NUMERICAL LINEAR ALGEBRA WITH APPLICATIONS
Numer. Linear Algebra Appl. 2003; **10**:445–465 (DOI: 10.1002/nla.323)

Effects of problem decomposition (partitioning) on the rate of convergence of parallel numerical algorithms

Jane K. Cullum[1,*], Keith Johnson[2] and Miroslav Tůma[3,4]

[1] *Los Alamos National Laboratory, Los Alamos, New Mexico 87545, U.S.A.*
[2] *The Boeing Company, P. O. Box 3707, MS 67-LX, Seattle, WA 98124, U.S.A.*
[3] *Institute of Computer Science, Academy of Sciences of Czech Republic, Pod vodárenskou věží 2, 182 07 Prague 8, Czech Republic*
[4] *TU Liberec, Hálkova 6, Liberec, Czech Republic*

SUMMARY

We focus on the interplay between the choice of partition (problem decomposition) and the corresponding *rate of convergence* of parallel numerical algorithms. Using a specific algorithm, for which the numerics depend upon the partition, we demonstrate that the rate of convergence can depend strongly on the choice of the partition. This dependence is shown to be a function of the algorithm and of the choice of problem. Information gleaned from tests using various 2-way partitions leads to new partitions for which some degree of convergence robustness is exhibited. The incorporation of a known correction for approximate Schur complements into the original algorithm yields a modified parallel algorithm which numerical experiments indicate achieves robust convergence behaviour with respect to the choice of partition. We conclude that tests of a parallel algorithm which vary the method of partitioning can provide constructive information regarding the robustness of the algorithm and guidance for modifying the algorithm or the choice of partitioning algorithm to make the overall computations more robust. Copyright © 2003 John Wiley & Sons, Ltd.

KEY WORDS: parallel algorithms; graph partitioning; problem decomposition; rate of convergence

1. INTRODUCTION

Many important applications require large complex simulations which can be executed only in parallel. Studies of the behaviour of the parallel numerical algorithms in such simulations typically select a method for problem decomposition, exercise the numerical algorithm

* Correspondence to: Jane Cullum, Los Alamos National Laboratory, Los Alamos, New Mexico 87545, U.S.A.

Contracts/grant sponsor: U.S. Department of Energy; contract/grant number: W-7405-ENG-36.
Contracts/grant sponsor: U.S. DOE Office of Science, ASCR, MICS, AMS Program; contract/grant number: KC-07-01-01.
Contract/grant sponsor: GA Czech Republic; contract/grant number: 201/02/0595/GA.
Contract/grant sponsor: GA AS Czech Republic; contract/grant number: A1030103.

Published online 7 May 2003 *Received 14 August 2001*
Copyright © 2003 John Wiley & Sons, Ltd. *Revised 30 April 2002*

over various sets of processors, and use the resulting data to answer questions about parallel efficiencies and scalability. Subsequently, the algorithm may be modified to improve these measures of performance.

We demonstrate that if the numerics of a parallel algorithm depend on the choice of the partition, then such studies should also examine the sensitivity of the *rate of convergence* of the algorithm to the choice of problem decomposition. Ideally, the convergence behaviour is invariant with respect to the choice of partition. However, for parallel algorithms for which the numerics depend on the partition, achieving uniform rates of convergence may not be feasible. In such a case, the objective may become robust behaviour, the selection of a suitable method for problem decomposition for which good convergence is achieved with limited variations in performance across the number of processors specified. Numerical studies which vary the partition can provide constructive information regarding the robustness of an algorithm and can be used to provide guidance for modifying the algorithm or the partitioning method to make the overall computations more robust.

To make the discussion concrete, we focus on one such algorithm, ParAINV [1–3]. In Section 2 we review ParAINV, which when given a system of linear equations, constructs an approximate inverse preconditioner and then applies a Krylov iterative method to the resulting preconditioned system. We also describe briefly two families of test matrices, Two-Material and Kershaw [4], which we use in our tests.

In Section 3, using partitions generated by the Metis package [5, 6] with modified matrix adjacency graphs as input to Metis, we explore the effects of the choice of Metis-generated partitions on the rate of convergence of ParAINV. For a Two-Material problem, we observe a *uniform* rate of convergence with respect to the choice of partition method, but for the Kershaw problem, the rate of convergence is *irregular* with respect to both the choice of the partition method and the specified number of processors.

In Section 4, for the Kershaw problem, we introduce new Metis-generated partitions which are obtained by using particular weighted dual graphs of the Kershaw grid as inputs to Metis. Numerical tests indicate that using particular members of this set of partitions, it is possible to achieve some degree of *robust* convergence.

In practice, we would like to have numerical algorithms which are not sensitive to the choice of partition method. In Section 5 we consider a direct modification of the ParAINV algorithm, and repeat the set of tests which were defined in Sections 3 and 4. In these tests this *corrected* ParAINV algorithm exhibits robustness with respect to the choice of partition, at the expense of slightly more arithmetic operations and storage. On *easy* problems we also observe somewhat higher iteration counts to convergence. In Section 6, we summarize the results and indicate some future directions for this work.

References [7–10] provide examples of other types of parallel computations where using a minimum edge cut partition requires significantly more time to execute than computations which use a partition which has a larger number of cut edges but for which the partition subdomains have better aspect ratios. Relationships between these references and the work in this paper are discussed in Section 4.

For some parallel algorithms, for example, domain decomposition [11], the dependence of the numerics of the algorithm on the partition is a consequence of the design of the algorithm. In other cases, approximations introduced in the transcription of a theoretically global parallel algorithm into a practical parallel numerical implementation may yield an implementation with such dependencies. Whether or not such dependencies significantly affect the convergence behaviour of a given numerical algorithm can be, as we will demonstrate, a function not only of the algorithm but also of the particular problem being solved.

2. ParAINV: APPROXIMATE INVERSE PRECONDITIONER

In our discussion and in our numerical tests we focus on a particular parallel algorithm (ParAINV) for computing approximate inverse preconditioners of real symmetric matrices [1]. The version of ParAINV which we use is based on the stabilized version of the uni-processor, approximate inverse preconditioner algorithm, SAINV [2].

Given a system of linear equations,

$$Ax = b \tag{1}$$

the ParAINV algorithm directly constructs the factors of an approximate inverse preconditioner $\tilde{P}^{-1}$ for a permuted matrix $\tilde{A}$ obtained from A, and then applies an iterative method to solve the corresponding preconditioned problem,

$$\tilde{P}^{-1}\tilde{A}\tilde{x} = \tilde{P}^{-1}\tilde{b} \tag{2}$$

Each application of the preconditioner consists of two matrix–vector multiplications. This is in contrast to an approximate inverse which is based upon an approximate factorization of $\tilde{A}$ which would require two triangular solves. Throughout this paper we assume that A is real, symmetric and positive definite.

If K denotes the number of processors to be used, the ParAINV algorithm proceeds as follows. Graph partitioning is used to divide the variables x into K disjoint subsets. Each subset is assigned to one of the K processors. Boundary variables, v_i and v_j, corresponding to non-zero entries $a_{ij} \neq 0$ in A with i and j in different partition subsets, are identified and become input to a vertex separator algorithm which attempts to determine a smaller set of separator(interface) nodes. This separator algorithm is based on the Dulmage–Mendelsohn canonical decomposition [12]. See also References [13, 14]. The resulting separator vertices are extracted from the K subsets and assigned to the master processor. The corresponding permuted matrix has the following form:

$$\tilde{A} \equiv \begin{bmatrix} A_1 & & & & B_1 \\ & A_2 & & & B_2 \\ & & & & \vdots \\ & & & A_K & B_K \\ B_1^{\mathrm{T}} & B_2^{\mathrm{T}} & \ldots & B_K^{\mathrm{T}} & A_{\mathrm{s}} \end{bmatrix} \tag{3}$$

The subscript s denotes the portion of the matrix which corresponds to the separator nodes.

Since A is real symmetric and positive definite, the A_j and A_{s} are real symmetric and positive definite. The Schur complement matrix,

$$S \equiv A_{\mathrm{s}} - \sum_{j=1}^{K} B_j^{\mathrm{T}} A_j^{-1} B_j \tag{4}$$

corresponding to the block diagonal matrix A_{D}, which is defined by the $A_j, 1 \leqslant j \leqslant K$, is also symmetric and positive definite.

Theoretically, $\tilde{A}^{-1}$ has the block factorization,

$$\tilde{A}^{-1} \equiv \begin{bmatrix} L_1^{-\mathrm{T}} & & & & E_1 \\ & L_2^{-\mathrm{T}} & & & E_2 \\ & & & & \vdots \\ & & & L_K^{-\mathrm{T}} & E_K \\ 0 & & \dots & 0 & L_S^{-\mathrm{T}} \end{bmatrix} \begin{bmatrix} L_1^{-1} & & & & 0 \\ & L_2^{-1} & & & 0 \\ & & & & \vdots \\ & & & L_K^{-1} & 0 \\ E_1^{\mathrm{T}} & E_2^{\mathrm{T}} & \dots & E_K^{\mathrm{T}} & L_S^{-1} \end{bmatrix}$$

where

$$\begin{aligned} A_j &= L_j L_j^{\mathrm{T}} \\ S &= L_S L_S^{\mathrm{T}} \\ E_j &= -L_j^{-\mathrm{T}} L_j^{-1} B_j L_S^{-\mathrm{T}} \end{aligned} \tag{5}$$

In practice, it is not feasible to compute exact factorizations of the matrices A_j, $1 \leqslant j \leqslant K$, and of S, and S is not even computable. It is feasible, however, to construct factored approximate inverses for the A_j, $1 \leqslant j \leqslant K$, and for an associated approximate Schur complement matrix, S_{sa}. These can be used to compute a corresponding factored approximate inverse preconditioner for $\tilde{A}$ [3, 1].

