a five-point operator on a $n \times n$ grid of processors, using row by row ordering of the mesh points.*

An alternate way to consider Gauss–Seidel-type algorithms is to express them in iteration matrix form. To solve $Au = b$, for example, the iteration takes the form

$$u^{(k+1)} = (D-L)^{-1}[b + Uu^{(k)}]$$

where $A = D - L - U$, L is strictly lower triangular, and U is strictly upper triangular. Some researchers have proposed explicitly forming $(D-L)^{-1}$ or some approximation to it so that the iteration can be performed completely in parallel.

Because of success in solving problems on standard machines, methods like conjugate gradients are attractive candidates for parallel processors. They impose one further requirement on machine architectures, however: in addition to easy access to mesh neighbors, it is also necessary to accumulate inner products. On a rectangular $n \times n$ grid of processors with only nearest neighbor connections, this is an $O(n)$ process, quite slow for large grids. Some additional processor communication channels are necessary. Some alternatives follow:

(1) One common proposal is to add to each column of processors the ability to accumulate an inner product quickly using a bus.

(2) Perfect shuffle connections among processors in each row and column reduce inner product time to $O(\log n)$. Connections for a single column of $n = 16$ processors are shown in Fig. 3. Information in a processor is redistributed as if it were on a card being shuffled in a deck. For $n = 16$ processors, the successive reorderings are

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	9	2	10	3	11	4	12	5	13	6	14	7	15	8	16
1	5	9	13	2	6	10	14	3	7	11	15	4	8	12	16
1	3	5	7	9	11	13	15	2	4	6	8	10	12	14	16

The cycle repeats every $\log n$ steps. Thus, if originally each even processor j accumulates the jth and $(j+1)$st elements in the inner product, then at stage 2, processors numbered $2, 4, \cdots, 16/2$ can accumulate four terms. At the third stage, 8-term partial sums can be accumulated, and the 16-term inner product is available after $\log 16$ steps. It can then be communicated to each processor in another $\log n$ steps.

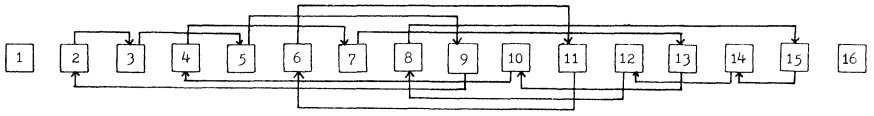

FIG. 3. *Perfect shuffle connections among* 16 *processors.*

(3) An arrangement with the same speed for inner products but with fewer connections and fewer wire crossings is shown in Fig. 4. In this incomplete interchange, the even processors send their information to the top of the grid in reverse order. The successive reorderings for $n = 16$ are

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
16	14	12	10	8	6	4	2	*	*	*	*	*	*	*	*
*	*	*	*	2	6	10	14	*	*	*	*	*	*	*	*
*	*	*	*	14	6	*	*	*	*	*	*	*	*	*	*

Again, inner products can be accumulated in $\log n$ steps and broadcast along the reverse pathways.

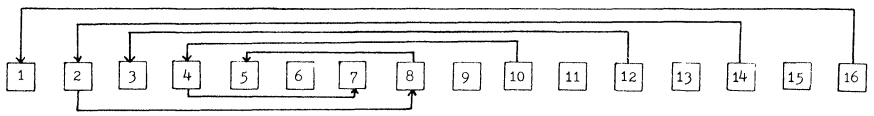

FIG. 4. *Incomplete interchange connections among* 16 *processors.*

It is interesting to note that either the perfect shuffle or the incomplete interchange connections shown above make multigrid iterations [4] possible on a nearest neighbor grid, since examination of the permutation pattern shows that within $\log n$ steps, rearrangements are made which could be used to place in proximity every other mesh point, every fourth mesh point, every eighth, etc.

3. Orderings for nine-point operators. Figure 5 shows orderings of mesh points which can make algorithms like SOR practical for parallel computation when the equation at each mesh point depends on the point itself and any subset of its eight immediate neighbors. To make the discussion clear, we will use the P^3 scheme as an example. The other schemes are similar and, in many cases, simpler.

