RC 20667 (12/19/96)
Computer Science/Mathematics 20 pages

Research Report

Distribution and Scheduling Strategies for Parallel Sparse System Solvers in Device Simulation Applications

Vladimir Meñkov

IBM Research Division
T. J. Watson Research Center
Yorktown Heights, NY 10598
and
Computer Science Department
Indiana University
Bloomington, IN 47405
s5menkov@watson.ibm.com

Vijay K. Naik

IBM Research Division
T. J. Watson Research Center
Yorktown Heights, NY 10598
vkn@watson.ibm.com

IBM **Research Division**
Almaden · T. J. Watson · Tokyo · Zurich

Distribution and Scheduling Strategies for Parallel Sparse System Solvers in Device Simulation Applications

Vladimir Meñkov
IBM T. J. Watson Research Center
P. O. Box 218
Yorktown Heights, NY 10598
and
Computer Science Department
Indiana University
Bloomington, IN 47405
s5menkov@watson.ibm.com

Vijay K. Naik
IBM T. J. Watson Research Center
P. O. Box 218
Yorktown Heights, NY 10598
vkn@watson.ibm.com

Abstract

We discuss strategies for partitioning data and scheduling computation and communication suitable for the parallel solution of sparse triangular systems arising in the iterative solution of non-linear PDEs. For partitioning, we consider recursive spectral bisection (RSB) and its modification, RSB with vertex separator. To extract available parallelism and to minimize the effects of communication delays, we consider five different schemes for ordering the variables within each partition. To further minimize the overheads, we use a micro scheduler that preschedules the computation and communication steps for each triangular solve by taking into account both the problem specific and architecture specific parameters. We present performance results of a solver used in a general purpose device simulation code called Fielday on a 32-processor IBM SP2.

This page is intentionally left blank.

Distribution and Scheduling Strategies for Parallel Sparse System Solvers in Device Simulation Applications

Vladimir Meñkov
IBM T. J. Watson Research Center
P. O. Box 218
Yorktown Heights, NY 10598
and
Computer Science Department
Indiana University
Bloomington, IN 47405
s5menkov@watson.ibm.com

Vijay K. Naik
IBM T. J. Watson Research Center
P. O. Box 218
Yorktown Heights, NY 10598
vkn@watson.ibm.com

Abstract

We discuss strategies for partitioning data and scheduling computation and communication suitable for the parallel solution of sparse triangular systems arising in the iterative solution of non-linear PDEs. For partitioning, we consider recursive spectral bisection (RSB) and its modification, RSB with vertex separator. To extract available parallelism and to minimize the effects of communication delays, we consider five different schemes for ordering the variables within each partition. To further minimize the overheads, we use a micro scheduler that preschedules the computation and communication steps for each triangular solve by taking into account both the problem specific and architecture specific parameters. We present performance results of a solver used in a general purpose device simulation code called Fielday on a 32-processor IBM SP2.

1 Introduction

Semiconductor device simulation applications such as Fielday [4, 9, 5] solve the discretized non-linear PDEs representing the drift-diffusion model, on irregular meshes by Newton method. At each Newton step, the solver generates the matrix A (Jacobian) and the right-hand side $\mathbf{b}$ for a non-symmetric linear system $A\mathbf{x} = \mathbf{b}$, and then solves the linear system using a preconditioned iterative method such as BCGS [1, 3]. An incomplete LU (ILU) based preconditioner is often used in these methods. The ILU preconditioner is the product of an upper (U) and a lower triangular (L) matrices that are obtained by incomplete LU-factorization of A with a specified (fixed) fill level l; therefore, sparse lower and upper triangular systems $L\mathbf{v} = \mathbf{w}$ and $U\mathbf{z} = \mathbf{v}$ need to be solved in every step of the iterative solution of the linear system.

When we parallelize such a PDE solver, most computational procedures involved can be parallelized easily and efficiently. These include generating the Jacobian and the right-hand side on every Newton step, as well as performing matrix-vector multiplications and various comparatively cheap vector operations (DAXPYs, dot products) on every CG step. As an example, in Table 1, we present the total execution time (indicated by NL PDE Solve) and the time taken by the individual steps in the solution of the non-linear PDEs, for two representative problems with $n \approx 3 \times 10^4$ variables ($n_{\mathrm{m}} \approx 10^4$ mesh nodes). The number of non-zeros (indicated by nnz) in the A matrix, for both problems, is approximately half a million. The timings are from an IBM SP2 with thin node processors. For the parallel execution, we used one of the best ordering schemes described in this paper (RSB-VS with IBR V). It can be seen from the table that a speed-up of around 7–7.5

Table 1: Breakdown of time used by parallel Fielday for solving a non-linear PDE in two test cases, on different number of processors. The last two rows indicate the total number of Newton and CG iterations required. All timing measurements are wall clock timings (in seconds) on an IBM SP2 with *thin node* processors.

Problem Size	n_{m} =10,759, nnz(A) =517,906					n_{m} =11,219, nnz(A) =539,836				
# Processors	1	2	4	8	16	1	2	4	8	16
nnz($L+U$) $\times 10^3$	1,000	1,017	1,042	1,071	1,088	1,052	1,081	1,109	1,056	1,088
NL PDE Solve	264.5	159.0	104.3	64.3	53.0	261.7	132.4	88.7	47.2	37.6
Jacobian/RHS	47.6	24.2	12.8	6.4	3.9	44.2	22.5	11.5	6.2	3.6
ILU Factoring	51.2	45.6	31.8	25.5	17.5	50.3	39.5	33.4	8.9	7.5
CG Solve	163.8	88.1	59.1	32.0	31.2	165.2	69.2	43.2	31.6	26.1
LU Solve	99.3	54.8	38.6	19.8	20.5	98.9	43.3	25.5	19.5	17.4
Mat-vec Prod	47.0	24.0	14.5	7.2	6.3	48.8	18.4	10.1	7.8	5.7
Newton Iterations	9	9	9	9	9	8	8	8	8	8
CG Iterations	422	428	506	455	501	398	312	335	443	449

on 8 processors can be achieved for Jacobian and right-hand side generation (Jacobian/RHS) or matrix-vector product (Mat-vec Prod) in these sample problems.

It is much more difficult to parallelize ILU factoring, which produces the preconditioner factors computed once per Newton iteration, and the solution of the resulting sparse lower and upper triangular systems on every CG step (LU solve). Efficient parallelization of these two operations, however, is essential for designing a practical parallel PDE solver, because they consume a very significant part of CPU time even when the problem is solved on a single processor ($\sim 20\%$ and $\sim 40\%$, respectively, in the examples shown in Table 1).

Some work on parallelizing these two steps of the PDE solver of Fielday has been presented in [10]. In this report, we describe our further work on improving parallel efficiency of the sparse triangular solver.

In solving the sparse triangular systems $L\mathbf{v} = \mathbf{w}$ and $U\mathbf{z} = \mathbf{v}$ on a parallel computer with p processors, the performance is highly dependent on the selected method of distribution and scheduling of computations. Straightforward partitioning of matrix A and the preconditioner factors L and U into p block rows and mapping the block computations onto processors results in poor performance for two reasons. First, the volume of data communicated may turn out to be very large. In the worst case, a processor may end up sending/receiving almost all of the computed values to/from all other processors. The second reason is that the data dependence in the L- and U-solves (forward and backward substitutions) may serialize many of the computations, reducing the available parallelism.

Since the general problem of finding a distribution and scheduling of computations in LU solve such that maximum parallelism is exploited with minimum amount of communication is NP-complete, we consider heuristic approaches. As a convenient method for dividing the domain into p subdomains of approximately equal size, connected by a small number of edges, we use the recursive spectral bisection (RSB) [2, 13] applied to the graph of the matrix A. In this paper, we occasionally refer to this type of partitioning as "plain RSB", to distinguish it from RSB-VS (see below). The partitioning of the domain leads to reordering of the variables in system to be solved. In Section 2, we present an overview and the rational for the various methods used in ordering the variables. We also discuss the approach we have used for reordering the variables. To exploit parallelism and to overlap computations with communication as far as possible, we reorder variables inside each partition produced by RSB. Several methods for this process, which we call intrablock reordering, and their effects on the data dependency patterns are discussed in Section 3. We also consider an extension to RSB partitioning, which we call RSB-VS, where small vertex separators are formed to decouple the RSB induced partitions. This partitioning and the relevant intrablock reordering options for it are discussed in Section 4.

In Section 5 we analyze the partitioning and ordering methods described in previous sections, and obtain parallel efficiency estimates for them under simplified assumptions.

To further improve the performance, we take advantage of the fact that the LU solver is applied repeatedly with the same coefficient matrix. We use a predictive scheduler that preschedules the computation and communication operations on each processor taking into account the architectural parameters such as cost of arithmetic operations and cost of message passing, as well as problem specific parameters such as the inter-processor data dependency patterns. This scheduler is briefly described in Section 6.