In ParAINV, after $\tilde{A}$ in Equation (3) is obtained, the uniprocessor approximate inverse algorithm (SAINV) [2] is applied in parallel to the matrices A_j, $1 \leqslant j \leqslant K$, to compute corresponding factored approximate inverses, $Z_j D_j^{-1} Z_j^{\mathrm{T}}$. By construction, each D_j is diagonal and positive definite, and each Z_j is upper triangular.

The computed approximate inverses are combined to form an approximate Schur complement matrix,

$$S_{\mathrm{sa}} \equiv A_{\mathrm{s}} - \sum_{j=1}^{K} B_j^{\mathrm{T}} Z_j D_j^{-1} Z_j^{\mathrm{T}} B_j \tag{6}$$

SAINV is applied to S_{sa} to obtain a corresponding approximate inverse $Z_{\mathrm{sa}} D_{\mathrm{sa}}^{-1} Z_{\mathrm{sa}}^{\mathrm{T}}$. Since the SAINV implementation maintains positive entries in D_{sa}, the computed approximate inverse for S_{sa} is always positive definite. Therefore, the diagonal matrices D_j, $1 \leqslant j \leqslant K$, and D_{sa} can be absorbed into the respective Z_j, Z_{sa} matrices. The resulting $\tilde{Z}_j$, $1 \leqslant j \leqslant K$, and $\tilde{Z}_{\mathrm{sa}}$ factors are combined to obtain the upper triangular factor $\tilde{Z}$ of an approximate inverse for $\tilde{A}$.

$$\tilde{Z} \equiv \begin{bmatrix} \tilde{Z}_1 & & & & \hat{E}_1 \\ & \tilde{Z}_2 & & & \hat{E}_2 \\ & & & & \vdots \\ & & & \tilde{Z}_K & \hat{E}_K \\ 0 & 0 & \dots & 0 & \tilde{Z}_{\mathrm{sa}} \end{bmatrix}$$

where

$$\hat{E}_j \equiv -\tilde{Z}_j \tilde{Z}_j^{\mathrm{T}} B_j \tilde{Z}_{\mathrm{sa}} \tag{7}$$

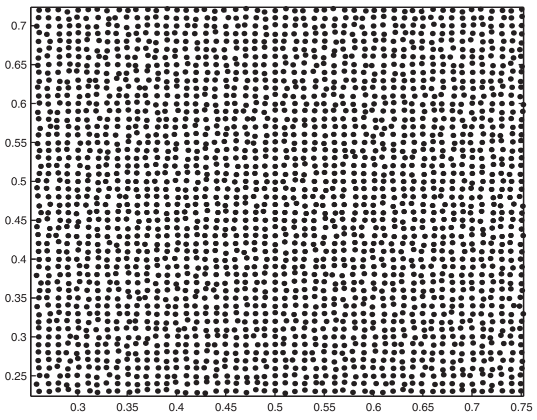
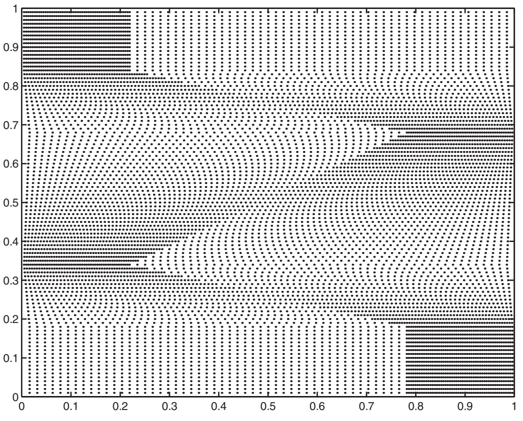

Figure 1. LEFT: enlargement of section of typical grid for a Two-Material problem: RIGHT: typical grid for a Kershaw problem.

The corresponding positive definite approximate inverse preconditioner, $\tilde{P}^{-1}$ for $\tilde{A}$, is defined as

$$\tilde{P}^{-1} \equiv \tilde{Z}\tilde{Z}^{\mathrm{T}} \tag{8}$$

The corresponding system of equations, Equation (2), is solved by applying an iterative method. For additional details on ParAINV and on the SAINV algorithm see References [1, 2].

2.1. Test matrices

In the tests presented in this paper, we used representative matrices belonging to two families of diffusion problems, Two-Material and Kershaw [4]. Each member of each family is real symmetric and positive definite. Both families use Support Operator discretizations [15, 16]. In two-dimensions the underlying grid consists of quadrilaterals defined on a unit square.

The grid in a Two-Material problem is obtained by applying small random perturbations to an orthogonal grid. The grid in a two-dimensional Kershaw problem is a Kershaw grid [17], obtained by applying a particular skewing function to one direction of an orthogonal grid [4]. See Figure 1. In a Two-Material problem, the diffusion coefficients incur discontinuities across the logical vertical centreline of the perturbed orthogonal grid. For additional details, see Reference [4].

3. EFFECTS OF CHOICE OF PARTITION

The application of any parallel algorithm requires the decomposition of the given problem across the specified number of processors. Our focus is on the interplay between the partitioning (problem decomposition) and the rate of convergence of parallel numerical algorithms.

In this section, using the ParAINV algorithm, we illustrate how the rate of convergence of a parallel numerical algorithm can depend strongly on the problem decomposition. Our emphasis in these experiments is on this *partitioning phenomenon* and what to do about it. The illustrated phenomenon is not specific to the algorithm used in these tests.

Ideally, the rate of convergence of a parallel numerical algorithm is not a function of the particular method for problem decomposition which is used to place the problem across the parallel nodes or of the number of nodes used. Also, ideally, the problem decomposition takes into account the relevant but typically numerous, not well-defined, and often in conflict, partitioning objectives, such as minimize the complexity of the communication, acknowledge the hierarchical nature of memory and communication, balance the computational load, and address potential difficulties with geometric, algorithmic, or solution features.

In practice, existing popular partitioning packages, (e.g. Chaco [18], Metis [5, 6]) accept a single abstract undirected graph $G \equiv (V, E)$ as input and use surrogate measures of one or more of the actual partitioning objectives combined with a variety of graph partitioning heuristics to construct a partition. V denotes the set of graph vertices, $V = \{v_1 \dots v_m\}$, and E denotes the set of edges, $\{(v_i, v_j)\}$, in the graph G. The most popular surrogate is the number of edges which are *cut* by the partition. An edge (v_i, v_j) is *cut* by the partition if v_i and v_j belong to different subsets of the partition. The number of edges cut by a partition is often used as an estimate of the communication costs corresponding to that partition.

It is well-known that there is a need for new graph and hypergraph abstractions and partitioning algorithms with the capability of incorporating problem and computer constraints [19, 20]. Reference [21] provides an interesting commentary regarding inadequacies and deficiencies of existing partitioning algorithms. Reference [22] is a related paper which discusses some of the issues connected with partitioning for load balancing. Reference [23] provides a survey of partitioning algorithms used in the VLSI community. Many good partitioning heuristics have come from the VLSI community.

By construction, for any partition, the matrices $A_j, 1 \leqslant j \leqslant K$, A_s, and S in Equations (3) and (4) are symmetric and positive definite. However, unless A is a M-matrix, there is no guarantee that the corresponding approximate Schur complement matrix, S_{sa}, is positive definite. The ParAINV algorithm is implemented to force the computed approximate inverse preconditioner, $\tilde{P}^{-1}$, to be positive definite. However, if S_{sa} is not positive definite, then $\tilde{P}^{-1}$ may not be a very good preconditioner for $\tilde{A}$. Whether or not S_{sa} is positive definite, is a function of the choice of partition.

3.1. *Metis algorithms, matrices used*

All of the experiments described in this paper utilized version 4.0 of the Metis partitioning software package [5, 6]. Five of the partitioning algorithms available in Metis version 4.0 were exercised. Each of these five algorithms is based on multi-level (ML) graph partitioning [24–28]. In each of them, matching algorithms are applied to recursively coarsen the input graph, and a graph partitioning algorithm is applied to the coarsest graph generated. Metis allows the imposition of integer weights on either or both of the vertices and the edges of the input graph. The resulting partition on the coarsest level is expanded and refined at each successively higher level to obtain a partition of the original graph.

Each of these algorithms is based on either recursive bisection (RB) or K-way (KWay) partitioning. In each of the tests presented, the matching algorithms used sorted, heavy edge matching. These Metis algorithms are listed by acronym and by full name in Table I and are assigned corresponding numbers. For example, the multi-level recursive bisection algorithm, MLRB, is also referred to as Algorithm 1.