Note in Fig. 5b that we have divided the mesh points into three groups. Those labeled "1" are to be ordered before those labeled "2", and those labeled "3" are last. Within each group, neighboring points—those in the same "P"—are numbered consecutively in an arbitrary way. The matrix corresponding to the mesh in Fig. 5b has the sparsity structure shown in Fig. 6. Notice that the pattern is

$$\begin{bmatrix} D_1 & A & B \\ A^T & D_2 & C \\ B^T & C^T & D_3 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \tag{1}$$

1	2	1	2	1	2	1	2	1	2
4	3	4	3	4	3	4	3	4	3
1	2	1	2	1	2	1	2	1	2
4	3	4	3	4	3	4	3	4	3
1	2	1	2	1	2	1	2	1	2
4	3	4	3	4	3	4	3	4	3
1	2	1	2	1	2	1	2	1	2
4	3	4	3	4	3	4	3	4	3
1	2	1	2	1	2	1	2	1	2
4	3	4	3	4	3	4	3	4	3

FIG. 5a. *Four colors are necessary for checkerboard ordering for a nine-point operator.*

1	1	2	2	1	1	2	3	1	1
1	1	3	3	1	1	3	3	1	2
1	2	3	3	2	2	3	3	2	2
2	2	3	1	2	2	1	1	2	2
2	2	1	1	2	3	1	1	3	3
3	3	1	1	3	3	1	2	3	3
3	3	2	2	3	3	2	2	3	1
3	1	2	2	1	1	2	2	1	1
1	1	2	3	1	1	3	3	1	1
1	1	3	3	1	2	3	3	2	2

FIG. 5b. *The P^3 ordering.*

where D_1, D_2, and D_3 are block diagonal matrices with blocks of size 5×5 or less, and the vector u_i consists of all variables numbered "i". For parallel processing, the $(k+1)$st step of this scheme would be as follows:

(1) Perform an iteration of block SOR on the first group of equations:

$$u_1^{(k+1)}=(1-\omega)u_1^{(k)}+\omega D_1^{-1}(v_1-Au_2^{(k)}-Bu_3^{(k)}).$$

Note that each "P" group can be processed independently and concurrently by a

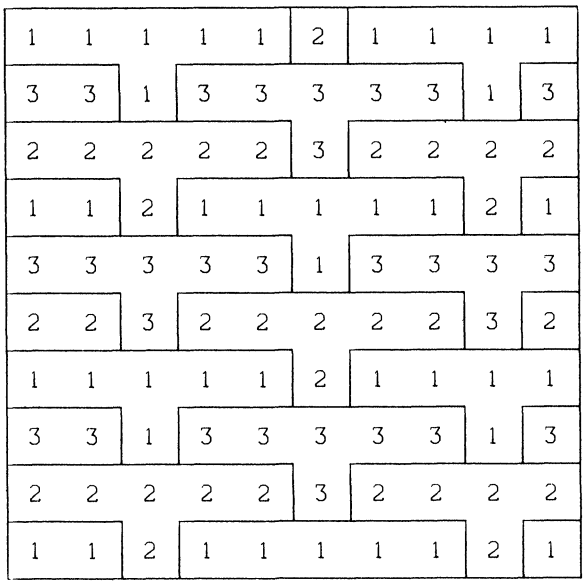

FIG. 5c. *The T^3 ordering.*

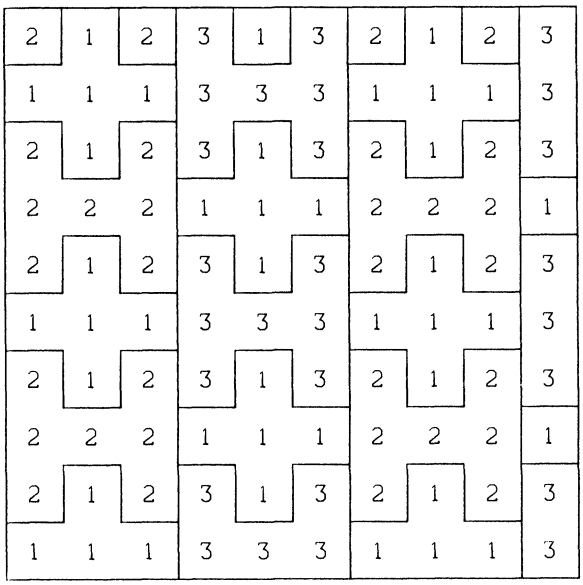

FIG. 5d. *The H + H ordering.*

separate processor or group of 5 processors. Only 5×5 linear systems need to be solved directly.