We have implemented these partitionings and scheduling strategies in a version of parallel Fielday. We present experimental results from a 32-processor IBM SP2 in Section 7.

2 Distribution and Scheduling of Computations

Much work has been done to improve parallel efficiency of triangular solve by reordering variables. The strategies proposed in the literature can be divided into two wide classes, which can be called pre-factorization and post-factorization reordering.

A post-factorization reordering amounts to accepting the structure of the preconditioner as produced by incomplete LU factorization of A in the original ordering and finding a suitable (parallelizable) order of performing arithmetic operations in solving a triangular system $L\mathbf{v} = \mathbf{w}$. One method of finding such an order is level scheduling [8, and references therein]. Post-factorization reordering does not affect the convergence of the preconditioned iterative methods.

A pre-factorization approach, commonly used for both direct and iterative solution of $A\mathbf{x} = \mathbf{y}$, involves reordering rows and columns of A using a permutation described by some matrix P, so that instead of $A\mathbf{x} = \mathbf{y}$ one solves the equivalent linear system $(PAP^{\mathrm{T}})(P\mathbf{x}) = P\mathbf{y}$. Complete or incomplete LU factorization of PAP^{T} is performed to obtain either the LU-decomposition of A for direct solve, or the preconditioner $LU \approx PAP^{\mathrm{T}}$ for use in iterative solve. Pre-factorization reordering is commonly done for purposes other than improving parallelism. The permutation is chosen either to reduce the fill or the operation count in LU factorization (in case of the direct solve), or so that the preconditioner factors L and U possess some desirable properties (in case of the iterative solve).

The overview of reordering methods in [11, Sections 4.5–4.8] describes reorderings done to reduce the bandwidth or profile of PAP^{T} (Cuthill-McKee or reverse Cuthill-McKee ordering) or the amount of fill generated in the complete factorization of PAP^{T} (minimum degree ordering). Nested dissection orderings [11, Sections 4.10–4.12 and references therein], such as SND [14], also aim at reducing the amount of fill in, as well as the cost of, complete factorization. Nested dissection ordering also produces matrix structure suitable for parallel implementation of complete or incomplete LU factorization.

Our general approach is pre-factorization reordering discussed in this and following sections, with auxiliary post-factorization operation scheduling (Section 6). Our reordering goals are somewhat different from those of the popular reorderings mentioned above, since in this paper we are mainly interested in parallelizing triangular solve of systems with the sparse matrices L and U obtained by incomplete LU factorization of PAP^{T} with a low level of fill l (typically, $l = 1$ or 2).

For understanding and analyzing the reordering of variables for use in a parallel environment, it is convenient to think of it in terms of two steps. The first step, which we call *partitioning and grouping*, is concerned with partitioning the set of variables into groups, each of which becomes a contiguous group (a *block*) of variables in the new ordering. All matrix rows corresponding to the variables in a block are assigned to the same processor. The system matrix thus reordered will therefore be partitioned into blocks of rows. The second reordering step, which we call *intrablock reordering* or *IBR*, is rearranging variables within each block created in the first step, i.e. on each processor.

This two-stage view of reordering is appropriate for the parallel solution of the non-linear PDEs we consider here. Among the computational steps listed in Table 1, parallel efficiency of Jacobian/RHS generation and matrix-vector product is affected only by the system partitioning (primarily, by the number of edges of the graph of A cut by the partitioning) in the first step. The ILU factoring and LU solve, on the other hand, are also influenced by the ordering of the variables within each block.

Thus, the conceptual plan of designing the global ordering consists of two parts:

- Choosing a domain partitioning scheme that results in a good load balance and cuts only a small number of edges, with or without a view to LU solve performance. Currently we use RSB or its modification RSB-VS for this purpose.

- Choosing an intrablock reordering scheme to improve the parallelism of L- and U-solves. We have developed a number of intrablock reordering methods, referred to as *IBR I, IBR II, ..., IBR V*. These methods try to order variables on a processor so that either the data needed on other processors are computed as early as possible, or the operations that use data produced by other processors are scheduled as late as possible.

In this report we are not concerned with the issues in parallelizing LU factorization, because parallel implementation of this factorization, presented in [10], already has a fairly good parallel efficiency, as it can be seen in Table 1.

In the finite-element method solution of drift-diffusion equations, typically there are three variables per mesh node (electric potential, concentration of electrons, and concentration of holes). Three consecutive rows or columns of the matrix A per each mesh node correspond to these variables. It is desirable to keep these triplets of variables together after reordering. To ensure this, we in fact reorder mesh nodes, rather than single rows and columns of the matrix. We perform the reordering with the "collapsed" one-variable-per-node graph, and then expand the final ordering to produce the three physical variables in each node.

From the point of view of reducing the cost of an LU-solve, the goal of optimizing the system partitioning and intrablock reordering is to achieve the best possible degree of concurrency of computation on all processors, as well as the best degree of overlap of computation and communication. *One has to keep in mind, however, that any reordering of the variables will result in a different ILU preconditioner, even if the same level of fill is maintained in the incomplete LU-factorization step.* This usually does not affect significantly the amount of arithmetic operations per CG iteration, but may drastically influence the convergence rate for some problems. Since both the initial partitioning and the subsequent IBR operations affect the ordering of the variables, they affect the overall performance and must be chosen with care. We intend to address the problem of convergence rate dependence on variable ordering in a future report.

2.1 Simplifications

For simplicity, we assume that A is structurally symmetric. In reality, due to its mesh origin, A is almost structurally symmetric, and we simply insert a few "missing" edges into its graph before doing the dependence analysis.

In the following discussion, when there is no ambiguity, we will simply say "matrix A after such-and-such reordering procedure" and thereafter simply use the notation A instead of PAP^{T}.

2.2 Graphs

For domain partitioning and reordering the variables, we make use of graphs associated with the structures of various sparse matrices. We assume that the reader is familiar with the ordered graph (a.k.a. labelled graph) representation of a sparse matrix. In this discussion, we also assume the standard correspondence between the vertices the graph and the variables of the system to be solved, between the edges of the graph and the non-zeros (either original or "filled") in the matrix. For a discussion on these concepts, please refer to any introductory textbook on sparse matrix computations (e.g., [11]).

In the solution of the non-linear PDE, graph partitioning and intrablock reordering can be applied as soon as the non-zero structure of A is known. By applying symbolic ILU factorization with the specified level of fill l, we can also obtain the structure of $f_l(A) \equiv L + U$, where L and U are the factors of the preconditioner.

The following are the three graphs one can be used in domain partitioning and intrablock reordering:

1. The graph of A. This graph is normally used in RSB partitioning. It is also used in some of the intrablock reordering methods (IBR II, IBR III, IBR IV).

2. The graph of the ILU preconditioner factors. Use of the graph of $f_l(A)$ as an alternative to that of A was proposed in [10]. There is, however, a problem limiting usability of this graph: LU-factoring and reordering are not commutative.

   Since the preconditioner actually used during CG iterations is obtained by ILU factorization not of A (the system matrix in original ordering), but of the permuted matrix PAP^{T} (where we don't know the permutation matrix P yet), it is the graph of $P^{\mathrm{T}} f_l(PAP^{\mathrm{T}})P$, and not that of $f_l(A)$ which is of interest for us.

3. The super preconditioner graph, or SPG. The $\mathrm{SPG}_l(A)$ of A for a given level of fill l is defined as the minimal graph such that for any ordering P of the variables the graph of $P^{\mathrm{T}} f_l(PAP^{\mathrm{T}})P$ is a subset of $\mathrm{SPG}_l(A)$. SPG notion allows one to guarantee a certain graph property of the preconditioner resulting from the ILU of the reordered A (i.e. that of $f_l(PAP\mathrm{T})$) even though P itself is yet to be found. The SPG is used in some reordering schemes described below, such as the second stage of RSB with vertex separator, or in IBR V, described in Section 4.

   Note that the concept of SPG is useless in the case of complete LU factorization, since the SPG in this case is simply a clique (or a union of several cliques, if the graph of A consists of several disconnected components).

3 Plain RSB with Intrablock Ordering for Better Parallelism

We now describe four intrablock reordering methods (IBR I, IBR II, IBR III, and IBR IV) that we developed for use with plain RSB partitioning. Of these, IBR II and IBR III are the best. Methods

I and IV are described here mainly for historical interest and for comparison. A fifth reordering method (IBR V), developed for RSB with vertex separator, is discussed in Section 4.

IBR I. This reordering is based on the graph of the tentative ILU preconditioner: the preconditioner we would have if we factored A after partitioning the set of variables and grouping them into p groups, but before reordering them within the blocks.

