Preconditioning Parallel Sparse Iterative Solvers for Circuit Simulation

A. Basermann, U. Jaekel[*]*, and K. Hachiya*[†]

1 Introduction

One important mathematical problem in simulation of large electrical circuits is the solution of high-dimensional linear equation systems. The corresponding matrices are real, non-symmetric, very ill-conditioned, have an irregular sparsity pattern, and include a few dense rows and columns.

When the systems become large, iterative solvers are very likely to outperform direct methods. For convergence acceleration of iterative solvers, parallelization and appropriate preconditioning are suited techniques to reduce the execution time.

We present a parallel iterative algorithm with distributed Schur complement (DSC) preconditioning [5] which achieves an accuracy of the solution similar to a direct solver but usually is distinctly faster for large problems. The parallel efficiency of the method is increased by transforming the equation system into a problem without dense rows and columns which results in a system reduction as well as by exploitation of parallel graph partitioning methods. The costs of local, incomplete LU decompositions are decreased by fill-in reducing reordering methods of the matrix and a threshold strategy for the factorization.

2 Problem of Dense Rows and Columns

Matrices from circuit simulation problems are usually very sparse but include a few (nearly) dense rows and columns. In the parallel case, dense rows and columns are difficult to handle for partitioning methods since they result in couplings between

[*]C & C Research Laboratories, NEC Europe Ltd., D-53757 Sankt Augustin, Germany, {basermann, jaekel}@ccrl-nece.de

[†]High-End Design Technology Development Group, Technology Foundation Development Division, NEC Electronics Corp., Japan, k-hachiya@ax.jp.nec.com

all equations. In addition, good load balance is hard to achieve if the matrix is distributed row-wise. Fill-in reducing ordering methods may become very costly due to a few dense rows and columns, and the matrices may get very ill-conditioned.

Fortunately, dense rows and columns are usually easy to remove from circuit simulation matrices since the corresponding columns or rows as a rule have only one non-zero entry on the diagonal. Such equations normally include voltage sources (constraints). A dense column whose corresponding row has only one diagonal entry can be removed since the corresponding unknown can be determined from the row equation and substituted in all other equations. On the other hand, a dense row (equation) whose corresponding column has merely one diagonal entry is only responsible for the corresponding unknown. All other equations can be solved independently. Fig. 1 illustrates the principle of the matrix reduction in both cases. If the corresponding columns or rows of dense rows and columns do not have one

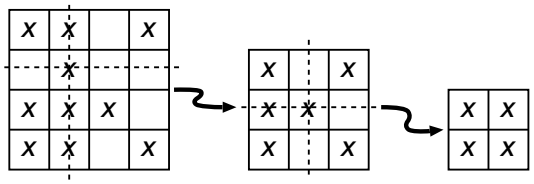

Figure 1. *Removal of (nearly) dense rows and columns.*

diagonal entry only such rows and columns can be handled by using the Woodbury formula [3]. This case is rare for circuit simulation problems and does not occur for the matrices investigated here.

3 Distributed Schur Complement Techniques

In the following, techniques for the iterative solution with DSC preconditioning of circuit simulation equation systems are sketched. More details, in particular on the theory, can be found in [5].

3.1 Definitions

Fig. 2 schematically displays the row-wise distribution of a matrix A to two processors. Each processor owns its local row block. The square matrices A_i are the local diagonal blocks of A. We assume that the local rows are arranged in such a way that the rows without couplings to the other processor(s) come first and then the rows with couplings. The former are called *internal rows*, have only entries in the A_i part of the local rows and are not coupled with rows of other processors. The latter additionally have entries outside the A_i part or are coupled with rows of other processors. These local rows are named *local interface rows*. The part outside A_i which represents couplings between the processors is called *local interface matrix* X_i. From the view of processor 2 in Fig. 2, the local interface rows of processor 1 with entries at column positions in the area of X_2 are *external interface rows*. Since