Table I. Metis algorithms used.

Alg	Acronym	Full name
1	MLRB	MultiLevel Recursive Bisection
2	MLKWay	MultiLevel *K*-Way
3	MLVKWay	MultiLevel Total Volume *K*-Way
4	mcMLRB	MultiConstraint, MultiLevel Recursive Bisection
5	mcMLKWay	MultiConstraint, MultiLevel *K*-Way

The use of V in the acronym for Algorithm 3 denotes the use of the following estimate of the total communication volume as the partitioning objectives

$$\sum_{v_i \in \text{interface}} w_i Nadj[v_i] \tag{9}$$

In Equation (9), $Nadj[v_i]$ denotes the number of processors, other than the one to which v_i is assigned, which require v_i. Each w_i denotes a weight which the user can specify for each v_i.

The partitioning objective in Algorithms 1, 2, 4, 5 is to minimize the sum of the weights over the edges which are cut by the partition,

$$\sum_{(v_i, v_j) = \text{cut edge}} w_{ij} \tag{10}$$

The use of *mc* in the acronyms for Algorithms 4 and 5 in Table I denotes the fact that these algorithms require multiple weights on the vertices. In addition, Algorithm 5 explicitly allows vertex weight imbalances in the resulting subsets of the partition. In the tests presented, we set the Metis imbalance parameter to 1.03. The algorithmic options used, ordered from Algorithm 1 to 5, were (3110, 3110, 3130, 3210, 3210). With the exception of Algorithm 3, these were the specified Metis default options. Algorithms 1 and 2 correspond, respectively, to *pmetis* and *kmetis* in earlier versions of the Metis software. For additional details on these algorithms and the corresponding options see [26–28].

Two representative test matrices, 2M-SO-2D-100 and K2-SO-2D-100 from the Two-Material and Kershaw diffusion families, were used in the experiments presented in this paper. In each of these acronyms, SO denotes support operator discretization, 2D indicates that the matrix comes from a two-dimensional problem, and 100 denotes the number of subdivisions in the grid along each axis.

In each test the iterative method used was preconditioned conjugate gradients (PCG) [29], preconditioned with $\tilde{P}^{-1}$ defined as in Equation (8). The maximum number of iterations allowed was 999. Convergence was said to have occurred at iteration J, if J was the smallest integer j for which the normalized residual norm satisfied

$$\|\tilde{r}_j\| / \|\tilde{r}_0\| \leqslant 10^{-8} \tag{11}$$

$\tilde{r}_j \equiv -\tilde{A}\tilde{x}_j + \tilde{b}$ with $\tilde{x}_j$ the jth iterate. The starting guess in each test, $\tilde{x}_0$, was the zero vector so that $\tilde{r}_0 \equiv \tilde{b}$.

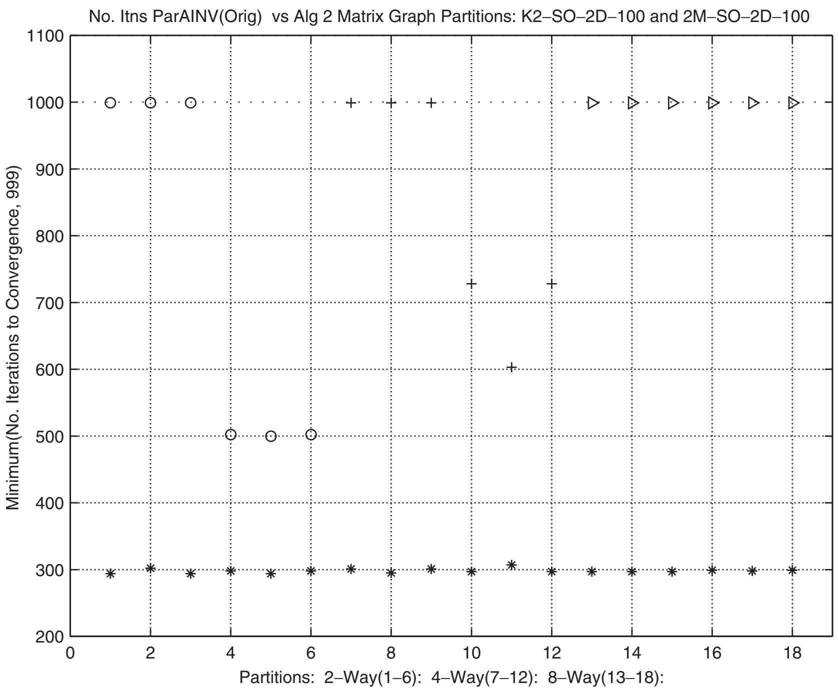

Figure 2. ParAINV(Orig): Matrices K2-SO-2D-100 and 2M-SO-2D-100: Number of iterations versus choice of matrix graph partition generated using Metis with Algorithm 2. Counts for K2-SO-2D-100: 2-Way(o): 4-Way(+): 8-Way(▷): for 2M-SO-2D-100: K-Way(*).

3.2. Matrix graph partitions

Metis requires the user to provide an abstract graph formulation of the given problem as input to the Metis software. In the original implementation of ParAINV that input graph is the adjacency graph of the matrix A in Equation (1). This graph consists of a vertex for each component of x and an edge for each non-zero entry in A.

The experiments described in this section utilized variants of the adjacency graph of the matrix A as input to Metis. We refer to any one of these variants as a matrix graph for A. For a matrix graph, the graph edge weights, when used, were set equal to the integer parts of the following weights

$$w_{ij} \equiv \max\{1, 1000. * |a_{ij}|\} \tag{12}$$

on the edges (v_i, v_j).

Figures 2 and 3 summarize the results of applying ParAINV to problems 2M-SO-2D-100 and K2-SO-2D-100 using eighteen 2-way, 4-way, and 8-way Metis-generated partitions. These partitions were obtained using Metis Algorithm 2 with corresponding matrix graphs defined with various combination of vertex and edge weights. The symbol $*$ corresponds to 2M-SO-2D-100 for any $K=2,4,8$. The symbols $\text{o}, +, \triangleright$ correspond respectively to $K=2,4,8$ for K2-SO-2D-100.

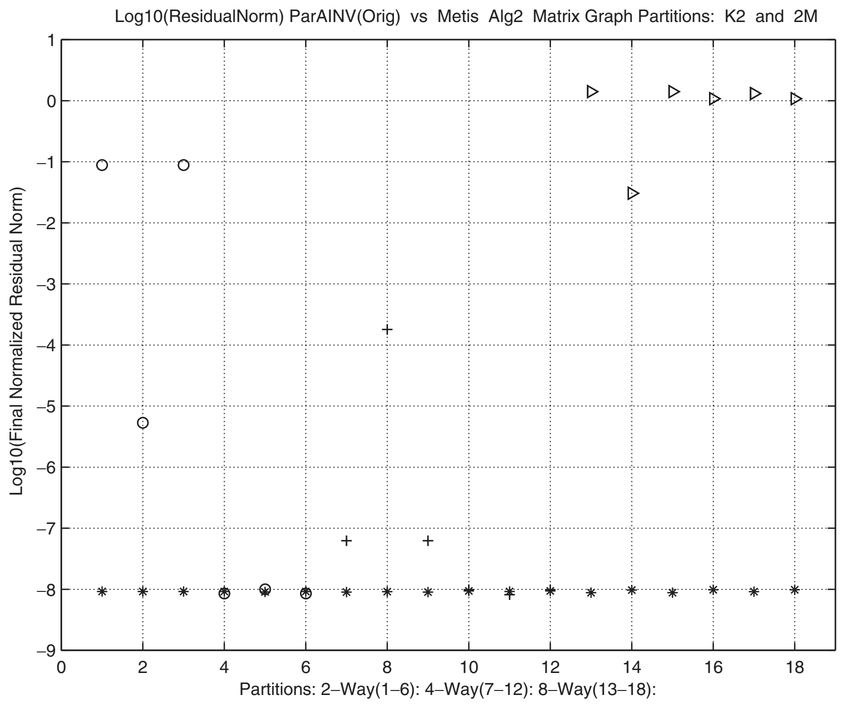

Figure 3. ParAINV(Orig): Matrices K2-SO-2D-100 and 2M-SO-2D-100: Log10(Final normalized residual norm) versus choice of matrix graph partitions generated using Metis algorithm 2. In the Figure, K2-SO-2D-100: 2-Way(o): 4-Way(+): 8-Way(▷): 2M-SO-2D-100: K-Way(*).

For each value of $K=2,4,8$, the iteration counts in Figure 2 correspond to the same sequence of various combinations of vertex and edge weights. For each K, the first three tests correspond to matrix graphs without non-trivial edge weights. The last three tests correspond to matrix graphs with non-trivial edge weights.

In these tests, to within small variations, for problem 2M-SO-2D-100, the number of iterations to convergence is independent of the choice of the partition and the number of processors $K=2,4,8$. However, for problem K2-SO-2D-100, the rate of convergence is far from uniform. No convergence was observed in any of the tests which used partitions which were generated using matrix graphs without non-trivial edge weights. In tests with non-trivial edge weights, as defined in Equation (12), the rate of convergence deteriorated as the number of processors K was increased, with no convergence observed for any of the corresponding 8-way partitions.