This reordering is done as follow. Suppose that after partitioning and grouping the k-th processor has $n_k = i_2^k - i_1^k + 1$ variables numbered $[i_1^k : i_2^k]$. For each variable we find out whether it possesses any one or both of the following properties:

- "L-independent": the variable i is called L-independent if, during the L solve, it can be calculated without immediate use of variables from other blocks. That is, $L_{ij} = 0$ for all $j < i_1$.
- "U-independent": the variable i is called U-independent if, during the U solve, it can be calculated without immediate use of variables from other blocks. That is, $U_{ij} = 0$ for all $j > i_2$.

Thus all variables assigned to processor k can be divided into four groups depending on what combinations of the two properties they possess.

Then they are reordered in such a way that the reordered k-th block row of $L + U$ looks as follows:

Group 1	0		$*$
Group 2	0		0
Group 3	$*$	$L_{kk} + U_{kk}$	$*$
Group 4	0		0
Group 5	$*$		0

Here

- the variables that are L-independent but not U-independent compose group 1;
- the variables that are U-independent but not L-independent compose group 5;
- the variables that are neither L-independent nor U-independent compose group 3;
- the variables that are both L-independent and U-independent are distributed among groups 2 and 4 in the way minimizing $(|G_1| + |G_2|) - (|G_4| + |G_5|)$, where $|G_j|$ is the size of Group j.

As seen from the above diagram, it is desired to obtain a preconditioner structure where, during the L solve, all processors can start computing at the same time, and proceed independently almost until the middle of the block;, and similarly in the U solve. The problem with this scheme, however, is that if we have any fill at all, reordering of variables may result in fill-in in U appearing in different rows than expected according to the diagram above. (The L part does not suffer from this problem.) Thus, the quality of this method leaves much to be desired.

IBR II. Reordering by this method (as well as by IBR III and IBR IV described below) is done using the graph of the original matrix A.

Suppose that after applying the RSB partitioning, the k-th partition (assigned to the k-th processor) has the variables $[i_1^k : i_2^k]$, and of these $n_k = i_2^k - i_1^k + 1$ variables, only m_k variables

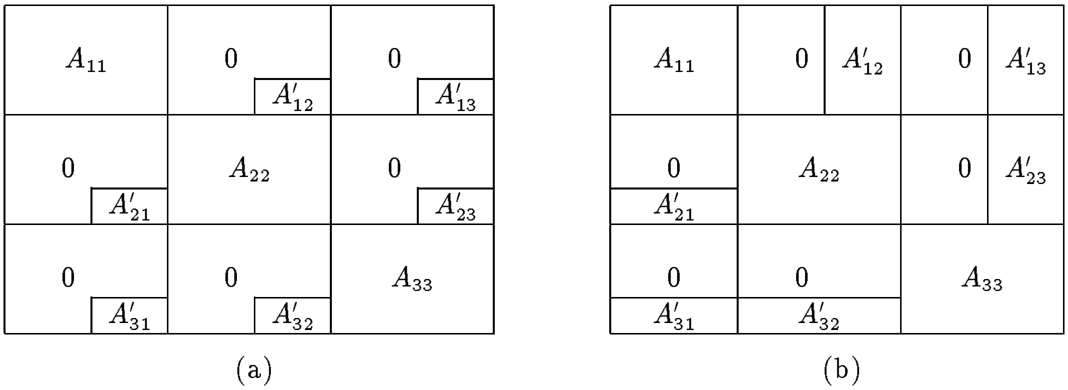


Figure 1: Block structure of A for $p = 3$: (a) after IBR II; (b) after IBR III. For IBR II, each non-zero block A'_{jk} is m_j rows tall and m_k columns wide. For IBR III, each non-zero block A'_{jk} ($j > k$) is $\hat{m}_j$ rows tall, and each non-zero block A'_{jk} ($j < k$) is $\hat{m}_k$ columns wide. For both methods, the block structures of the L and U factors of the ILU preconditioner are the same as those of the corresponding parts of A.

are *border variables*; i.e., the corresponding vertices in the graph of A are connected by edges to vertices in other partitions of the graph. Formally, m_k is the number of elements in the set

$$MB(k) = \{i \;:\; i \in [i_1^k : i_2^k] \quad \wedge \quad (\exists j \quad | \quad j \notin [i_1^k : i_2^k] \;\wedge\; a_{ij} \neq 0)\}.$$

In matrix terms, $MB(k)$ is the set of variables in the k-th processor's block such that corresponding rows of A contain some non-zeros outside of the diagonal block A_{kk}. (The block A_{kk} is defined as the block produced by rows and columns with indices in the range $[i_1^k : i_2^k]$).

If in each block k, the m_k border variables are ordered the last in that block, then there will be now no non-zeros in the first $n_k - m_k$ rows of the k-th block row of A outside of the block A_{kk}, or in the first $n_k - m_k$ columns of the k-th block column of A outside of the block A_{kk}. That is, the non-zeros in any off-diagonal block A_{jk} will be confined to the $m_j \times m_k$ sub-block A'_{jk} in the right lower corner of A_{jk}, as shown in Figure 1(a).

The non-zeros in the off-diagonal blocks of the factors L or U are confined to the same sub-blocks, due to the way LU-factorization (complete or incomplete) is computed. Thus, during the L-solve the k-th processor can start computing the first $n_k - m_k$ of its variables immediately, and calculate them without any input from other processors. Only for computing its last m_k variables the processor will have to wait for input from other processors. In the U-solve, the k-th processor first computes all m_k variables needed by other processors.

IBR III. This is a modification of IBR II. Instead of selecting the m_k "border-area variables"—those coupling block k with other blocks—we select only $\hat{m}_k$ variables connected to the preceding blocks. These variables form the set

$$ML(k) = \{i \;:\; i \in [i_1^k : i_2^k] \quad \wedge \quad (\exists j \quad | \quad (j < i_1^k \;\wedge\; a_{ij} \neq 0))\},$$

i.e. the set of variables i in the k-th block such that there is a non-zero somewhere in the i-th row to the left of the diagonal block A_{kk}. Since $ML(k)$ is a subset $MB(k)$, we have $\hat{m}_k \leq m_k$.

If these $\hat{m}_k$ variables of $ML(k)$ are ordered the last in the k-th block, there will be no non-zeros in the first $n_k - \hat{m}_k$ rows of the k-th block row of A to the left of the block A_{kk}. Because of the assumed structural symmetry of A, there will be no non-zeros in the first $n_k - \hat{m}_k$ columns of the k-th block column above the block A_{kk} as well (see Figure 1(b)). Nor will there be any non-zeros in the same areas of the preconditioner's factors L and U, due to the way LU-factorization (complete or incomplete) is computed. Thus, just like with IBR II, during the L-solve the k-th processor can start computing the first $n_k - \hat{m}_k$ of its variables immediately, and calculate them without any input from other processors; during the U solve, the $\hat{m}_k$ variables needed by other processors are the first thing processor k produces.

IBR IV. This method is a refinement of IBR III. In each block, the $\hat{m}_k$ variables of $ML(k)$ are not only moved to the end of the local variable range, but are also sorted so that the variables from $ML(k)$ which can be computed using input only from processor 1 appear before those that also need input from processor 2, and so on. This could somewhat improve the level of parallelism. Unfortunately, since the reordering is done using the graph of A, this benefit disappears in ILU preconditioners with fill-in.

4 Partitioning with Dividing Planes (Vertex Separators)

The global orderings discussed above (plain RSB + some intrablock reordering) do not eliminate serial dependencies among processors, so common in parallel LU-solve. Figure 2(a) shows a typical dependence diagram for an LU-solve with the preconditioner whose factors L and U have the structure shown in Figure 1(a) or (b). (The diagram is drawn in the assumption that there are some non-zeros in every block $L_{k,k-1}$ and $U_{k,k+1}$, and that all other off-diagonal blocks are zero). In that figure, kP, $1 \leq k \leq 4$, stands for the fully parallel part of the computations assigned to Processor k and kS stands for the sequential computations—computations that need input from other processors in the L-solve, or those that produce data for other processors in U-solve. Although intrablock reordering methods such as IBR II or IBR III are designed to minimize the size of the sequential part, it is generally non-empty. The serial parts of computation are done one after another in a sequential manner and similarly all communication latencies in the 'S' steps are added to the total execution cost.

An attractive alternative to such a dependence pattern is the one that is shown in Figure 2(b). Note that here the length of its critical path is independent of the number of processors. Dependence pattern of this type is obtained if the graph of matrix A is partitioned into $2p+1$ groups, $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_p$, $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_p$, $\mathcal{C}$, so that

- Groups $\mathcal{A}_k$ and $\mathcal{B}_k$ are assigned to Processor k; group $\mathcal{C}$ is assigned to any one processor;
- All preconditioner blocks $L_{\mathcal{B}i,\mathcal{A}j}$ and $U_{\mathcal{A}i,\mathcal{B}j}$ are zeros (Figure 3).

