Supernodal Sparse Linear Equations Solver – Limitations and Alternatives

Marek Stabrowski

***Abstract* —Supernodal technique, used in the field of sparse linear equation solvers, introduces dense kernel for speeding-up the computations. SuperLU solver is the most sophisticated and efficient example of this technique. The research reported here shows that supernodal solver is really fast, but this spectacular speed is obtained through abandoning of numerical stability. It has been shown that SuperLU solver fails in whole class of practical real-world problems. An attempt to characterize and diagnose this problems in terms of matrix parameters has been made. Pointer solver, being stable numerically and fairly fast alternative for supernodal solver is presented quite comprehensively.**

I. INTRODUCTION

The algorithms and the software for sparse matrices processing make extensive use of matrices sparsity. It may be observed, on most general level, that in the sparse approach only the nonzeros are stored, together with diversified information on location of these nonzeros. Such approach reduces spectacularly the consumption of memory. However processing of the nonzeros is accompanied by the overhead of locating currently used items and by the overhead of fill-in. The last phenomenon results from popping-up of nonzeros in some previously unoccupied places, for example as a consequence of computing LU decomposition of the matrix.

The overhead of sparse structure processing may be quite significant. It offsets (luckily only partially) the benefits of reduced memory consumption. These considerations have led to the idea of hybrid approach, i.e. mixing of sparse and dense approach. In dense matrix algorithms the processed elements are picked in some precisely defined (usually serial) order and no fill-in occurs due to matrix density.

II. UNSYMMETRIC SUPERNODAL SOLVER

It is believed that most successful introduction of dense kernel technique has been achieved in supernodal unsymmetric solver with partial pivoting [1]. Basic idea of a supernode consists in grouping together the matrix columns with the same nonzero structure and dense region around main diagonal. Four types of supernodes with slightly different dense region and different row structures have been defined by the authors of supernodal unsymmetric solver. Two types of supernodes are shown in fig. 1

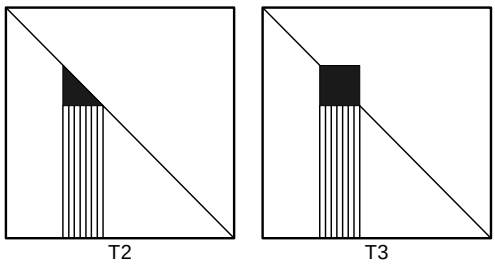

Fig. 1. Two types of supernodes. Solid black area - dense region, vertical lines - columns with the same nonzero structure.

Introduction of supernodes offers several quite obvious benefits:

1) Due to identical sparsity pattern in a set of columns no fill-in occurs there.

2) Dense region around main diagonal may be processed with the aid of dense numerical kernel.

3) No fill-in occurs in the supernodes with identical row sparsity pattern (not shown in fig. 1).

All these factors speed-up processing of the matrix. Detection, forming and processing of unsymmetric supernodes have been analyzed comprehensively by the authors of this idea in excellent paper on supernodal solver SuperLU with partial pivoting [1]. Extensive numerical experiments have proved, according to method's authors, the superior performance of supernodal solver. The tests of this solver have been performed on large set of diversified sparse matrices. Two parameters characterizing unsymmetric sparse matrices have been introduced:

- *StrSym* is the fraction of nonzeros matched by nonzeros in symmetric locations.
- *NumSym* is the fraction of nonzeros matched by equal values in symmetric locations.

For the test set of the matrices [1, 8] the *StrSym* and *NumSym* parameters ranged from approximately 1.0 down to almost 0.0. Matrix dimensions reached 80000 and the areas of application encompassed circuit simulation, economics, chemical engineering and many others. SuperLU solver performance has been in most cases distinctly better (time, memory requirements) than older multifrontal UMFPACK solver [1].

However this fine and sophisticated piece of software seems to be undertested, despite large matrix collection engaged in the tests. Table I presents the results of the tests performed with the set of unsymmetric matrices provided by Friedrich Grund from Bayer AG [8]. These matrices are available, among others, from University of Florida collection. SuperLU solver code, for the purposes of current research, has been downloaded from NIST repository of the open source software.

TABLE I.

Author is with the Institute of Computer Science, Lublin University of Technology, ul. Nadbystrzycka 36B, 20-618 Lublin, Poland, e-mail: *mmst@bravo.pol.lublin.pl*

RESULTS OF SAMPLE UNSYMMETRIC SPARSE MATRICES PROCESSING BY SUPERLU SOLVER; ERROR INFO = NUMBER OF THE STEP, WHERE THE SOLVER FAILED

matrix name	dimen-sion	*StructSym*	*NumSym*	error info uncom-press.	error info com-pressed
Gru30	3268	0.0068	0.0011	1043	896
Gru31	3008	0.0068	0.0011	1	1
Gru32	3268	0.0068	0.0011	1043	896
Gru33	3008	0.0068	0.0011	1	1
Gru34	3083	0.0213	0.0106	1	2
Gru35	13436	0.0026	0.0002	1	6
LHR01	1477	0.0087	0.0013	OK	OK
LHR02	2954	0.0087	0.0013	OK	OK
LHR04	4101	0.0162	0.0011	OK	OK
LHR71	70304	0.0016	0.0001	OK	OK
ORANI678	2529	0.0729	0.0024	OK	OK
MAHINDAS	1258	0.0302	0.0138	OK	OK
WEST2021	2021	0.0039	0.0007	OK	OK
VAVASIS3	41092	0.0010	0.0000	OK	OK
PSMIGR1	3140	0.4817	0.0162	OK	OK

No test matrix from Bayer collection used in current research (table I) has been successfully decomposed. SuperLU solver failed in assorted stages of LU decomposition. Matrix compression option included in SuperLU solver (compression idea by Mayoh [3]) has been tried but without success. The SuperLU solver failed in several cases in other stages of LU decomposition.