Figure 3 indicates that if the iterations for problem K2-SO-2D-100 had been allowed to continue, convergence would also have been achieved for runs 2, 7, and 9. These simple tests demonstrate clearly that for some problems, the convergence behaviour of a numerical algorithm may depend markedly on the number of processors specified and on the choice of problem decomposition.

Is this behaviour peculiar to partitions generated using Metis Algorithm 2? Figures 4 and 5 summarize the results of similar tests on K2-SO-2D-100 using 57 different matrix graph

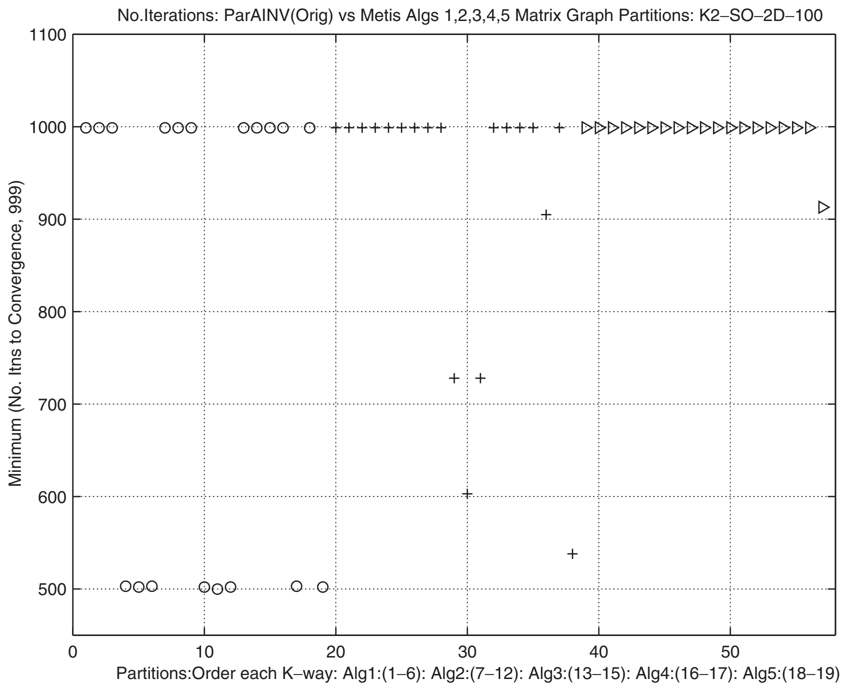

Figure 4. ParAINV(Orig): Matrix K2-SO-2D-100: Number of iterations versus choice of matrix graph partition generated using Metis algorithms $1, 2, 3, 4, 5$. In the Figure, 2-Way(o): 4-Way(+): 8-Way(▷). Results within each K-way ordered by Alg1(1–6): Alg2(7–12): Alg3:(13–15): Alg4:(16–17):Alg5:(18–19).

partitions. These partitions were generated for K-ways 2, 4, and 8, using Metis algorithms $1, 2, 3, 4$ and 5 and with and without non-trivial vertex and edge weights.

Figures 4 and 5 illustrate clearly that the answer to that question is no. Convergence within 999 iterations was achieved on eight of the nineteen 2-way partitions, on five of the nineteen 4-way partitions, and on only one of the nineteen 8-way partitions. Figure 5 indicates that convergence would have been achieved in several other tests if the iterations had been continued beyond 999.

We can conclude from these tests that for problem K2-SO-2D-100, the naive use of Metis partitions generated using a matrix graph of K2-SO-2D-100 as an abstraction for the problem may lead to poor or even no convergence. To achieve convergence, it is necessary (but not sufficient) to use non-trivial edge weights in the matrix graph input to Metis. The best way to select these weights is an open question.

4. NEW PARTITIONS

Can suitable partitions for problem K2-SO-2D-100, in the sense of some degree of robust convergence, be obtained by exercising Metis algorithms on a different abstract graph model of the K2-SO-2D-100 problem?

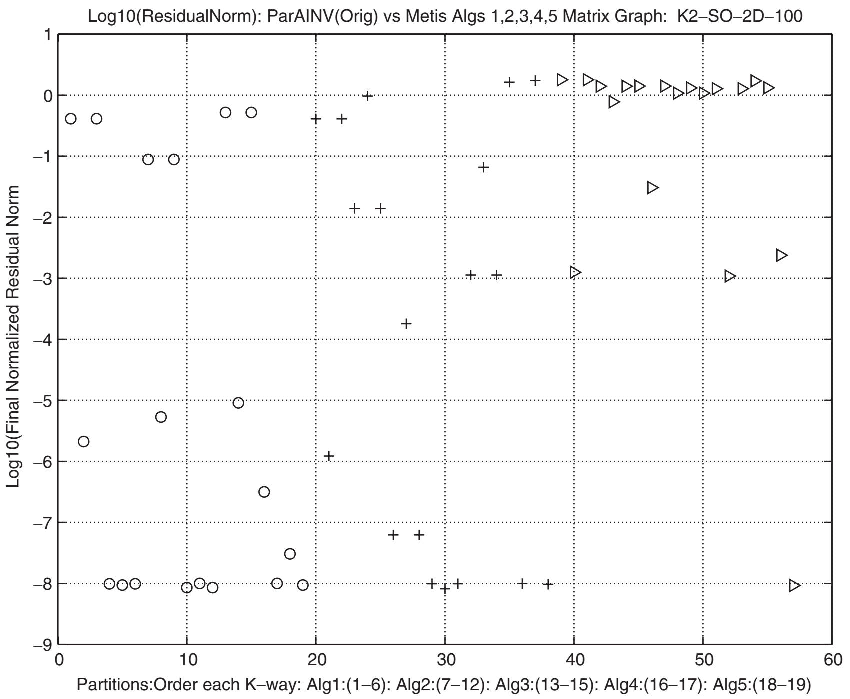

Figure 5. ParAINV(Orig): Matrix K2-SO-2D-100: Log10(Final normalized residual norm) versus choice of matrix graph partitions generated using Metis algorithms 1, 2, 3, 4, 5. In the Figure: 2-Way(o): 4-Way(+): 8-Way(▷). Results within each K-way ordered by Alg1(1–6): Alg2(7–12): Alg3:(13–15): Alg4:(16–17): Alg5:(18–19).

Earlier preliminary work by the second author, using hand-generated 2-way partitions, demonstrated that partitions which cut through highly distorted parts of the Kershaw grid adversely affect the rate of convergence of ParAINV. Thus, the expectation is that the non-uniform convergence observed for problem K2-SO-2D-100 is connected to the distortions in the Kershaw grid. Therefore, we focus on an abstract graph associated with the grid. We use the following definition.

Definition 4.1
Given a grid G containing nc elements(cells), the dual grid graph of G is the graph $\mathscr{G}_D \equiv \{V_D, E_D\}$ consisting of vertices c_j, $j=1,nc$, for each element (cell) in the grid and of edges (c_i, c_j) for each interior face in the grid.

If no weights are specified, the dual grid graphs for problems 2M-SO-2D-100 and K2-SO-2D-100 are identical. (This is also true of the matrix graphs for these two problems). To distinguish between these two problems, non-trivial edge weights must be specified.

With the exception of Algorithm 3, the Metis algorithms considered attempt to minimize the criteria in (10). We can change the partition generated by assigning different, well-chosen weights to the edges of the dual grid graph. If the weights are chosen to reflect the distortions in the grid, then the sorted heavy matching algorithms used to coarsen the graph may combine with the minimization of the objective in (10) to yield *good* partitions. The guess is that *good* partitions will correspond to splitting the dual grid graph so as to avoid separator nodes (cells) which correspond to highly distorted elements in the Kershaw grid.

Therefore, we want edge weights which are largest on the most distorted parts of the Kershaw grid. We borrow an idea from the finite element community. References [7, 8, 30, 10] demonstrate the adverse effects on the convergence of domain decomposition preconditioned conjugate gradient computations when the problem decomposition used corresponds to geometric subdomains which have poor aspect ratios.

References [8, 10] propose partitioning algorithms with the objective of minimizing the average aspect ratio of the geometric sudomains resulting from a partition of the dual grid graph. They present results of numerical tests to justify the use of this objective, tests where the computational time required using their partitions is significantly smaller than the time required using a minimum edge cut partition. In their partitioning algorithms they use the aspect ratios of the elements as weights on the *vertices* of the dual grid graph.

Minimizing the aspect ratio of the resulting geometric domains is computationally intensive, requiring the repeated computation of subdomain aspect ratios as the partitioning algorithm proceeds. The authors of [7, 8, 30, 10] have proposed various surrogates which indirectly control the aspect ratios of the subdomains and for which the amount of computation required is decreased, but the computations required are still significant.

Our approach is quite different. We attempt to determine *good* partitions by utilizing the Metis algorithms directly with abstract graph models which adequately reflect the distorted grid. We do not try to control the average of the aspect ratios of the geometric subsets generated by the partitions. We localize the aspect ratio computations and use them to define *edge* weights.