(2) Process the second group of equations similarly:

$$u_2^{(k+1)} = (1-\omega)u_2^{(k)} + \omega D_2^{-1}(v_2 - A^T u_1^{(k+1)} - C u_3^{(k)}).$$

Part of this computation could overlap (1).

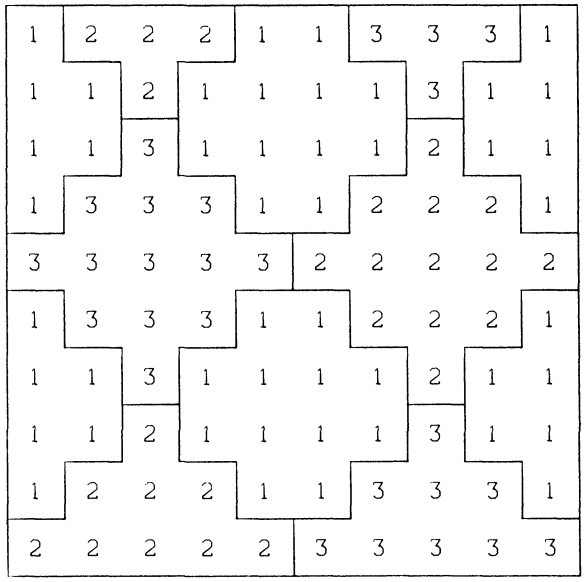

FIG. 5e. *The Cross ordering.*

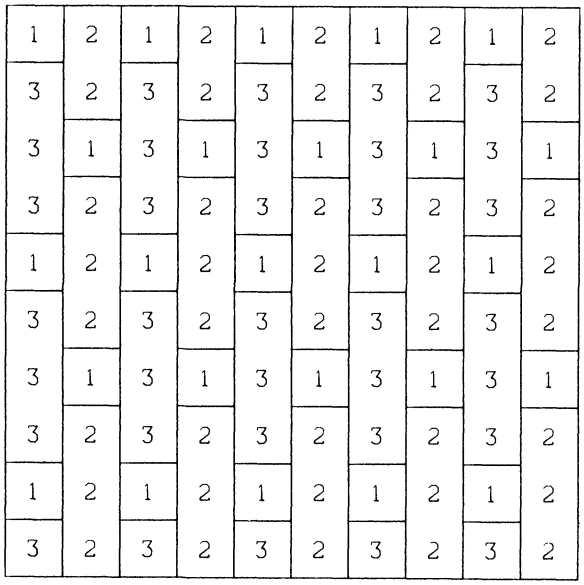

FIG. 5f. *The Box ordering.*

(3) Process the third group:

$$u_3^{(k+1)} = (1-\omega)u_3^{(k)} + \omega D_3^{-1}(v_3 - B^T u_1^{(k+1)} - C^T u_2^{(k+1)}).$$

Computation of $(1-\omega)u_3^{(k)}$ could overlap (1), and computation of $v_3 - B^T u_1^{(k+1)}$ could overlap (2).

This scheme can be implemented efficiently on computers with n^2 processors (one per point), $n^2/5$ processors (one per "P"), or $n^2/15$ processors (one per cluster of

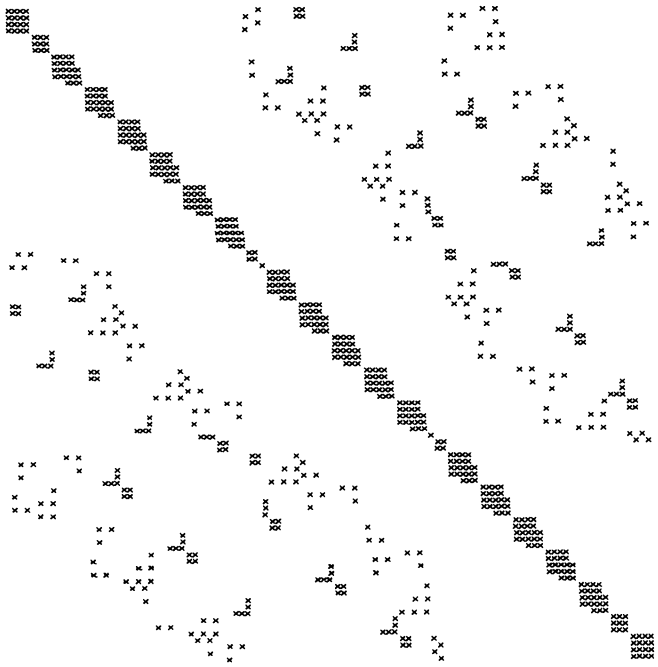

FIG. 6. *Matrix sparsity structure for the P^3 ordering.*

three "P's" numbered 1, 2, and 3). Communication is local: each P communicates with at most six of its neighbors, and by distributing the mesh points in the natural way, these will be on neighboring processors.