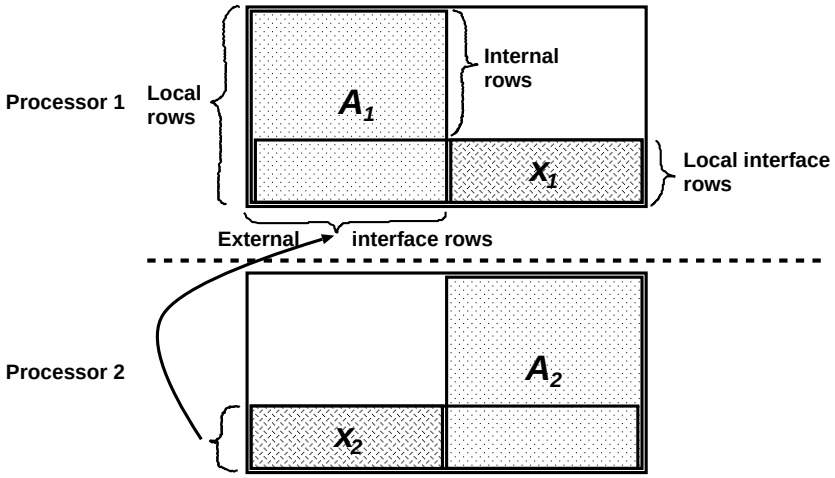

Figure 2. *DSC definitions: Matrix distributed to two processors.*

the sparsity pattern of circuit simulation matrices usually is non-symmetric local interface rows of processor i may have entries in A_i only but are uni-directionally coupled with rows of other processors. These rows are external interface rows from the view of the other processors. This can not be determined locally on processor i, communication is necessary. Since each row of the matrix corresponds to a specific unknown of the equation system (row 1 to solution vector component 1 and to right hand side component 1, e.g.) *internal unknowns*, *local interface unknowns*, and *external interface unknowns* can be defined correspondingly.

3.2 Algorithm

Fig. 3 gives a schematic survey of the DSC algorithm per processor. On each processor an outer Bi-CGSTAB iteration [6] is performed for all local rows (unknowns). As basic iterative method, a flexible variant of GMRES, FGMRES [4, 5], is also well suited for the DSC algorithm but is not considered here because of its higher storage requirements. The outer iteration contains a partial matrix-vector multiplication which requires communication since each processor only owns its local segment of the vector. It is necessary to exchange components of non-local vector segments which correspond to external interface unknowns (rows).

Within the outer Bi-CGSTAB iteration, an inner Bi-CGSTAB iteration for the local interface rows (unknowns) only is performed. This includes a partial matrix-vector multiplication of the interface system but the communication scheme is the same as for the outer matrix-vector multiplication and thus has to be implemented only once.

From the mathematical point of view, each processer i solves the following

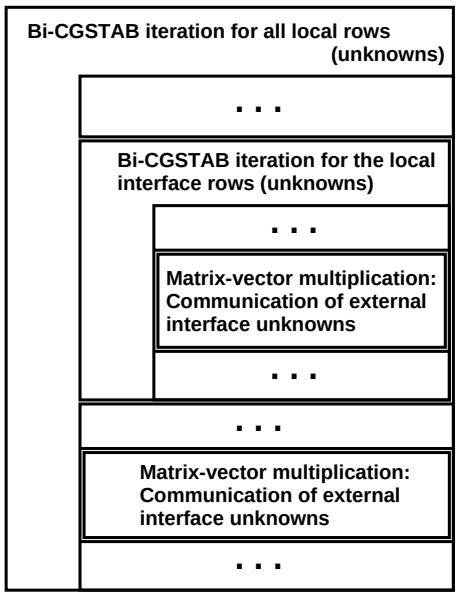

Figure 3. *Schematic view of the DSC algorithm on each processor.*

equation:

$$A_i\, x_i + X_i\, y_{i,\mathrm{ext}} = b_i\,, \quad x_i = \begin{pmatrix} u_i \\ y_i \end{pmatrix}, \quad b_i = \begin{pmatrix} f_i \\ g_i \end{pmatrix}\,. \tag{1}$$

x_i are the local vector components, $y_{i,\mathrm{ext}}$ the external interface vector components, and b_i is the local segment of the right hand side vector. x_i is split into the internal vector components u_i and the local interface vector components y_i, b_i accordingly.

A_i is then split (see [5] for details), and (1) is reformulated:

$$A_i = \begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix} \rightarrow \begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix} \begin{pmatrix} u_i \\ y_i \end{pmatrix} + \begin{pmatrix} 0 \\ \sum_{\mathrm{Neighbours}\, j} E_{ij}\, y_j \end{pmatrix} = \begin{pmatrix} f_i \\ g_i \end{pmatrix}. \tag{2}$$

The result of the sum over all neighbouring processors j with couplings to processor i in (2) is the same as that of $X_i\, y_{i,\mathrm{ext}}$ in (1). $E_{ij}\, y_j$ is the part of $X_i\, y_{i,\mathrm{ext}}$ which reflects the contribution to the local equation from the neighbouring processor j.