Instead of assigning weights to the vertices (cells) and directly changing the partitioning algorithm, we introduce new edge weights for the dual grid graph which provide estimates of the local distortion of the grid. Using non-trivial edge weights is a de facto modification of the partitioning objective. See Equation (10). By definition, each edge (c_i, c_j) in the dual grid graph corresponds to an interior face in the Kershaw grid. The aspect ratios of these two cells can be used to define an edge weight for this edge.

Reference [10] lists several formulas for defining aspect ratios. We use the following definition which Reference [10] indicates is robust.

$$\mathscr{A}\mathscr{R}(\Omega) \equiv [C(\Omega)]^2/16A(\Omega) \tag{13}$$

where Ω represents a connected domain, $C(\Omega)$ is the circumference of that domain, and $A(\Omega)$ is the area of that domain.

The abstract graph input to Metis must be undirected, so the edge weights must be defined symmetrically. Edge (c_i, c_j) must have the same weight as edge (c_j, c_i). We make the following definition.

$$w_{ij} \equiv [\max\{\mathscr{A}\mathscr{R}(\text{cell}_i), \mathscr{A}\mathscr{R}(\text{cell}_j)\}]^2 \tag{14}$$

These weights give some measure of the local grid distortion and are used as edge weights for the dual grid graph. For problem K2-SO-2D-100, we plot these edge weights in Figure 6. The weights in Figure 6 are ordered linearly using the natural ordering of the cells in the Kershaw grid. A comparison of Figure 6 with Figure 1 confirms that the largest edge weights correspond to the most distorted parts of the Kershaw grid.

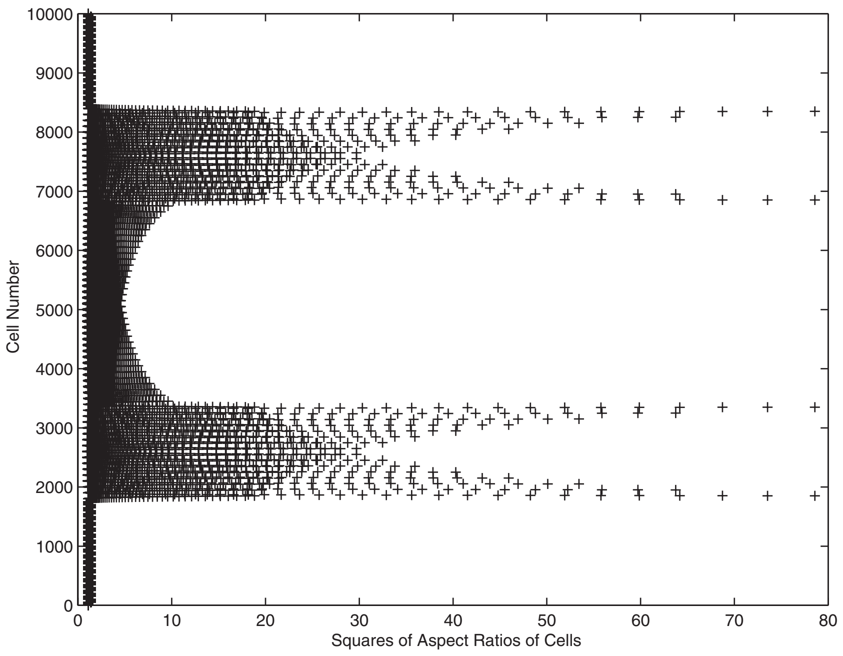

Figure 6. Squares of aspect ratios for cells in K2-SO-2D-100 grid versus cell number in natural ordering.

We reran the set of tests discussed in Section 3 using partitions generated from Metis with the dual grid graph (instead of the matrix graph). The results of those tests are summarized in Figures 7 and 8. Comparing Figure 7 to Figure 4, we observe significant differences in performance across the corresponding sets of tests. For 2-way(o) and 4-way(+), all of the ParAINV tests which used partitions which were generated using the dual grid graph with the edge weights defined in (14) converged. For 8-way($\triangleright$), for Algorithms 1, 2, and 5, the ParAINV tests which used partitions which were generated using the dual grid graph with those edge weights, converged within 999 iterations or shortly thereafter. See Figure 8. These tests exhibited a robustness with respect to convergence across the choice of partitions which were generated using the new edge weights.

In these tests, Metis-generated partitions using Algorithm 1 with vertex weights set equal to the degree of the vertex and edge weights defined by (14), exhibited a robustness with respect to iteration counts to convergence across the 2, 4, and 8 way partitions, with corresponding iteration counts of (501, 502, 524). Thus, if ParAINV is used on problem K2-SO-2D-100 with partitions which are generated using the dual grid graph, Algorithm 1 with vertex weights set equal to the vertex degree, and with edge weights as specified in Equation (14), the desired robustness across 2, 4, 8 way partitions is achieved. For 8-way partitions, there is not, however, robustness in iteration count across partitions generated using the other Metis algorithms. In Table II, n.c. indicates no convergence exhibited.

It is interesting to look at the 8-way partitions used in these tests. Do we observe any correlations between *good* convergence behaviour and partitions which avoid cuts through the

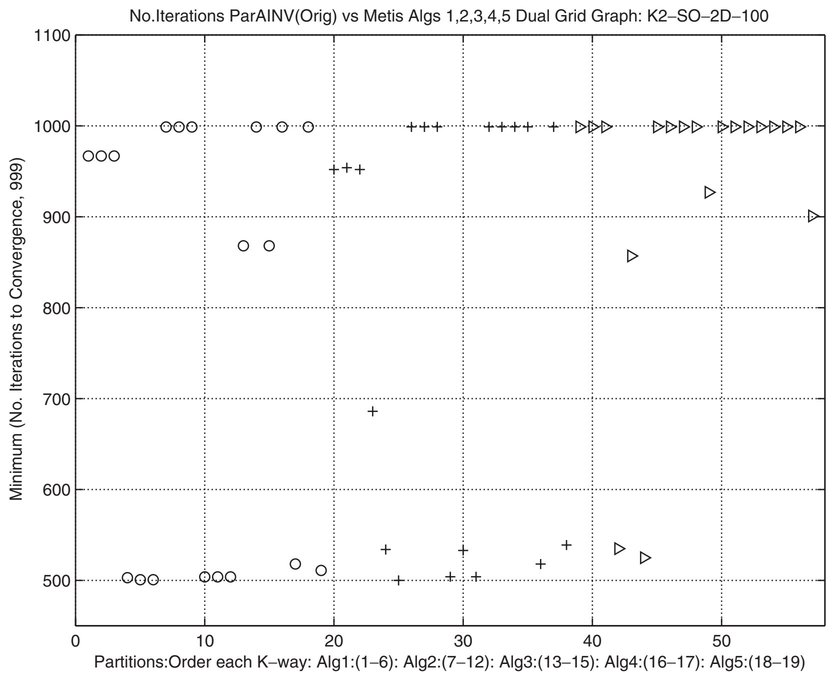

Figure 7. ParAINV(Orig): K2-SO-2D-100: Number of iterations versus choice of dual grid graph partition generated using Metis algorithms 1, 2, 3, 4, 5. In the Figure: 2-Way(o): 4-Way(+): 8-Way(▷). Within each K-way set of tests results are ordered Alg1(1–6): Alg2(7–12): Alg3:(13–15): Alg4:(16–17): Alg5:(18–19).

highly distorted parts of the Kershaw grid? Are there correlations between *bad* convergence and partitions which cut along or through highly distorted parts of the Kershaw grid?

Plate 1 corresponds to test number 44 in Figure 7 where *good* convergence was achieved. The number of iterations required to achieve convergence was 524. Plate 2 corresponds to test number 57 in Figure 4 where convergence was achieved but the convergence was very poor. The number of iterations required to achieve convergence was 913. Plate 3 corresponds to test number 39 in Figure 7 where convergence was not achieved. At the end of 999 iterations the normalized residual norm was larger than 1.0. Each of these Plates 1, 2 and 3, is a plot of the geometric locations of the variables by partition subset.

If we compare the 8-way partition in Plate 1 with the Kershaw grid in Figure 1, we observe that the most highly distorted sections of that grid are encapsulated within two of the partitions subsets. This *good* partition was generated using the dual grid graph and Algorithm 1 with vertex weights equal to the vertex degree and the edge weights defined in Equation (14) which provide excellent measures of the local distortions in the Kershaw grid.