With this preconditioner structure, the forward substitution can be accomplished in 3 stages, the first two of which are highly parallel:

1. Each processor k computes variables from its group $\mathcal{A}_k$ independently of other processors.
2. Each processor k imports variables from groups $\mathcal{A}_1, \ldots, \mathcal{A}_p$ from other processors, as needed, and then computes variables from its group $\mathcal{B}_k$.
3. If there are any nodes in group $\mathcal{C}$, the processor to which the group is assigned imports required variables from other processors and computes group $\mathcal{C}$.

During the U-solve, variables from groups $\mathcal{C}$, $\mathcal{B}_k$, and $\mathcal{A}_k$ are computed in the reverse order.

The main attractive feature of the above partitioning is that computations are performed in clearly defined stages and the data computed during any stage may be needed by another processor only in the computations of the next stage. We call any partitioning with this property a *depend-on-early* scheme.

Figure 4 illustrates two possible approaches for implementing partitioning of the set of variables having the depend-on-early property: *slices* and *dividing planes*.

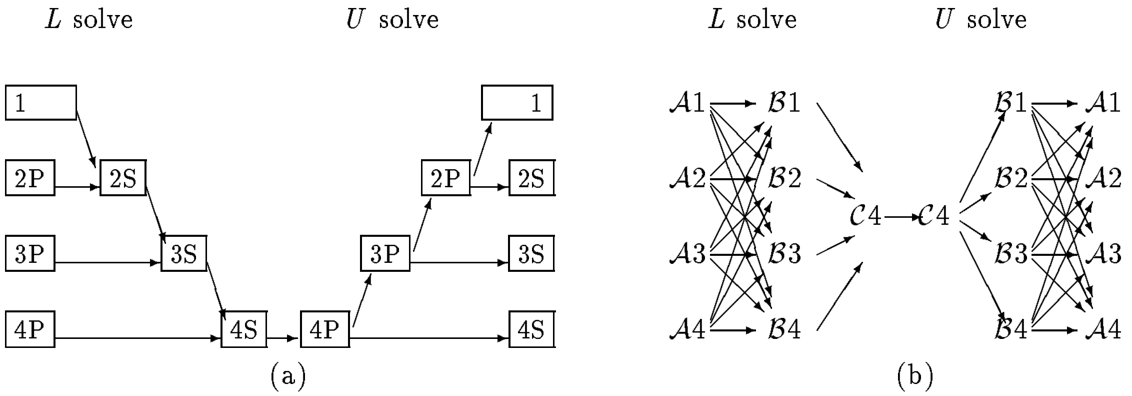


Figure 2: Typical dependence graphs for LU-solve: (a) for a plain RSB and (b) for RSB with vertex separator, using $p = 4$ processors. Digits 1 to 4 stand for processor numbers. In (b), we assume that the variable group $\mathcal{C}$ has been assigned to processor 4.

4.1 Slices

Figure 4(a) shows a domain partitioned into $2p$ slices. If the set of variables can be partitioned into slices so that

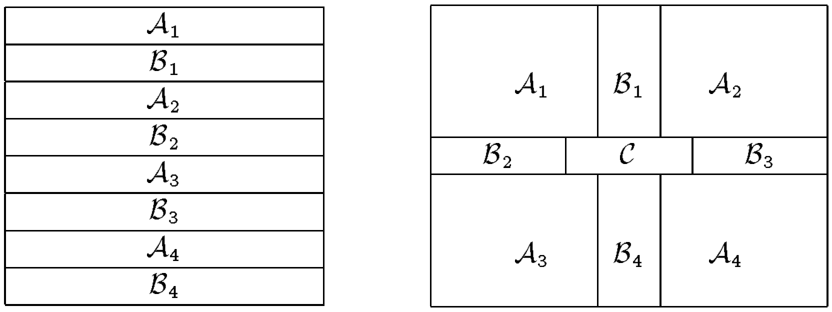
(i) no edge of the graph of A connects a node in $\mathcal{A}_i$ to a node in $\mathcal{A}_j$, $i \neq j$, and

(ii) no edge of the preconditioner graph connects a node in $\mathcal{B}_i$ to a node in $\mathcal{B}_j$, $i \neq j$,

then the resulting partitioning has the desirable depend-on-early property. The requirement (ii), which ensures block-diagonal structure of the $L_{\mathcal{B},\mathcal{B}}$ and $U_{\mathcal{B},\mathcal{B}}$ part of the preconditioner (the block

$$L = \begin{bmatrix} L_{\mathcal{A}1,\mathcal{A}1} & & & & & & & & \\ 0 & L_{\mathcal{A}2,\mathcal{A}2} & & & & & & & \\ 0 & 0 & L_{\mathcal{A}3,\mathcal{A}3} & & & & & & \\ 0 & 0 & 0 & L_{\mathcal{A}4,\mathcal{A}4} & & & & & \\ * & * & * & * & L_{\mathcal{B}1,\mathcal{B}1} & & & & \\ * & * & * & * & 0 & L_{\mathcal{B}2,\mathcal{B}2} & & & \\ * & * & * & * & 0 & 0 & L_{\mathcal{B}3,\mathcal{B}3} & & \\ * & * & * & * & 0 & 0 & 0 & L_{\mathcal{B}4,\mathcal{B}4} & \\ * & * & * & * & * & * & * & * & L_{\mathcal{C},\mathcal{C}} \end{bmatrix}$$

Figure 3: The structure of L in a "depend on early" scheme. Non-empty group $\mathcal{C}$, $p = 4$.



(a) (b)

Figure 4: Schemes of "depend on early" domain partitioning for $p = 4$. (a) Slices and (b) RSB with vertex separator.

diag$\{L_{\mathcal{B}1,\mathcal{B}1}, \dots, L_{\mathcal{B}4,\mathcal{B}4}\}$ in Figure 3), distinguishes this partitioning from classic one-way dissection. (See, for example, [11, Section 4.13], where one-way dissection is considered in the context of complete factorization the notation R_i for $\mathcal{A}_i$ and S_i for $\mathcal{B}_i$ is used.) Note that no $\mathcal{C}$ node group appears in this partitioning.

Although for a regular rectangular mesh this kind of one-way dissection looks attractive, this scheme has many drawbacks:

- Larger amounts of data is transmitted among processors in matrix-vector multiplications or LU solve steps. (For a $\sqrt{n} \times \sqrt{n}$ regular mesh partitioned into p parts, slicing cuts $O(p\sqrt{n})$ edges, as opposed to $O(\sqrt{pn})$ edges cut by the RSB.)

- The maximum possible number of partitions is tightly limited, especially for 3-D problems or for high fill levels. For an estimate, consider a regular rectangular mesh. Any $\mathcal{B}_k$ slice must be at least one node plane thick; and $\mathcal{A}_k$ slice must be at least l planes thick. Thus p processors can be used effectively only if the total thickness of the domain is at least $(l+1)p$ planes. Thus for an $n^{1/3} \times n^{1/3} \times n^{1/3}$ 3D mesh this partitioning cannot be used with more than $n^{1/3}/(l+1)$ processors.

- If the mesh is irregular and slices are thin (many processors), it is difficult or impossible to obtain slices with roughly equal number of nodes and at the same time ensure that separating properties (i) and (ii) hold.

4.2 Dividing planes

Figure 4(b) shows domain partitioning using RSB followed by extracting the set of *dividing planes* or the *vertex separator*, which itself is partitioned subsequently.

We refer to this scheme as *RSB with vertex separator*, or *RSB-VS*. The partitioning is obtained in two steps as follows:

1. Using RSB based on the graph of the matrix A, the entire set of nodes is partitioned into p groups $\mathcal{A}_1^0, \dots, \mathcal{A}_p^0$. Group $\mathcal{BC}$ (the vertex separator) is extracted in some way from the set of the nodes that are near the borders between these p subdomains, so that it separates the remainder groups $\mathcal{A}_1 \equiv \mathcal{A}_1^0 \setminus \mathcal{BC}, \dots, \mathcal{A}_p \equiv \mathcal{A}_p^0 \setminus \mathcal{BC}$ from one another in the graph of A. (The

notation $\mathcal{A}_p^0 \setminus \mathcal{BC}$ means set difference between $\mathcal{A}_p^0$ and $\mathcal{BC}$.) Thus, after this step we have total of $p+1$ groups $\mathcal{A}_1, \ldots, \mathcal{A}_p, \mathcal{BC}$.