The matrix equation (2) represents two equations. From the first, we derive an expression for u_i, substitute u_i in the second equation and get

$$u_i = B_i^{-1}(f_i - F_i\, y_i) \;\rightarrow\; S_i\, y_i + \sum_{\mathrm{Neighbours}\, j} E_{ij}\, y_j = g_i - E_i\, B_i^{-1} f_i\,. \tag{3}$$

$S_i = C_i - E_i\, B_i^{-1} F_i$ is the *local Schur complement*. Note that (3) is an equation for the interface vector components only.

(3) can be rewritten as a block-Jacobi preconditioned Schur complement sys-

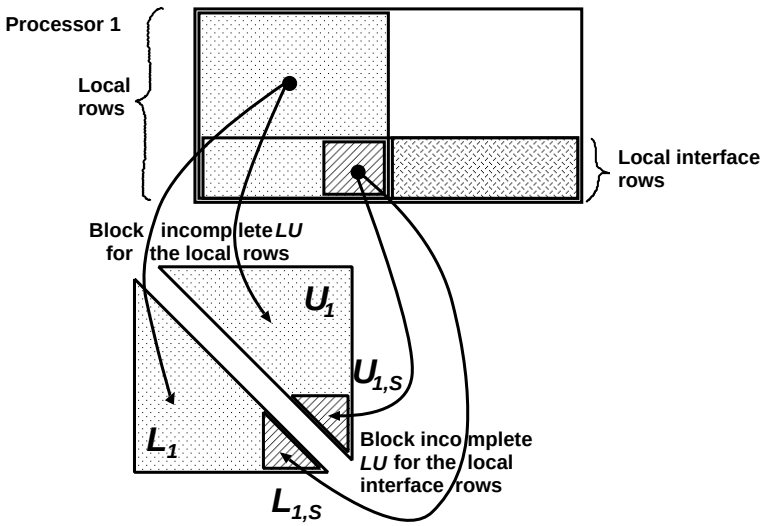

Figure 4. *Principle of preconditioning within the DSC algorithm.*

tem [5]:

$$y_i + S_i^{-1} \sum_{\text{Neighbours } j} E_{ij}\, y_j = S_i^{-1}(g_i - E_i\, B_i^{-1} f_i) \ . \tag{4}$$

3.3 Preconditioning

Fig. 4 illustrates the principle of preconditioning within the DSC algorithm per processor. The outer iteration from Fig. 3 is preconditioned per processor by a block incomplete LU decomposition with threshold (ILUT) [4] of the local diagonal block ($L_1\, U_1$ in Fig. 4). For preconditioning the inner iteration, a block ILUT for the local interface rows only is exploited. This factorization need not be computed but can be used from the lower right part of the decomposition for the outer iteration ($L_{1,S}\, U_{1,S}$ in Fig. 4).

Mathematically speaking, we perform a block factorization of A_i on processor i using the splitting from (2):

$$A_i = \begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix} = \begin{pmatrix} B_i & 0 \\ E_i & S_i \end{pmatrix} \begin{pmatrix} I & B_i^{-1} F_i \\ 0 & I \end{pmatrix} \ . \tag{5}$$

We then assume that we have the LU decomposition $S_i = L_{i,S}\, U_{i,S}$ of the local Schur complement. With this, we formulate the LU factorization

$$L_i\, U_i = \begin{pmatrix} L_{i,B} & 0 \\ E_i\, U_{i,B}^{-1} & L_{i,S} \end{pmatrix} \begin{pmatrix} U_{i,B} & L_{i,B}^{-1} F_i \\ 0 & U_{i,S} \end{pmatrix} \tag{6}$$

with $B_i = L_{i,B}\, U_{i,B}$ the LU decomposition of B_i. By transforming the right hand

side of (6) into

$$\left[\begin{pmatrix} L_{i,B} & 0 \\ E_i\, U_{i,B}^{-1} & L_{i,S} \end{pmatrix} \begin{pmatrix} U_{i,B} & 0 \\ 0 & U_{i,S} \end{pmatrix}\right] \begin{pmatrix} I & U_{i,B}^{-1} L_{i,B}^{-1} F_i \\ 0 & I \end{pmatrix} =$$