If we compare the 8-way matrix partition in Plate 2 with the Kershaw grid in Figure 1, we observe that much of the upper part of the highly distorted parts of the grid is encapsulated but that the partition slices diagonally through the corresponding lower part. This *poor* partition was generated using the K2-SO-2D-100 matrix graph and Algorithm 5 with degree and unit vertex weights, the edge weights defined in Equation (12), and setting the Metis imbalance

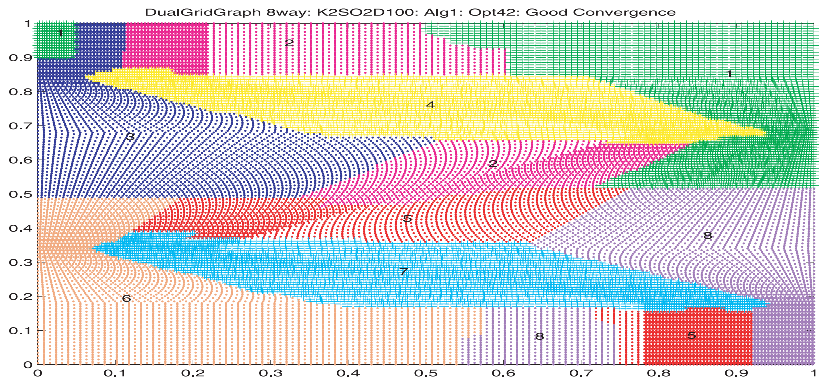

Plate 1. *Good convergence*: 8-way dual grid graph partition for K2-SO-2D-100: Generated using Metis algorithm 1 with vertex degree as vertex weights and with edge weights defined as in (14).

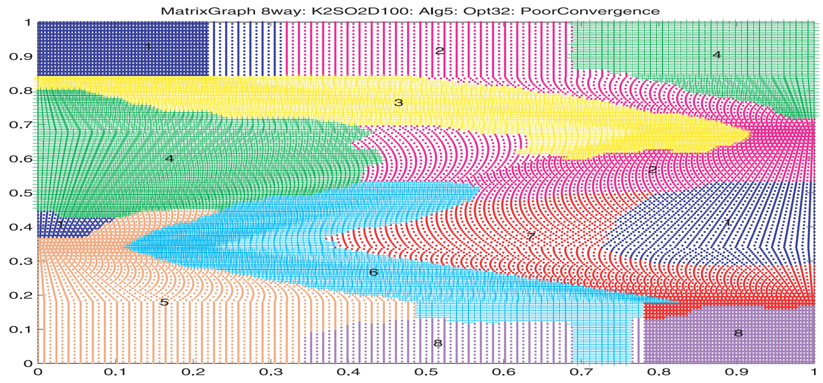

Plate 2. *Poor convergence*: 8-way matrix graph partition for K2-SO-2D-100: Generated using Metis algorithm 5 with vertex degree as vertex weights and with edge weights as defined in (12) and with an imbalance of 1.03.

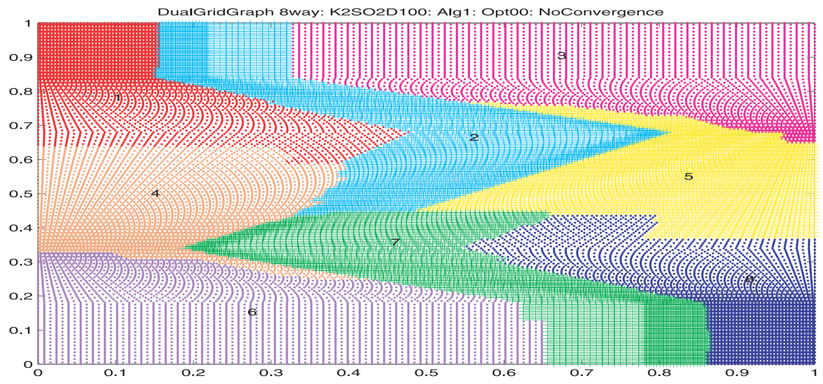

Plate 3. *No convergence*: 8-way dual grid graph partition for K2-SO-2D-100: Generated using Metis algorithm 2 with no non-trivial vertex or edge weights.

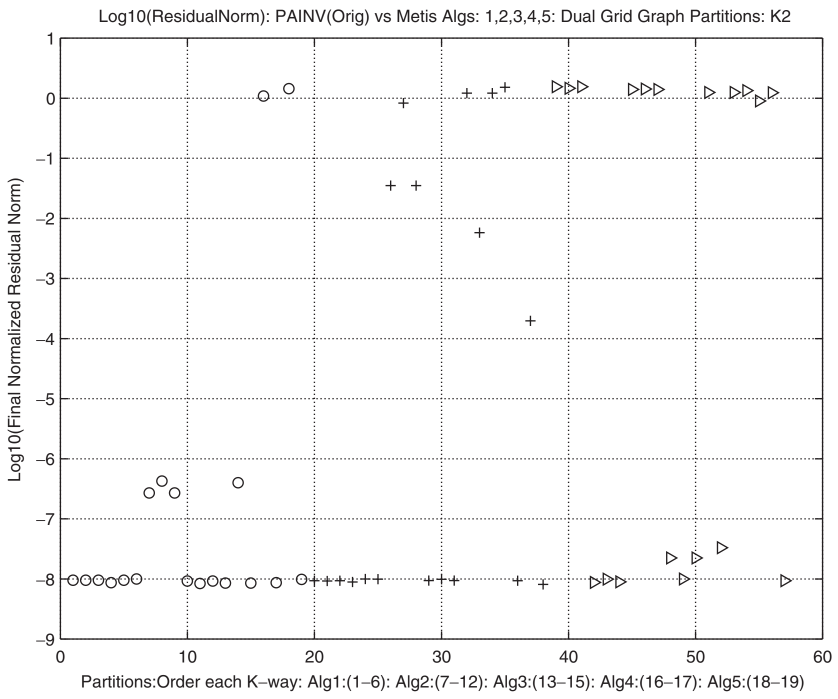

Figure 8. ParAINV(Orig): Matrix K2-SO-2D-100: Log10(Final normalized residual norm) versus choice of dual grid graph partitions generated using Metis algorithms 1, 2, 3, 4, 5. In the Figure: 2-Way(o): 4-Way(+): 8-Way(▷). Results within each K-way are ordered by Alg1(1–6): Alg2(7–12): Alg3:(13–15): Alg4:(16–17): Alg5:(18–19).

Table II. ParAINV(Orig): Using edge-weighted dual grid graph partitions.

Algorithm	Min itns 2-way	Min itns 4-way	Min itns 8-way
1	501	500	524
2	504	504	927
4	518	518	n.c.
5	511	539	901

parameter to 1.03. For the matrix graph tests on K2-SO-2D-100 this was the only 8-way partition where convergence was achieved.

If we compare the 8-way partition in Plate 3 with the Kershaw grid in Figure 1, we observe that this partition slices diagonally through the upper and the lower most highly distorted parts of the grid in many places. This *bad* partition was generated using the dual grid graph and Algorithm 1 without non-trivial vertex or edge weights to provide information about the distortions in the grid.

In the tests on problem 2M-SO-2D-100, to within small variations, the rate of convergence was independent of the choice of the partition. This observed behaviour provides additional evidence that the difficulties with K2-SO-2D-100 are caused by the distortions in the grid.

5. A ROBUST ParAINV ALGORITHM

Ideally, the convergence behaviour of a parallel algorithm is robust with respect to the choice of partition and number of processors. The incorporation of a known approximate Schur complement correction [31, 32] into the ParAINV algorithm yields a modified ParAINV algorithm, ParAINV(wCorr) which in our experiments exhibits robustness with respect to both choice of partition and number of processors.

As already noted in Section 2, if A is not a M-matrix, then the approximate Schur complement, S_{sa}, which is generated in the ParAINV algorithm may not be positive definite. If S_{sa} is not positive definite, then undesirable approximations can be introduced in the application of the SAINV algorithm to S_{sa} to force the resulting approximate inverse for S_{sa} to be positive definite. These approximations which may not be representative of the original problem, become part of the preconditioner for $\tilde{A}$ and can lead to slow convergence of the iterative method in ParAINV. We can guarantee positive definiteness of the computed approximate Schur complement if we modify it, using the modification proposed in Reference [31, 32].

For the ParAINV algorithm, S_{sa} and the corresponding correction, C_{sa} can be written as

$$\begin{aligned} S_{\mathrm{sa}} &= A_{\mathrm{s}} - B^{\mathrm{T}}M \\ C_{\mathrm{sa}} &\equiv M^{\mathrm{T}}A_{\mathrm{D}}M - M^{\mathrm{T}}B \end{aligned} \tag{15}$$

where

$$B \equiv \begin{bmatrix} B_1 \\ \vdots \\ B_K \end{bmatrix}, \quad M \equiv \begin{bmatrix} \tilde{Z}_1\tilde{Z}_1^{\mathrm{T}}B_1 \\ \vdots \\ \tilde{Z}_K\tilde{Z}_K^{\mathrm{T}}B_K \end{bmatrix} \tag{16}$$

This modification was added to ParAINV and the tests on problem K2-SO-2D-100 which were described in Sections 3 and 4 were rerun using this modified algorithm. Figure 9 contains the results of those tests. In these tests we observe a robustness in iteration count across partitions generated with and without weights, generated using either the matrix or the dual grid graph models of the K2-SO-2D-100 problem, and across K-ways for K = 2, 4 and 8. For the tests which used partitions generated using the dual grid graph the overall variations in iteration count are less than 10%.

In Figure 9, $*$ is used for partitions corresponding to the matrix graph for K2-SO-2D-100 and $\circ, +, \triangleright$ are used for partitions corresponding to the dual grid graph for K2-SO-2D-100.