2. In the second step, the super preconditioner graph for our level of fill l is used. Using RSB based on the restriction of $\mathrm{SPG}_l(A)$ on $\mathcal{BC}$, the set $\mathcal{BC}$ is first partitioned into p groups $\mathcal{B}_1^0, \ldots, \mathcal{B}_p^0$, from which we then extract the separator set $\mathcal{C}$ separating the groups $\mathcal{B}_1 \equiv \mathcal{B}_1^0 \setminus \mathcal{C}, \ldots, \mathcal{B}_p \equiv \mathcal{B}_p^0 \setminus \mathcal{C}$ from one another in the SPG. One of the groups $\mathcal{B}_1, \ldots, \mathcal{B}_p$ is assigned to each of the p processors; group $\mathcal{C}$ can be assigned to any one processor. (In practice, of course, the assignment of the $\mathcal{B}$ groups to processors should be done in such a way as to reduce the amount of data that needs to send between processors).

This method is more scalable than the slicing scheme, since the number of edges cut is at most about twice as large as in plain RSB, and it is easier to have subdomains of approximately equal sizes for larger p. Our experience indicates that the method performs well with as few as 1000–2000 nodes per subdomain.

RSB-VS vs. SND. The global ordering given by RSB-VS is different from the one given by a nested dissection ordering such as SND. On the first step of the RSB-VS partitioning described above, the separator nodes from all $\log_2 p$ levels of partitioning form a single group $\mathcal{BC}$, placed at the end of the global ordering, while in the nested dissection orderings they are distributed in the hierarchical fashion.

As a result, the forward substitution (or backward substitution) dependence tree for RSB-VS, as shown in Figure 2(b), has only 3 levels ($\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$), regardless of the number of processors. For the usual nested dissection ordering, the number of levels in the forward substitution tree is $\log_2 p + 1$ (if the domain is divided into halves on each dissection) or $\log_4 p + 1$ (if the domain is divided into four parts). Moreover, on the most of these levels only a few processors can be used (see Figure 5(a)).

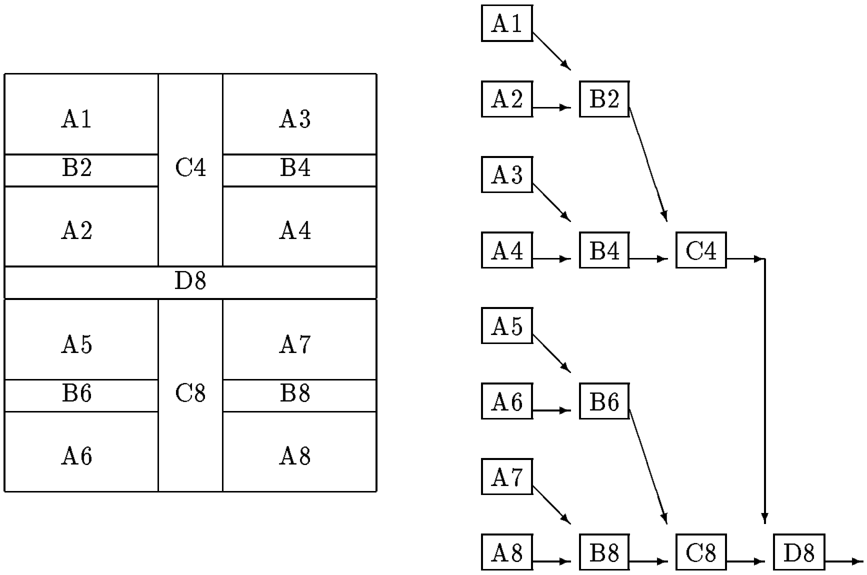
Further partitioning. To further enhance parallelism, one can design an extension of RSB-VS where, group $\mathcal{C}$ itself is partitioned into subgroups assigned to different processors. Will this result in a greater number of levels than only three (corresponding to partitions $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$)?

Group $\mathcal{C}$ in RSB-VS, by its design, is a vertex separator of the $\mathcal{B}$ groups in the super preconditioner graph ($\mathrm{SPG}_l(A)$). In a 2D problem, group $\mathcal{BC}$ in RSB-VS is a union of several almost one-dimensional grids ("connected path[s] of grid lines", as per Duff *et al.* [6]), and group $\mathcal{C}$ is likely to consist of a several ($O(p)$) groups of nodes which are not connected to one another in the super preconditioner graph, but each of which is a clique of the SPG. Such groups ($\mathcal{C}_1$, $\mathcal{C}_2$, ...) can be placed on different processors, but no meaningful next level ("$\mathcal{D}$") appears, since none of the group $\mathcal{C}_1$, $\mathcal{C}_2$, ...is divisible by a separator

On the other hand, in a 3D problem, group $\mathcal{BC}$ is a union of some almost two-dimensional grids, and $\mathcal{C}$ constitutes a union of some almost one-dimensional grids (mesh strips of $\emptyset(l)$ nodes wide and $\emptyset(1)$ node thick). Thus, $\mathcal{C}$ can now be divided by a separator $\mathcal{D}$ into p parts, much like $\mathcal{BC}$ was divided by $\mathcal{C}$ on the previous step. $\mathcal{D}$, in its turn, may consist of a a number of node groups that are not further separable in the SPG.

Thus, the maximum number of levels here is independent of p and is determined by the dimensionality of the problem (3 in 2D and 4 in 3D).

We have not implemented any schemes with partitioning beyond the $\mathcal{C}$ group assigned to a single processor, and do not consider them in the rest of this report.



(a) (b)

Figure 5: Nested dissection for $p = 8$: (a) Domain partitioning. (b) Dependence graph for the L-solve

Vertex separator design. The RSB-VS partitioning scheme described above requires creating two vector separators: first, in the entire graph of A, the separator $\mathcal{BC}$, which separates parts $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_p$ from each other; second, within $\mathcal{BC}$, the separator $\mathcal{C}$ which separates parts $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_p$ from each other. Creating each of those separators is piggybacked on the recursive bisection (RSB) which partitions the graph.

On each stage of RSB, after a domain is split into two subdomains Ω_1 and Ω_2, one can mark in each of them the border nodes set: Γ_1—the set of nodes in Ω_1 adjacent to Ω_2, and Γ_2—the set of nodes in Ω_2 adjacent to Ω_1. We then select in some way (discussed below) a vertex separator $S \subset \Gamma_1 \cup \Gamma_2$, isolating $\Omega_1' = \Omega_1 \setminus S$ from $\Omega_2' = \Omega_1 \setminus S$. After that, Ω_1' and Ω_2' are bisected again in the same way, and so on for $\log_2 p$ levels of RSB. The sum of all separators produced on all levels constitute the total vector separator $\mathcal{BC}$. Extracting $\mathcal{C}$ from $\mathcal{BC}$ is done in a like manner.

How exactly is the separator S created from Γ_1 and Γ_2? There are now several options:

1. Use either Γ_1 or Γ_2;

2. Use $\Gamma_1 \cup \Gamma_2$;

3. Use the minimal vertex separators, as proposed by [13, Section 5]. To do this, we find the minimum cover of the bipartite graph H that is formed by the vertices of Γ_1 and Γ_2 and the edges connecting them. Dulmage-Mendelsohn decomposition of H is used to find this minimum cover [12].

The second approach produces a separator about twice as large as that given by the first, but results in more even balancing of the sizes of subdomains Ω_1' and Ω_2'. The third approach gives the smallest separator, although in practice the difference from the first approach is not that great. Experiments show that partitionings with the minimal separators (the third option) provide better parallel efficiency than those with the other two separator designs. Obviously, it is desirable to reduce the thickness of the separator $\mathcal{BC}$, reducing the number of nodes in $\mathcal{C}$, which is a major drag on performance (see Section 5.2).

4.3 Intrablock ordering for RSB with vertex separator

The data dependence pattern in preconditioner factors with the structure shown in Figure 3 makes intrablock reordering less important for parallelism in RSB-VS than in plain RSB.

IBR V. This reordering method arranges variables inside each group $\mathcal{A}_k$ so that those variables that are needed by other processors during the $\mathcal{B}$ stage of L-solve are put in the beginning of the group and so are computed first. To be able to identify such variables before the intrablock reordering is done, $\mathrm{SPG}_l(A)$ is used.The effect of this reordering on the U-solve is the postponing of the computing of those variables in $\mathcal{A}$ groups that are dependent on data from $\mathcal{B}$.