For illustration, we describe the algorithm for a two-dimensional grid of processors with communication connections to horizontal and vertical neighbors only, assigning three vertically adjacent "P's" to each processor. A processor's view of a typical iteration is as follows:

(1) For its block of 5 equations for $u_1^{(k+1)}$, each processor accumulates the terms involving points in u_1 and u_2 from information it already has. When the necessary u_3 values from the previous iteration arrive from (a subset of) the north, south, east, and west neighboring processors, then the u_3 terms are computed, a 5×5 linear system is solved, and then u_1 can be updated. Appropriate subsets of the 5 new u_1 values are then sent north, south, east, and west.

(2) Next, in a similar way, the processor accumulates terms for its 5 components of $u_2^{(k+1)}$ which involve points in u_2 and u_3 and completes the update after u_1 information arrives from the 4 neighbors. Then the 5 new u_2 values are sent north, south, east, and west as appropriate.

(3) The third set of 5 points is updated and communicated in the same way.

The iteration is synchronized by the data flow rather than by any global communication. If fewer processors are available, the "P's" can be enlarged, at the cost of solving linear systems larger than 5×5: each number in the "P" can represent a $j\times l$ group of mesh points for any integers j and l, giving $5jl\times 5jl$ systems to be solved. The iteration can be terminated after a fixed number of iterations or by a convergence test. If the communication required for a convergence test requires m times the time of an iteration, it could be performed roughly that often. Any global communication paths in the grid of processors (such as the ones described for inner products in § 2)

are idle during the iteration process, and thus could be dedicated to convergence communication.

To study the convergence rate of the P^3 SOR method (as well as the others in Fig. 5) we list some properties of the matrix of (1) corresponding to the nine-point difference approximation to the Laplacian. Let D denote the block diagonal part; $-L$ the strictly lower triangular part; and $-U$ the strictly upper triangular part. The main tools we use are

(T1) [23, p. 91] For a regular splitting M–N of a Stieltjes matrix (one which is symmetric, positive definite, and nonpositive off the diagonal), if M and N have no common nonzeros, then putting more nonzero elements in M monotonically improves the spectral radius of the iterative method

$$u \leftarrow M^{-1}Nu - M^{-1}v.$$

(T2) [23, p. 124] For irreducible Stieltjes matrices, the "SOR theory" holds "approximately"; i.e., for

$$\omega_{\text{opt}} = \frac{2}{1+\sqrt{1-\rho^2(J)}}$$

where $\rho(J)$ denotes the spectral radius of the Jacobi iteration matrix, then, for the SOR iteration using this value of the relaxation factor ω,

$$\omega_{\text{opt}} - 1 < \rho(\text{SOR}) < \sqrt{\omega_{\text{opt}} - 1}.$$

We have the following properties:

(a) The original matrix is an irreducible Stieltjes matrix.

(b) The P^3 Jacobi method ($M = D$, $N = L + U$) and the P^3 Gauss–Seidel method ($M = D - L$, $N = U$), are regular splittings and thus convergent. They are also p-regular splittings.

(c) By (T1), the rate of convergence for each P^3 method is better than that for the corresponding standard method.

(d) Consider dividing the mesh of Fig. 5b into blocks, each containing two vertical lines of mesh points. By (T1), the spectral radius for the P^3 Jacobi method is not less than that for the two-line Jacobi method (since it is independent of ordering) and not greater than the standard point-Jacobi method. Thus

$$\rho(J_{2\text{-line}}) \leqq \rho(J_{P^3}) \leqq \rho(J_{\text{point}}),$$

and, since $\rho(J_{2\text{-line}})$ and $\rho(J_{\text{point}})$ are both $1 - O(1/n^2)$ [18], so is $\rho(J_{P^3})$.