$$\begin{pmatrix} B_i & 0 \\ E_i & S_i \end{pmatrix} \begin{pmatrix} I & B_i^{-1} F_i \\ 0 & I \end{pmatrix}$$

we find after comparison with (5) that $L_i\, U_i$ is an LU factorization of A_i. The other way round, we also see the practical advantage from (6) that the LU factorization $S_i = L_{i,S}\, U_{i,S}$ of the local Schur complement has not to be computed explicitly if we already have an LU factorization of the local diagonal block A_i.

If we perform incomplete decompositions we get an approximate, preconditioned Schur complement system with the approximation $\tilde{S}_i$ of the local Schur complement S_i (compare with (4)):

$$y_i + \tilde{S}_i^{-1} \sum_{\text{Neighbours } j} E_{ij}\, y_j = \tilde{S}_i^{-1} (g_i - E_i\, B_i^{-1} f_i) \ . \tag{7}$$

3.4 Repartitioning and Reordering

The distributed sparsity pattern of the matrix can be represented as a distributed graph with nodes and edges. Graph repartitioning can then be used to reduce the number of couplings between the distributed matrix row blocks. In graph theory formulation, the reduction is done by a minimization of the number of edges cut in the graph. This goal of graph partitioning corresponds to a minimization of the number of interface unknowns in the DSC algorithm, and thus problem (7) is made very small. For graph partitioning, we use the ParMETIS software from the University of Minnesota [2]. Since ParMETIS requires an undirected graph as input the non-symmetric pattern of the matrix has to be symmetrized for the matrix graph construction.

For local, incomplete decompositions, we use METIS nested dissection reordering to reduce fill-in into the factors [2]. Nested dissection reordering usually generates a similar sparsity pattern for the local diagonal blocks A_i on each processor i. This results in similar fill-in for each ILUT and thus supports load balancing.

4 Results

The following experiments were performed on NEC's PC cluster GRISU (32 2-way SMP nodes with AMD Athlon MP 1900+ CPUs, 1.6 GHz, 1 GB main memory per node, Myrinet2000 interconnection network between the nodes).

For the following experiments, the equation systems `ccp` and `circ2a` which stem from simulations of NEC circuits, the systems `circuit_2` and `circuit_3` from Philips circuits [1] as well as the systems `hcircuit` and `scircuit` from Motorola circuits are used. The latter four systems are available from the Matrix Market (`http://www.cise.ufl.edu/~davis/sparse/{Bomhof, Hamm}`)

Table 1. *Original and reduced systems: Partitioning into eight sub-domains.*

	Original matrix			Reduced matrix		
Matrix	Order	Non-zeros	#If vars	Order	Non-zeros	#If vars
ccp	89556	760630	36077	89378	603753	2010
circ2a	482969	3912413	422900	482963	2750390	402
circuit_2	4510	21199	1265	1262	9702	629
circuit_3	12127	48137	2688	7607	34024	186
hcircuit	105676	513072	2490	105592	506970	975
scircuit	170998	958936	1874	170850	958370	1700

Table 2. *DSC times on 8 GRISU processors: Effect of ordering and partitioning.*

	Partitioning + ordering		Partitioning only	No permutation	
Matrix	#If vars	Time/s	Time/s	#If vars	Time/s
ccp	2010	0.51	0.97	65856	27.48
circ2a	402	0.57	2.02	8752	13.51
circuit_2	629	0.04	0.05	1245	0.07
circuit_3	186	0.25	0.25	1152	0.65
hcircuit	975	0.23	117.99	92654	9.27
scircuit	1700	2.16	29.69	112444	807.30

Table 1 presents orders and numbers of non-zeros for the original and the reduced matrices as well as the number of interface variables (if vars) for partitioning into eight sub-domains. Repartitioning is applied. The results in table 1 show a distinctly smaller number of interface variables in the case of the reduced systems. Repartitioning is effective in this case and results in very low costs for the inner iteration from Fig. 3.