This reordering has two advantages:

- Without intrablock reordering, even if the group-$\mathcal{A}$ computations are finished at the same time on all processors, the beginning of the group-$\mathcal{B}$ computations may be delayed by the time needed for the data to be transported between processors. The proposed reordering scheme removes this problem, since now data from group-$\mathcal{A}$ nodes are sent out well in advance of the end of the group-$\mathcal{A}$ computing; thus, communication and computation overlap.
- Some overlap between $\mathcal{A}$-work and $\mathcal{B}$-work is now possible. One does not need to balance the amount of work for all $\mathcal{A}$ groups and for all $\mathcal{B}$ groups separately; instead, within reasonably wide limits, only the balance of $A_1 + B_1 \approx A_2 + B_2 \approx \ldots \approx \mathcal{A}_p + \mathcal{B}_p$ (where A_i, B_i, and C are the amounts of work for $\mathcal{A}_i$, $\mathcal{B}_i$) is required.

5 Parallel Efficiency Analysis

In this section we will estimate parallel efficiency of sparse LU solve for several orderings, ignoring communication costs. For the analysis we will also assume that equal amount of arithmetic operations is needed to process each row of L and U in sparse triangular solve, which is the case for almost all matrix rows for many mesh types if the fill level l is low. It is also assumed that the amount of time required to perform a sequence of arithmetic operations is linearly proportional to the number of operations.

5.1 Plain RSB

To estimate the parallel efficiency of sparse triangular solve for IBR II or IBR III, let us assume that we are solving the PDE on a mesh with n_m nodes partitioned into p subdomains with n_m/p nodes each. Suppose also that in each subdomain Ω_k, not more than $\alpha n_\mathrm{m}/p$ nodes are connected by the edges of the mesh to nodes of other subdomains, and out of them only $\hat{\alpha} n_\mathrm{m}/p$ nodes are connected to nodes of the preceding subdomains.

If the linear system has D variables for each mesh node, i.e. $n = Dn_\mathrm{m}$ variables overall and $n/p = Dn_\mathrm{m}/p$ variables in each block, then in each block $m_k \leq \alpha n/p$ and $\hat{m}_k \leq \hat{\alpha} n/p$.

Assuming that an equal amount of time is needed to process each row of L in L-solve, we see that if the total time that would be needed to solve $Lx = y$ on a single-processor machine is t, then Processor 1 will finish its part of L-solve at time t/p, Processor 2 no later than at

	2D					3D
p	2	4	8	16	$p \geq 16$	
α	$2/\sqrt{n}$	$4/\sqrt{n}$	$10/\sqrt{n}$	$16/\sqrt{n}$	$4\sqrt{p/n}$	$6(p/n)^{1/3}$ (for $p \geq 64$)
$\hat{\alpha}$	$2/\sqrt{n}$	$4/\sqrt{n}$	$6/\sqrt{n}$	$8/\sqrt{n}$	$2\sqrt{p/n}$	$3(p/n)^{1/3}$ (for $p \geq 8$)
E_p (IBR II)	0.98	0.89	0.58	0.29	$E_p \geq \frac{1}{1+4(p-1)\sqrt{p/n}}$	$E_p \geq \frac{1}{1+6(p-1)(p/n)^{1/3}}$
$\hat{E}_p$ (IBR III)	0.98	0.89	0.70	0.45	$E_p \geq \frac{1}{1+2(p-1)\sqrt{p/n}}$	$E_p \geq \frac{1}{1+3(p-1)(p/n)^{1/3}}$

Table 2: Estimates of α and $\hat{\alpha}$ for a regular square 2D mesh divided into p parts; corresponding E_p and $\hat{E}_p$ for $n = 10^4$.

$(t/p)(1+\alpha)$, Processor 3 no later than at $(t/p)(1+2\alpha)$ and so on. Processor p finishes last at time t_p, $t_p \leq (t/p)(1+(p-1)\alpha)$. This corresponds to the parallel efficiency

$$E_p = t/(pt_p) \geq 1/(1+(p-1)\alpha) \tag{1}$$

for IBR II. Similarly, we have

$$\hat{E}_p \geq 1/(1+(p-1)\hat{\alpha}) \tag{2}$$

for IBR III.

As a rough estimate for a 2D or 3D mesh problem, we can use the values of α and $\hat{\alpha}$ for the regular square (or cubic) rectangular mesh shown in Table 2. The table also shows resulting parallel efficiency estimates for $n = 10^4$ given by formulas (1) and (2).

5.2 RSB with vertex separators

In general (if no overlap between computation and communication is assumed),

$$E_p = \frac{\Sigma_i(A_i + B_i) + C}{p(\max_i A_i + \max_i B_i + C)}, \tag{3}$$

where A_i, B_i, and C are the amounts of work for $\mathcal{A}_i$, $\mathcal{B}_i$, and $\mathcal{C}$. With good enough computation-communication overlap and $\mathcal{A}$-$\mathcal{B}$ overlap (achieved with IBR V), and no extraordinary load imbalance,

$$E_p = \frac{\Sigma_i(A_i + B_i) + C}{p(\max_i(A_i + B_i) + C)} = \frac{\langle A_i + B_i\rangle + C/p}{\max_i(A_i + B_i) + C}, \tag{4}$$

where $\langle\cdot\rangle$ stands for averaging over the processors. If the balance is good ($\max_i A_i = \langle A_i\rangle$, $\max_i B_i = \langle B_i\rangle$ for no intrablock reordering; or simply $\max_i(A_i + B_i) = \Sigma_i(A_i + B_i)/p$, with IBR V), equations (3) and (4) give

$$E_p = \frac{1}{1+(p-1)\gamma},$$

where

$$\gamma = \frac{C}{W},$$

and W is the total amount of work. An approximate estimate of γ can be given as

$$\gamma \approx \frac{n_C}{n},$$

where $n_{\mathcal{C}}$ is the number of variables in group $\mathcal{C}$.

For many domain and mesh geometries it can be verified that $n_{\mathcal{C}} = O(lp)$ or $O(lp(n/p)^{1/3})$ for a 2D or 3D problems, respectively. Thus, for a sufficient small fill level l, γ behaves as a lower power of n ($O(p/n)$ and $O((p/n)^{2/3})$) than α (or $\hat{\alpha}$) in formulas (1) and (2), which behave as $O((p/n)^{1/2})$ and $O((p/n)^{1/3})$ for 2D and 3D problems, respectively. The estimated efficiency of the LU-solver with well balanced RSB-VS partitioning for a 2D mesh is

$$E_p(\text{RSB-VS, 2D}) \approx \frac{1}{1 + cl(p-1)(p/n)},$$

and for a 3D mesh it is

$$E_p(\text{RSB-VS, 3D}) \approx \frac{1}{1 + cl(p-1)(p/n)^{2/3}}$$

with some constant c. These compare favorably with those obtained for LU solve with SND, which can be estimated as

$$E_p(\text{SND}, 2\text{D}) \approx \frac{1}{1 + cp/\sqrt{n}}$$

or

$$E_p(\text{SND}, 3\text{D}) \approx \frac{1}{1 + cp/n^{1/3}}$$

with some constant c, if the data are assigned to processors as shown in Figure 5.

As l grows, the advantage of RSB-VS disappears. In the limit of complete LU factorization, the entire area $\mathcal{BC}$ will be given over to $\mathcal{C}$, which means that the sequential part of the LU solve is now of approximately the same length as in plain RSB with IBR II or IBR III.

6 Symbolic Execution

Unlike a sequential *LU* solver, which can be described by a dozen lines of code computing all variables in linear sequence, a parallel solver is much more complex. It needs to make a number of important decisions, concerned e.g. with the order of sending and receiving messages to/from other processors, or with the order of computing variables depending on what data are currently available.

Our early experiments have demonstrated that it is very inefficient to postpone such decisions until the actual L and U solves. Instead, once the preconditioner structure is known, we model the *LU* solve using our so-called *symbolic execution* module or *scheduler*. Based on our knowledge about the cost of arithmetic operations and communication latency and bandwidth, this module predicts, for each processor, how much time it will take for it to get to one or another point in *LU* solve, and when data from other processors become available, and schedules interprocessor communications. The resulting "program"—a sequence of pseudocommands indicating operations to perform—is then used by the LU solver on each iteration.

An important use of the symbolic execution module is to analyze quality of a particular partitioning or ordering as judged by the scheduler. Among other things, it predicts for what messages the given processor will wait, and the total amount of waiting.

Although it is a useful tool, one still cannot rely on the scheduler too much. Among its shortcomings is the fact that it counts only floating-point operations and assumes that all addition-multiplication pairs take the same amount of time. Thus it disregards overhead costs (relating to indirection etc.) and dependence of the floating-point operation rate on data locality.

Another source of imprecise predictions is the fact that the processors do not belong to Fielday 100% of the time: occasionally, a system process can take a processor away for a few OS scheduling quanta. However, this turned out not a big problem though: even for 16 processors, only 4 or 5 LU solves out of 1200 in a test run were delayed (by some 55 or 60 msec) by what appeared to be one of the processor's being "taken away".