(e) By (T2), there is a value of ω for which $\rho(\text{SOR}_{P^3}) = 1 - O(1/n)$. Thus, in using the P^3 ordering we have not sacrificed rate of convergence.

The P^3 scheme is the most complicated of the schemes in Fig. 5. The pattern repeats a shifted scheme of 4 columns. The other schemes are more regular.

The P^3 and T^3 schemes are balanced, dividing the mesh points into three groups of equal size. The $H+H$ and *Cross* schemes have about twice as many "1" points as "2's" or ("3's"), and the *Box* scheme has three times as many "2's" (or "3's") as "1's".

The patterns can be enlarged in various ways; for example, the $H+H$ scheme can be stretched in both dimensions, with each number in Fig. 5c representing a $j \times l$ block; stretched vertically, with each number in a crossbar representing a $2j \times l$ block while all other numbers represent $j \times l$ blocks; stretched horizontally, with each number in a vertical bar representing a $j \times 2l$ block while all other numbers represent a $j \times l$ block; or in a variety of other proportions.

4. Further applications. We have shown how the mesh orderings of Fig. 5 can be used to make the time per iteration of the standard stationary iterative methods on a two-dimensional grid of processors independent of n without sacrificing rate of convergence. We now discuss implications for more complicated algorithms and problems. In particular, we consider further cases in which iteration time is independent of n.

A. Other connectivities and geometries. It is not critical that the mesh be equally spaced, that the region be a square, or that each variable be directly coupled to all 8 neighbors. For example, hexagonal connections, and piecewise linear finite elements over regular triangles, also fall within this scheme.

B. Nonlinear problems. The iterations of § 2 can also be applied to nonlinear systems of equations $f(u)=0$, where the Jacobian matrix of f has nine-point connectivity structure.

C. Gradient methods. These mesh orderings can also be used to simplify steepest descent or conjugate gradient algorithms for the problem

$$\min_u u^T A u - u^T b,$$

or a nonquadratic version of it, where A is positive definite and has nine-point structure. The standard algorithms require inner products over vectors of length n^2. On two-dimensional grids of processors with nearest neighbor connections only, this is an $O(n)$ process. An iteration can be derived, however, which holds two sets of variables constant while solving problems involving the third. Each iteration would take the form:

For i = 1, 2, 3

Decrease the function by changing u_i, holding the other variables fixed. ("Decrease" could mean solving the subproblem exactly or simply reducing the objective function by several iterations of a gradient method.)

This breaks the problem into three parallel sets of small problems (5 unknowns each, for the P^3 ordering) which can be solved using local communication only.

This is a descent algorithm, but does not have the finite termination property of conjugate gradients.

The mesh orderings can also be used with the standard preconditioned conjugate gradient algorithm (see, for example, [5]). In this case the preconditioning operator, iterations of the SSOR iteration, for example, could be applied in time independent of n using only local communication, although the conjugate gradient iteration would still require inner products of length n^2.

D. Constrained problems. Free boundary problems for partial differential equations can lead to minimization problems with upper and lower bounds on the variables [6]: for example,

$$\min_u u^T A u - u^T b, \qquad c \leqq u \leqq d,$$

or a similar problem with nonquadratic objective function. Iterations as in § 2 (Jacobi, Gauss–Seidel, SOR) are still applicable as long as each variable change is truncated if necessary to keep the variable within range.

Gradient methods are also often used for constrained problems. In addition to the inner products used to determine parameters, an additional global check is ordinarily necessary to calculate the maximum step which keeps the variables within

range. The orderings of § 2 can be used to produce an algorithm in which the inner products and step length checking for gradient methods are reduced to local operations. The algorithm is analogous to that in (C) above.

E. Three-dimensional problems. These methods also extend to the solution of three-dimensional problems on two-dimensional arrays of processors. For an $n \times n \times p$ grid, an iteration using the ordering schemes above crossed with a line scheme in the third dimension would produce algorithms with iteration time proportional to p on a two-dimensional grid of n^2 processors with local connections.

Acknowledgments. Seymour Parter, G. W. Stewart, and David Young have made helpful suggestions related to this work.

REFERENCES

[1] L. M. ADAMS, *Iterative algorithms for large sparse linear systems on parallel computers*, NASA Contractor Report 166027, NASA Langley Research Center, Hampton, VA, 1982.