Similar tests were run for problem 2M-SO-2D-100 using 57 partitions generated from the matrix graph for 2M-SO-2D-100, and Metis Algorithms 1, 2, 3, 4, 5, with and without non-trivial vertex and edge weights for K-way partitions with K = 2, 4, 8. Both the original ParAINV algorithm, ParAINV(Orig), and the corrected ParAINV algorithm, ParAINV(wCorr), were applied to problem 2M-SO-2D-100 using these partitions. The results of these tests are summarized in Figure 10. In Figure 10 $*$ corresponds to runs which used ParAINV(wCorr), and $\circ +, \triangleright$ correspond to runs which used ParAINV(Orig). The y-axis scale in Figure 10 is different from the scale used in Figure 9 for problem K2-SO-2D-100.

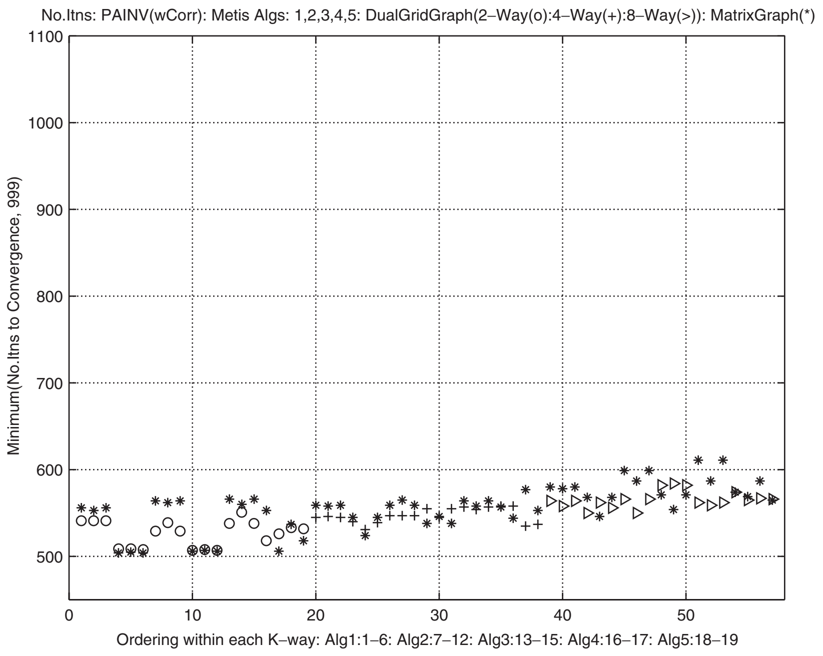

Figure 9. ParAINV(wCorr): Number of iterations for K2-SO-2D-100 versus matrix graph partitions (*) and dual grid graph partitions 2-Way(o): 4-Way(+): 8-Way(▷).

In these tests we observed *uniform* rate of convergence using both the original and the corrected versions of ParAINV. See Figure 10. It is also clear from Figure 10 that using the corrected version, ParAINV(wCorr), on 2M-SO-2D-100 required a few more iterations regardless of which partition was used in the computations. However, a small increase in the number of iterations on problems where ParAINV(Orig) converges well, is a small price for achieving robust convergence on difficult problems.

The positive definiteness of the corrected approximate Schur complement can be established by verifying that the corrected approximate Schur complement is a Galerkin projection, $R^{\mathrm{T}}\tilde{A}R$, of the positive definite matrix $\tilde{A}$ where R has full rank. It is not an ad hoc correction. It is the representation of $\tilde{A}$ on a particular subspace.

Theorem 5.1

Let $\tilde{A}$ be any real symmetric positive definite matrix defined as in Equation (3). Let f, s denote respectively, the number of rows in the diagonal block matrix A_{D} defined by the set of matrices A_j, $j=1,\ldots,K$, and the number of rows in A_{s}. Let S_{sa} be any approximate Schur complement matrix of the form

$$A_{\mathrm{s}} - B^{\mathrm{T}}M$$

where B, M are of dimension $f \times s$. Define C_{sa} by Equation (15). The corrected approximation

$$S_{\mathrm{sa}}^{c} \equiv S_{\mathrm{sa}} + C_{\mathrm{sa}}$$

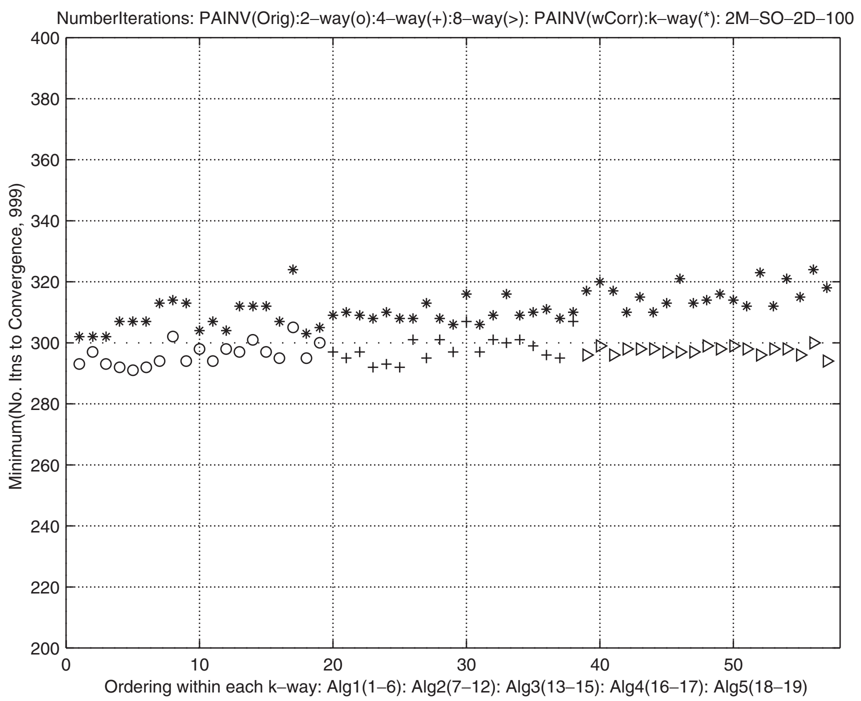

Figure 10. ParAINV: Number of iterations for 2M-SO-2D-100 versus matrix graph partitions for ParAINV(wCorr): (*) and ParAINV(Orig): 2-Way(o): 4-Way(+): 8-Way(▷).

is the Galerkin Projection $R^{\mathrm{T}}\tilde{A}R$ with

$$R \equiv \begin{bmatrix} M \\ -I_{\mathrm{s}} \end{bmatrix}$$

The error, E_{s}, in the Galerkin projection matrix when it is considered as an approximation to the true Schur complement matrix, S, is given by

$$\begin{aligned} E_{\mathrm{s}} &\equiv R^{\mathrm{T}}\tilde{A}R - S \\ &= (M - A_{\mathrm{D}}^{-1}B)^{\mathrm{T}} A_{\mathrm{D}} (M - A_{\mathrm{D}}^{-1}B) \end{aligned} \tag{17}$$

Equation (17) can be rewritten as

$$E_{\mathrm{s}} = (A_{\mathrm{D}}M - B)^{\mathrm{T}} A_{\mathrm{D}}^{-1} (A_{\mathrm{D}}M - B) \tag{18}$$

Proof

$$R^{\mathrm{T}}\tilde{A}R \equiv [M^{\mathrm{T}} \quad -I_{\mathrm{s}}] \begin{bmatrix} A_{\mathrm{D}} & B \\ B^{\mathrm{T}} & A_{\mathrm{s}} \end{bmatrix} \begin{bmatrix} M \\ -I_{\mathrm{s}} \end{bmatrix} \tag{19}$$

$$= A_s + M^T A_D M - M^T B - B^T M \tag{20}$$

But,

$$(M - A_D^{-1}B)^T A_D (M - A_D^{-1}B)$$
$$= M^T A_D M - M^T B - B^T M + B^T A_D^{-1} B \tag{21}$$

Therefore, since $S \equiv A_s - B^T A_D^{-1} B$,

$$R^T \tilde{A} R = S + (M - A_D^{-1}B)^T A_D (M - A_D^{-1}B) \tag{22}$$

Simple manipulations of Equation (21) yield Equation (18).

The quantity $(A_D M - B)$ is computable and if A_D is well-conditioned, estimates for how well S_{sa} approximates S can be obtained.

6. SUMMARY

We constructed a set of tools which allowed us to invoke the Metis Algorithms with any of the allowable options, to generate different matrix graphs and dual grid graphs for the two families of diffusion problems, to map dual grid graph partitions into partitions of the variables in the original problem, to plot partitions, and to run sequences of tests on the shared SGI Origin computer which is available at Los Alamos National Laboratory. All tests were run in shared queues. The Metis codes are written in C. Our codes, including the ParAINV code, are written in Fortran 90.

Using these tools, a specific algorithm(ParAINV) for solving Equation (1), and Metis-generated partitions which were based on the matrix graph for the problem, we illustrated the possible interplay between the choice of the partition (problem decomposition) and the corresponding *rate of convergence* of a parallel numerical algorithm when the numerics of the algorithm depend upon the partition. We showed that this dependence can be a function of the algorithm and of the choice of problem.