Scheduling becomes less reliable as p grows; it becomes harder to predict the order in which messages arrive. Incorrect assumptions about the communication model parameters may result in wrong scheduling decisions: for example, a processor is made to wait for a message that has not arrived yet, while another message that has already arrived is ignored. It is thought that these effects should be less pronounced in data layouts with simpler dependence patterns, like the one for RSB-VS (Figure 2(b)), than those with complex multi-step dependencies, such as those for plain RSB (Figure 2(a)).

7 Experimental Results.

Tables 3 and 4 contain some performance data for four numerical problems solved by Fielday on 1 to 16 or 32 processors using RSB and RSB-VS partitioning and various intrablock reordering methods. All experiments were performed on a dedicated 32-processor partition of an IBM SP2. Each processing element of this partition is an IBM RS/6000 model 390 processor (*thin node*), with 64 kbyte data cache, 66.5 MHz clock speed, and 128 Mbyte of RAM.

We considered four problems, all of which arise from finite element discretization of non-linear PDEs on two-dimensional meshes with $n_{\mathrm{m}} = 10207$, 21451, 10759, and 11219 nodes. The number of variables for each problem is $n = 3n_{\mathrm{m}} + n_{\mathrm{c}}$, where ($n_{\mathrm{c}} \leq 4$) of the number of contact variables. For each problem, the fill level (l) is indicated.[1]

For each method, Tables 3 and 4 contain the time needed for one conjugate gradient iteration (t_{it}) and time needed for one LU-solve (t_{LU}). These are actual wall clock times in seconds. In the variety of conjugate gradient methods (biconjugate gradient squared) being used to solve the arising non-symmetric linear systems, two LU-solve are needed per iteration. Thus. t_{LU} is routinely around one third of t_{it}, the rest of the time during an iteration being taken by matrix-vector multiplication, DAXPY operations, and computing dot products.

The tables also contain the actual (ω_{act}) and model (ω_{mod}) waiting ratios (given in per cent). The actual waiting ratio is calculated as

$$\omega_{\mathrm{act}} = \frac{t_{\mathrm{comm}}}{t_{\mathrm{LU}}},$$

where the (wall clock) time taken by all communication calls in an LU-solve, averaged over all processors. The model waiting ratio is given by the scheduler as

$$\omega_{\mathrm{mod}} = \frac{\sum_{i=1,\ldots,p} \text{model waiting time at Processor } i}{\max_{i=1,\ldots,p} \text{model LU time at Processor } i}.$$

If other overhead (due to communication costs, or due to shorter loops in the parallel program than in the sequential one) is negligible, the waiting ratio is directly linked to the speed-up on p processors

$$S_p = p(1-\omega),$$

and the parallel efficiency

$$E_p = 1-\omega,$$

for the waiting ratio ω.

[1] The fill level is given using the common convention where level 0 means no fill.

As expected, IBR II or IBR III appear the best for plain RSB, IBR IV giving no noticeable advantage over IBR III. IBR I occupies an intermediate position between these best methods and an ordering with no IBR. Performance data for IBR V are given for comparison. As expected (since this IBR method is not even designed for use with plain RSB), they are worse than for IBR II–IV.

Remarkably enough, the "waiting ratio", $\omega = 1 - E_p$, given by the estimates in Table 2 agrees, at least within the factor of 2, with the values observed in our experiments ($\omega_{\mathbf{act}}$ in Tables 3–4) for IBR II and IBR III.

However, no pronounced difference between the performance of LU-solver with IBR II and IBR III is observed in data presented in Tables 3 and 4, even though $\hat{m_k} \leq m_k$ and $\hat{E}_p \geq E_p$ for the estimates (1) and (2). Occasionally IBR II performs even better than IBR III, presumably due to other effects the reordering has on the preconditioner structure.

For RSB-VS, IBR V is supposed to be better than natural ordering, but this is not seen in the experimental data being presented.

8 Conclusions

Our new intrablock reordering methods—IBR II and IBR III—for plain RSB, as well as the new partitioning scheme—RSB-VS—give much better LU-solve performance than plain RSB with natural ordering. Although the experimental results we have presented in this paper do not indicate noticeable difference in performance between the RSB-VS method of partitioning and the plain RSB (with the best IBR available), a potential advantage of RSB-VS is that this method is relatively insensitive to the intrablock reorderings; i.e., one can use any intrablock ordering with RSB-VS and thus, there is more freedom choosing a reordering scheme according to other criteria, such as reducing fill-in or improving convergence. Performance improvements obtained by using plain RSB with new reordering methods or by using RSB-VS are most significant for the mid-range values of p (4 to 16 processors on problems with $\sim 10^4$ mesh nodes), often raising the limit of the maximal useful number of processors by a factor of two, as compared to plain RSB with natural ordering.

In this paper, we have focused primarily on the partitioning and scheduling of computations in the triangular solves. In some cases, parallelism of the L and U solves may be enhanced by improving load balancing. The RSB partitioning, as usually implemented, is concerned with balanced distribution of nodes among the subdomains. The balance of nodes may not exactly correspond to the balance of work, however, for two reasons:

1. The number of edges in a subdomain is more important for the amount of work than the number of edges.

2. During ILU, different amount of fill-in may appear in different rows.

In our experiments, we made an attempt to alleviate the problem by balancing, not just the number of nodes among processors, but some linear combination of the number of nodes and the number of edges. Our results so far have been inconclusive and, in our future work, we hope to get a better understanding of the load balancing aspect.

Finally, we note that our current observations, as well as those in [10] indicate that in some cases the parallelism-enhancing ordering of variables is bad for convergence of the CG iterative process. This fact has been noted by Duff and Meurant in their survey of orderings [7]. In a future publication we plan to address this problem. In that report we will introduce the concept of a so-called full-support property of the preconditioner, and will try to demonstrate correlation between the preconditioner's having this property and the convergence rate.

Table 3: Timing measurements for different partitioning and intrablock reordering methods, on IBM SP2 with *thin nodes*.

Problem	p		RSB						RSB-VS	
ID		IBR	None	I	II	III	IV	V	None	V
$n_{\mathrm{m}} = 10207$,	1	t_{it}	0.319	0.323	0.316	0.318	0.317	0.317	0.316	0.317
		t_{LU}	0.086	0.085	0.085	0.085	0.085	0.085	0.086	0.085
		ω_{act}, %	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
		ω_{mod}, %	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
l=1	2	t_{it}	0.212	0.167	0.172	0.173	0.173	0.193	0.168	0.167
		t_{LU}	0.066	0.046	0.046	0.047	0.048	0.059	0.046	0.046
		ω_{act}, %	25.9	3.9	4.8	5.1	5.4	19.7	5.5	4.6
		ω_{mod}, %	27.5	0.7	0.4	0.7	0.7	0.7	1.6	1.8
	4	t_{it}	0.137	0.112	0.101	0.102	0.102	0.111	0.095	0.090
		t_{LU}	0.048	0.035	0.031	0.031	0.031	0.035	0.027	0.025
		ω_{act}, %	44.0	21.7	13.9	14.0	14.3	27.2	13.6	9.9
		ω_{mod}, %	44.7	30.5	20.9	21.5	21.5	19.9	2.9	3.0
	8	t_{it}	0.096	0.057	0.056	0.055	0.056	0.065	0.056	0.055
		t_{LU}	0.037	0.017	0.017	0.016	0.017	0.021	0.016	0.016
		ω_{act}, %	57.7	22.6	21.3	21.8	22.6	38.3	27.6	25.7
		ω_{mod}, %	64.5	20.6	16.3	13.7	13.7	29.9	8.8	7.3
	16	t_{it}	0.065	0.038	0.039	0.040	0.035	0.043	0.042	0.041
		t_{LU}	0.026	0.012	0.013	0.012	0.011	0.015	0.013	0.013
		ω_{act}, %	68.2	38.8	41.9	46.4	38.2	51.7	51.5	50.9
		ω_{mod}, %	73.2	34.4	41.0	29.5	29.1	50.4	13.6	11.2
$n_{\mathrm{m}} = 11219$,	1	t_{it}	0.406	0.408	0.409	0.406	0.406	0.413	0.405	0.415
		t_{LU}	0.122	0.122	0.123	0.123	0.123	0.122	0.122	0.123
		ω_{act}, %	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
		ω_{mod}, %	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
l=2	2	t_{it}	0.224	0.234	0.228	0.227	0.227	0.223	0.222	0.222
		t_{LU}	0.071	0.070	0.071	0.070	0.069	0.068	0.069	0.069
		ω_{act}, %	4.9	8.7	6.2	4.2	3.9	5.2	3.6	5.4
		ω_{mod}, %	4.1	5.3	4.6	4.0	4.0	5.4	4.7	5.5
	4	t_{it}	0.145	0.131	0.127	0.131	0.129	0.133	0.132	0.129
		t_{LU}	0.051	0.043	0.042	0.044	0.043	0.045	0.042	0.042
		ω_{act}, %	21.7	14.9	13.1	15.4	14.8	17.3	13.4	14.8
		ω_{mod}, %	26.7	16.8	18.5	17.6	17.6	16.7	15.7	17.8
	8	t_{it}	0.094	0.092	0.079	0.080	0.081	0.085	0.075	0.071
		t_{LU}	0.034	0.034	0.026	0.026	0.027	0.030	0.025	0.023
		ω_{act}, %	44.1	32.1	27.9	27.4	27.0	33.2	26.9	26.9
		ω_{mod}, %	50.6	48.0	31.6	33.8	33.8	41.4	6.9	6.8
	16	t_{it}	0.081	0.050	0.045	0.051	0.052	0.054	$0.063^{(*)}$	0.058
		t_{LU}	0.033	0.018	0.015	0.018	0.019	0.020	0.022	0.020
		ω_{act}, %	64.1	35.4	34.2	36.0	35.9	44.6	56.8	54.5
		ω_{mod}, %	72.1	47.2	34.9	47.7	48.4	47.3	10.5	9.8

Note (*): Due to poor convergence, the fill level for this run was increased by the solver in run time; therefore, the time is out of line.