In table 2, times of the DSC method on eight GRISU processors with both repartitioning and reordering, with repartitioning only, and without repartitioning and reordering are displayed for the reduced systems. The number of interface variables is given for the first and last scenario; for the second, the number is the same as for the first one. The shortest times by far in table 2 are achieved for the DSC method with repartitioning and reordering. Repartitioning keeps the interface system small, reordering usually reduces fill-in and improves the load balance of the processors (see 3.4).

Table 3 shows executions times of the DSC method on 1 to 16 GRISU processors for the largest reduced test case circ2a.

In Fig. 5 total, sequential execution times on GRISU of NEC's circuit simulator MUSASI with the original direct solver (Schur complement method) and the developed iterative DSC algorithm are displayed for a medium size circuit problem (transient analysis) which results in equation systems of order 24705. The left times do not include the time for the setup phase while the right times do contain this time. The most expensive operations in the setup phase are the symbolic factoriza-

Table 3. *DSC times on GRISU: Scalability.*

	Time/s on p processors					
Matrix	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 12$	$p = 16$
`circ2a`	2.19	1.89	1.00	0.57	0.43	0.39

tion with Markowitz ordering which are necessary and state-of-the-art for a direct solver, but can be left out for the iterative DSC solver developed.

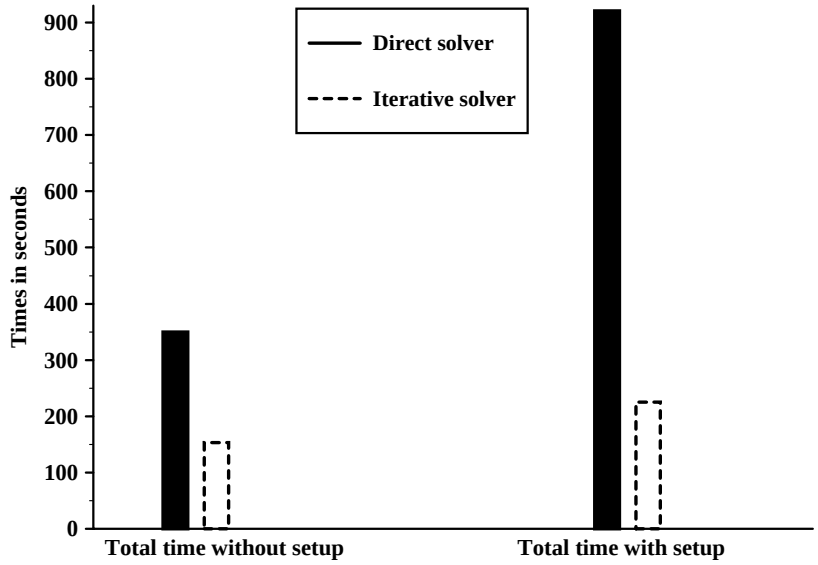

Figure 5. *Sequential execution times of NEC's circuit simulator MUSASI.*

The left times in Fig. 5 show a speedup of 2.3 of the DSC solver over the original direct solver. With setup phase, the iterative solver outperforms the direct one by even a factor of 4.1 since the costly symbolic factorization with Markowitz ordering can be skipped. From first tests, a similar ratio of the simulation times with direct and iterative solver can be expected for the parallel case, but the integration of the iterative DSC solver into the parallel circuit simulator MUSASI is not complete yet.

5 Conclusions

For equation systems from real problems, we demonstrated that the DSC algorithm combined with repartitioning and reordering is a well suited iterative solver for circuit simulation. For the performance of iterative solvers, a system reduction by the removal of trivial equations is crucial. Moreover, a distinct reduction of simulation time can be expected if commonly used direct methods in circuit simulation codes are replaced with the iterative DSC method presented.

Bibliography

[1] C. W. Bomhof and H. A. Van der Vorst, *A parallel linear system solver for circuit simulation problems*, Numerical Linear Algebra with Applications, 7 (2000), pp. 649–665.

[2] G. Karypis and V. Kumar, *ParMETIS: Parallel graph partitioning and sparse matrix ordering library*, Tech. rep. # 97-060, University of Minnesota, 1997.

[3] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C*, 2nd ed., Cambridge University Press, 1994.

[4] Y. Saad, *Iterative Methods for Sparse Linear Systems*, PWS, Boston, 1996.

[5] Y. Saad and M. Sosonkina, *Distributed Schur complement techniques for general sparse linear systems*, SISC, 21 (1999), pp. 1337–1356.

[6] H. Van der Vorst, *Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 631–644.