We introduced a different abstract graph model for the K2-SO-2D-100 problem which is based on the Kershaw grid and new graph edge weights which are reflections of the distortions in the Kershaw grid. We used this abstraction to generate partitions which, in our numerical tests, exhibited convergence robustness across choice of such partitions. We were also able to identify a partition method which exhibited robustness in iteration count across the number of processors, $K = 2, 4, 8$.

Ideally, the rate of convergence would be insensitive to the number of processors and to the choice of partition. Therefore, in Section 5 we modified ParAINV by incorporating a known approximate Schur complement correction. In our numerical experiments this modified parallel algorithm, ParAINV(wCorr), exhibited robust convergence with respect to both the choice of partition and the number of processors.

The use of this type of correction is, however, restricted to algorithms which involve approximate Schur complements. Moreover, using this correction requires additional computation and storage, and when using it on the Two-Material test problem, 2M-SO-2D-100, we experienced some increases in the number of iterations required for convergence. Therefore, it is not a straightforward alternative to constructing good partitions from well-chosen weighted graphs, but can be an alternative in complex problems where it is difficult to identify and construct good partitions.

We can conclude that for algorithms where the numerics depend on the choice of the partition, the determination of the sensitivity of the rate of convergence of the algorithm with respect to the choice of the partition should be part of any algorithmic performance studies. Including tests which vary the partition can provide constructive information regarding the robustness of the algorithm and can be used to provide guidance for modifying the algorithm or the choice of the partition method to make the overall computations more robust. For example, problems with complicated grid structure may require special attention.

We intend to continue this research by devising extensions of grid distortion measures to three-dimensional problems drawn from the two families of diffusion matrices, Two-Material and Kershaw, and by exploring the effects of the choice of partition on the rate of convergence of other parallel numerical algorithms, such as parallel incomplete Cholesky preconditioning for solving Equation (1) [33].

ACKNOWLEDGEMENTS

The authors wish to thank Michael DeLong for suggesting that studying the effects of the choice of partition on the rate of convergence of parallel solver algorithms could lead to interesting research questions and results, Matthias Bollhöfer for observing that the correction to a computed approximate Schur complements which is proposed in References [31, 32] is applicable to our work, Michele Benzi for communicating the suggestion of Bollhöfer to us, Michael L. Hall for detailed explanations of the processes used to construct the diffusion test problems, and Bruce Hendrickson for suggesting related domain decomposition references.

REFERENCES

1. Benzi M, Marín J, Tůma M. A two-level parallel preconditioner based on sparse approximate inverses. In *Iterative Methods in Scientific Computation IV*, IMACS Series in Computational and Applied Mathematics, Kincaid DR, Elster AC (eds). IMACS: New Brunswick, NJ, 1999; 167–178.
2. Benzi M, Cullum JK, Tůma M. Robust approximate inverse preconditioning for the conjugate gradient method. *SIAM Journal on Scientific Computing* 2000; **22**:1318–1332.
3. Benzi M, Meyer CD, Tůma M. A sparse approximate inverse preconditioner for the conjugate gradient method. *SIAM Journal on Scientific Computing* 1996; **17**:1135–1149.
4. Hall M. *http://www.lanl.gov/Augustus/[September 2000]*
5. Karypis G, Kumar V. Parallel multilevel graph partitioning. In *Proceedings of the 10th International Parallel Processing Symposium*. IEEE Computer Society Press: Silver Spring, MD, 1996; 314–319.
6. Karypis G, Kumar V. *http://www-users.cs.umn.edu/~karypis/metis/ [April 2002].*
7. Vanderstraeten D, Farhat C, Chen PS, Keunings R, Ozone O. A retrofit based methodology for the fast generation and optimization of large-scale grid partitions: beyond the minimum interface size criterion. *Computer Methods in Applied Mechanics and Engineering* 1996; **133**:25–45.
8. Diekmann R, Preis R, Schlimbach F, Walshaw C. Aspect ratio for mesh partitioning. In *Proceedings Euro-Par '98 Parallel Processing*. Springer-Verlag: Berlin, 1998; 347–351.
9. Walshaw C, Cross M, Diekmann R, Schlimbach F. Multilevel mesh partitioning for optimizing domain shape. *International Journal of High Performance Computing* 1999; **13**:334–353.
10. Diekmann R, Preis R, Schlimbach F, Walshaw C. Shape-optimized mesh partitioning and load balancing for parallel adaptive FEM. *Parallel Computing* 2000; **26**:1555–1581.
11. Diekmann R, Schlimbach F, Walshaw C. Quality balancing for parallel adaptive FEM. In *Proceedings Irregular '98*, Ferreira A, Rolim J, Simon H, Teng SH (eds). Springer-Verlag: Berlin, 1998; 170–181.

12. Dulmage A, Mendelsohn N. Coverings of bipartite graphs. *Canadian Journal of Mathematics* 1958; **10**:517–534.
13. Ashcraft C, Liu JWH. Applications of the Dulmage-Mendelsohn decomposition and network flow to graph bisection improvement. *SIAM Journal on Matrix Analysis and Applications* 1998; **19**:325–354.
14. Pothen A, Fan C. Computing the block triangular form of a sparse matrix. *ACM Transactions on Mathematical Software*. 1990; **16**:303–324.
15. Shashkov MJ, Steinberg S. Support-operator finite-difference algorithms for general elliptic problems. *Journal of Computational Physics* 1995; **118**:131–151.
16. Morel JE, Roberts RM, Shashkov MJ. A local support-operators diffusion discretization scheme for quadrilateral r-z meshes. *Journal of Computational Physics* 1998; **144**:17–51.
17. Kershaw DS. Differencing of the diffusion equation in Lagrangian hydrodynamic codes. *Journal of Computational Physics* 1981; **39**:375–395.
18. Hendrickson B, Leland R. The Chaco Users Guide, Version 2.0. *Technical Report*, SAND95-2344, Sandia National Laboratories, 1995.
19. Pinar A, Hendrickson B. Partitioning for complex objectives. In *Proceedings of 15th International Parallel and Distributed Processing Symposium, Workshop on Solving Irregularly Structured Problems in Parallel, 2001*, CD-ROM, IEEE Computer Society, IEEE, 2001.
20. Karypis G, Aggarwal R, Kumar V, Shekhar S. Multilevel hypergraph partitioning. In *Proceedings 34th Conference on Design Automation*, *Anaheim*, *CA*, *June 9–13*, *1997*, ACM: New York, 1997; 526–529.
21. Hendrickson B. Graph Partitioning and Parallel Solvers: Has the Emperor No Clothes? In *Proceedings Irregular '98*, Springer, Berlin, 1998; 218–225.
22. Hendrickson B. Load Balancing Fictions, Falsehoods and Fallacies. *Applied Mathematical Modelling* 2000; **25**:99–108.
23. Alpert CJ, Kahng AB. Recent directions in netlist partitioning: a survey. Integration, *The VLSI Journal* 1995; **19**:1–81.
24. Hendrickson B, Leland R. A multilevel algorithm for partitioning graphs. In *Proceedings of Supercomputing '95, 3–8 Dec. 1995, San Diego, CA, USA*, ACM: New York, 1995; 626–657
25. Alpert CJ, Huang JH, Kahng AB. Multilevel circuit partitioning. In *Proceedings 34th Conference on Design Automation*, *Anaheim*, *CA*, *June 9–13*, *1997*. ACM: New York, 1997; 530–533.
26. Karypis G, Kumar V. Parallel multilevel k-way partitioning schemes for irregular graphs. *SIAM Review* 1998; **41**:278–300.
27. Karypis G, Kumar V. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*. 1998; **48**:96–129.
28. Karypis G, Kumar V. Multilevel algorithms for multi-constraint graph partitioning. In the *Proceedings of Supercomputing '98*, *Nov. 7–13*, *1998*, *Orlando*, *FL*. IEEE Computer Society, 1998; 801–816.
29. Saad Y. *Iterative Methods for Sparse Linear Systems*, PWS Publishing Company, Boston, MA, USA, 1996.
30. Walshaw C, Cross M, Diekmann R, Schlimbach F. Multilevel mesh partitioning for optimising aspect ratio. In *Proceedings*, *Vector and Parallel Processing*, *VECPAR'98. Third International Conference*, *21–23 June 1998*, *Porto*, *Portugal*, Springer, Berlin, 1999; 285–300.
31. Haase G, Langer U, Meyer A. The Approximate Dirichlet Decomposition Method. Part I: An Algebraic Approach, *Computing* 1991; **47**:137–151.
32. Haase G, Langer U, Meyer A. The Approximate Dirichlet Decomposition Method. Part II: Application to 2nd-order Elliptic BVPs, *Computing* 1991; **47**:153–167.
33. DeLong M. Two examples of the impact of partitioning with Chaco and Metis on the convergence of additive-Schwarz preconditioned FGMRES. *Technical Report LA-UR-97-4181*, Los Alamos National Laboratory, Los Alamos, New Mexico, September 1997.