Table 4: Timing measurements for different partitioning and intrablock reordering methods, on IBM SP2 with *thin nodes*.

Problem	p		RSB						RSB-VS	
		IBR	None	I	II	III	IV	V	None	V
$n_{\mathrm{m}} = 10759$,	1	t_{it}	0.395	0.389	0.389	0.388	0.389	0.388	0.393	0.388
		t_{LU}	0.116	0.116	0.117	0.116	0.116	0.116	0.119	0.116
		ω_{act}, %	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06
		ω_{mod}, %	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
l=2	2	t_{it}	0.221	0.219	0.214	0.213	0.213	0.214	0.214	0.206
		t_{LU}	0.071	0.069	0.067	0.066	0.066	0.066	0.066	0.063
		ω_{act}, %	7.2	6.7	3.6	3.9	3.8	7.0	4.3	4.3
		ω_{mod}, %	7.0	3.7	4.2	3.6	3.6	3.8	4.3	3.4
	4	t_{it}	0.163	0.156	0.127	0.133	0.136	0.134	0.119	0.117
		t_{LU}	0.060	0.056	0.043	0.046	0.047	0.046	0.038	0.038
		ω_{act}, %	36.8	20.0	14.3	15.2	15.9	25.7	12.9	12.4
		ω_{mod}, %	45.0	34.7	21.6	26.9	26.9	9.3	11.3	11.9
	8	t_{it}	0.138	0.091	0.079	0.080	0.079	0.086	0.072	0.070
		t_{LU}	0.056	0.034	0.027	0.029	0.028	0.032	0.023	0.023
		ω_{act}, %	62.5	34.1	27.7	26.0	25.5	41.1	24.2	23.2
		ω_{mod}, %	69.9	45.9	36.0	36.8	36.7	35.6	19.9	19.4
	16	t_{it}	0.086	0.057	0.047	0.048	0.062	0.060	0.062	0.062
		t_{LU}	0.036	0.021	0.016	0.017	0.021	0.023	0.024	0.024
		ω_{act}, %	69.0	47.4	36.7	39.1	45.1	54.8	57.2	57.6
		ω_{mod}, %	75.1	54.8	40.2	38.7	50.2	53.5	37.5	34.9
$n_{\mathrm{m}} = 21451$,	1	t_{it}	0.691	0.692	0.723	0.667	0.662		0.702	0.709
		t_{LU}	0.175	0.178	0.200	0.172	0.168		0.188	0.171
		ω_{act}, %	0.08	0.07	0.03	0.04	0.07		0.04	0.04
		ω_{mod}, %	0.0	0.0	0.0	0.0	0.0		0.0	0.0
l=1	2	t_{it}	0.451	0.352	0.335	0.332	0.332		0.328	0.399
		t_{LU}	0.152	0.103	0.094	0.093	0.093		0.090	0.089
		ω_{act}, %	32.7	9.4	5.6	5.4	4.9		3.0	3.2
		ω_{mod}, %	37.2	0.5	1.1	0.6	0.6		0.5	0.4
	4	t_{it}	0.252	0.197	0.184	0.184	0.184		0.178	0.176
		t_{LU}	0.088	0.061	0.052	0.050	0.050		0.051	0.050
		ω_{act}, %	41.4	18.7	12.1	11.6	10.3		11.7	11.2
		ω_{mod}, %	45.5	9.6	3.0	2.3	3.0		1.2	1.0
	8	t_{it}	0.165	0.118	0.116	0.114	0.113		0.101	0.101
		t_{LU}	0.063	0.038	0.039	0.036	0.035		0.029	0.030
		ω_{act}, %	55.6	28.6	27.5	27.3	25.8		21.7	22.6
		ω_{mod}, %	61.6	28.1	30.3	23.3	23.3		2.9	3.1
	16	t_{it}	0.111	0.074	0.072	0.065	0.071		0.070	0.066
		t_{LU}	0.044	0.026	0.023	0.021	0.024		0.021	0.021
		ω_{act}, %	62.7	38.5	39.8	32.5	39.0		41.8	40.3
		ω_{mod}, %	71.6	40.0	32.7	32.8	32.8		5.0	4.3
	32	t_{it}	0.080	0.042	0.045	0.041	0.040		0.063	0.062
		t_{LU}	0.033	0.015	0.016	0.013	0.013		0.021	0.022
		ω_{act}, %	71.6	44.9	47.8	45.1	44.9		69.3	69.8
		ω_{mod}, %	80.4	43.4	46.8	45.4	45.6		9.9	9.7

Acknowledgments. The authors would like to thank MeiKei Ieong and Shukla Kapur of IBM Microelectronics division and Giovanni Fiorenza and Bob Walkup of IBM Research Division for many fruitful discussions on Fielday device simulator used in this study.

References

[1] O. Axelsson, *Iterative Solution Methods*, Cambridge University Press, Cambridge, UK, 1994.

[2] S. Barnard and H. Simon, *Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems*, Concurrency: Practice and Experience, 6 (1994), pp. 101–117.

[3] R. Barrett et al., *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, 1 ed., 1994. Also available at the URL `http://www.netlib.org/linalg/html_templates/Templates.html`.

[4] E. Buturla, P. Cottrell, B. Grossman, and K. Salsburg, *Finite-element analysis of semiconductor devices: The FIELDAY program*, IBM J. Res. Develop., 25 (1981), pp. 218–231.

[5] D. Cole, E. Buturla, S. Furkay, et al., *The use of simulation in semiconductor technology development*, Solid-State Electronics, 33 (1990), pp. 591–623.

[6] I. Duff, A. Erisman, and J. Reid, *On George's nested dissection method*, SIAM Journal of Numerical Analysis, 13 (1976), pp. 686–695.

[7] I. S. Duff and G. Meurant, *The effect of ordering on preconditioned conjugate gradients*, BIT, 29 (1989), pp. 635–657.

[8] F. Khoury, *Efficient Parallel Triangular System Solvers for Preconditioning Large Sparse Linear Systems*, PhD thesis, University of New South Wales, 1994. Also available at the URL `http://server.srcpc.unsw.edu.au/ftp/papers/Khoury-thesis/thesis.html`.

[9] S. Laux, E. Buturla, W. Chang, P. Carson, P. Cottrell, E. Farrell, and J. Mandelman, *Modeling using the "FIELDAY Program"*, in Large Scale Computational Device Modeling, K. Hess, ed., University of Illinois Press, 1986.

[10] V. K. Naik, K. Eswar, and M. Ieong, *Scalability of sparse solvers in the context of semiconductor device simulation applications*, Tech. Rep. RC 20436 (4/17/96), IBM Research Division, T. J. Watson Research Center, P. O. Box 218, Yorktown Heights, NY 10598, 1996. Also available in Proceedings of Semiconductor Device Modeling Workshop at NASA Ames Research Center, March 28–29, 1996, and at the URL `http://www.watson.ibm.com:8080/main-cgi-bin/search_paper.pl/entry_ids=8088`.

[11] S. Pissanetzky, *Sparse Matrix Technology*, Academic Press, Inc., London, UK, 1984.

[12] A. Pothen and C.-J. Fan, *Computing the block-triangular form of a sparse matrix*, ACM Trans. Math. Soft., 16 (1990), pp. 303–324.

[13] A. Pothen, H. Simon, and K.-P. Liou, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM Journal of Matrix Analysis and Applications, 11 (1990), pp. 430–452.

[14] L. Wang, *Spectral Nested Dissection*, PhD thesis, Pennsylvania State University, 1994.