[2] G. H. BARNES, R. M. BROWN, M. KATO, D. J. KUCK, D. L. SLOTNICK AND R. A. STOKES, *The* ILLIAC IV *computer*, IEEE Trans. Comput., C-17 (1968), pp. 746–757.

[3] SHADID H. BOKHARI, *On the mapping problem*, IEEE Trans. Comput. C-30 (1981), pp. 207–214.

[4] A. BRANDT, *Multigrid solvers on parallel computers*, in Elliptic Problem Solvers, M. Schultz, ed., Academic Press, New York, 1971.

[5] P. CONCUS, G. H. GOLUB AND D. P. O'LEARY, *A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations*, in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, New York, 1976, pp. 309–322.

[6] C. W. CRYER, P. M. FLANDERS, D. J. HUNT, S. F. REDDAWAY AND J. STANSBURY, *The solution of linear complementarity problems on an array processor*, Tech. Sum. Rep. 2170, Mathematics Research Center, Univ. Wisconsin, Madison, 1981.

[7] DENNIS GANNON, *On mapping non-uniform P.D.E. structures and algorithms onto uniform array architectures*, in Proc. Internat. Conf. on Parallel Processing, Ming T. Liu and Jerome Rothstein, eds., IEEE Computer Society, 1981.

[8] DON HELLER, *A survey of parallel algorithms in numerical linear algebra*, SIAM Rev. 20 (1978), pp. 740–777.

[9] H. JORDAN, *A special purpose architecture for finite element analysis*, Proc. 1978 Conference on Parallel Processing, IEEE Computer Society, pp. 263–266.

[10] J. S. KOWALIK, S. P. KUMAR AND R. E. LORD, *Solving linear algebraic equations on a* MIMD *computer*, Proc. International Conference on Parallel Processing, IEEE Computer Society, 1980.

[11] DAVID J. KUCK AND RICHARD A. STOKES, *The Burroughs scientific processor* (BSP), IEEE Trans. Comput. C-31 (1982), pp. 363–375.

[12] S. P. KUMAR, *Parallel algorithms for solving linear equations on* MIMD *computers*, Ph.D. thesis, Computer Science Dept., Washington State Univ., Pullman, 1982.

[13] H. T. KUNG AND C. E. LEISERSON, *Systolic arrays (for* VLSI), in Sparse Matrix Proceedings 1978, I. S. Duff and G. W. Stewart, eds., Society for Industrial and Applied Mathematics, Philadelphia, 1979, pp. 256–282.

[14] S. Y. KUNG, VLSI *array processor for signal processing*, Proc. Conference on Advanced Research in Integrated Circuits, Massachusetts Institute of Technology, Cambridge, 1980.

[15] S. Y. KUNG, R. J. GAL-EZAR, K. S. ARUN AND D. V. BHASKAR RAO, *Wavefront array processor: architecture, language, and applications*, IEEE Trans. Comput., C-31 (1982), pp. 1054–1066.

[16] A. LICHNEWSKY, *Solving some linear systems arising in finite element methods on parallel processors*, Tech. Rep., Université de Paris-Sud and INRIA, 1982.

[17] T. MOTO-OKA, ed., *Fifth Generation Computer Systems*, North-Holland, New York, 1982.

[18] SEYMOUR V. PARTER, *On estimating the "rates of convergence" of iterative methods for elliptic difference equations*, Trans. Amer. Math. Soc., 114 (1965), pp. 320–354.

[19] SEYMOUR V. PARTER AND MICHAEL STEUERWALT, *Another look at iterative methods for elliptic difference equations*, Computer Sciences Dept. Technical Report 358, Univ. of Wisconsin, Madison, 1979.

[20] ———, *Block iterative methods for elliptic and parabolic difference equations*, SIAM J. Numer. Anal., 19 (1982), pp. 1173–1196.

[21] CHUCK RIEGER, ZMOB: *Hardware from a user's viewpoint*, Proc. IEEE Computer Society Conference on Pattern Recognition and Image Processing, August, 1981, pp. 399–408.
[22] AHMED H. SAMEH, *Parallel algorithms in numerical linear algebra*, CREST Conference 1981, Bergamo, Italy, Academic Press, to be published.
[23] RICHARD S. VARGA, *Matrix Iterative Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1962.
[24] YEHUDA WALLACH AND V. KONRAD, *On block parallel methods of solving linear equations*, IEEE Trans. Comput., C-29 (1980), pp. 354–359.