One may be tempted to infer that Bayer matrices are singular. However other, more classical, solvers (e.g. [6]) decompose these matrices without any problems.

Two problems/questions arise. First of all it may be interesting to devise a method for quick problem diagnosis – is the matrix decomposable with SuperLU solver or not. Next – what are the reasons for SuperLU failure in some cases; is it coding error or some deficiency of the supernodal method.

III. STRUCTURAL CHARACTERICTICS OF SPARSE MATRICES

In order to get some insight into the difference between the Bayer matrices and other ones lets have a look at nonzero patterns. Figure 2 presents the structure of Gru30 matrix. Figure 3 shows the structure of LHR01 matrix used in original SuperLU tests. Both matrices are quite large (dimension from 1500 to approximately 3000) and therefore finer details of nonzero structure are missing in these figures.

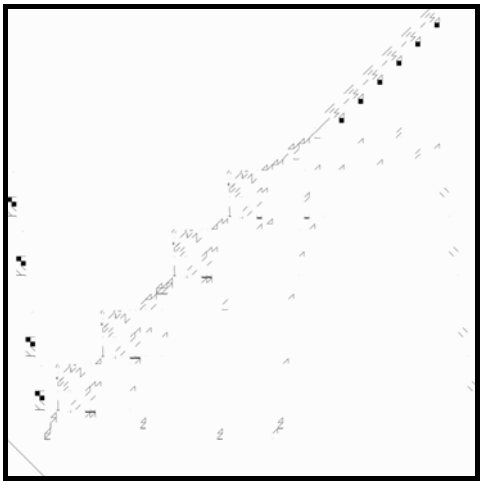

Fig. 2. Sparsity pattern of Gru30 matrix; nonzeros are located around the diagonal perpendicular to the main one.

Despite these limitations, it is evident that these structures are completely different. The nonzeros of Gru30 matrix are scattered in distinctly larger distances from main diagonal than the nonzeros of LHR01 matrix. Nonzeros of LHR01 are coarsely ordered along main diagonal, whereas in Gru30 they are located rather perpendicularly to main diagonal.

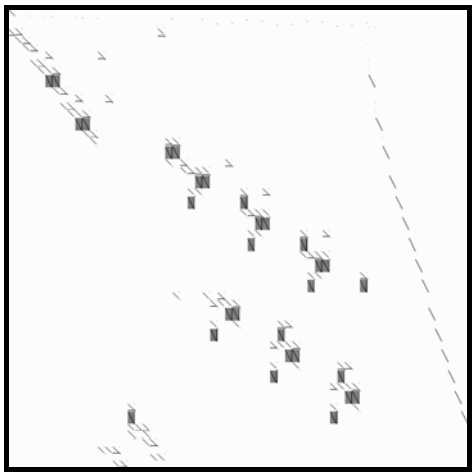

Fig. 3. Sparsity pattern of LHR01 matrix; nonzeros located roughly around or quite close to main diagonal

These qualitative observations may be supplemented with quantitative sparse matrix parameters. Two characteristic parameters have been introduced by the authors and developers of SuperLU package [1]. The definitions of *StrSym* and *NumSym* have been quoted earlier in this paper. However these parameters are the same or almost the same for the matrices decomposable and not decomposable by SuperLU solver.

Following two characteristic parameters seem to be adequate for describing the differences between Gruxx matrix series and other test matrices. The first one is the mean relative distance between nonzeros and main diagonal (standard diagonal distance), defined as:

$$d_n = \left(\sum_{i=1}^{n} \left(\sum_{\substack{j=1 \\ a_{ij} \neq 0}}^{n} \left(|j-i| / n \right) \right) \right) \Big/ n_{nz} \qquad (1)$$

It follows from this equation that the distance is referenced to matrix dimension and that it is measured horizontally.

Second parameter may be called orthogonal or perpendicular mean distance. It is measured between nonzeros and the diagonal perpendicular (orthogonal diagonal distance) to the main one:

$$d_o = \left(\sum_{i=1}^{n} \left(\sum_{\substack{j=1 \\ a_{ij} \neq 0}}^{n} \left(\left| j - (n-i) \right| / n \right) \right) \right) \Big/ n_{nz} \qquad (2)$$

The last two parameters, along with *StructSym* and *NumSym*, for selected set of matrices are shown in table II.

TABLE II.

CHARACTERISTIC PARAMETERS OF THE TEST SET OF MATRICES

matrix name	dimension	*Struct-Sym*	*Num-Sym*	relative distance d_n	relative ortho. distance d_o
Gru30	3268	0.0068	0.0011	0.531	0.166
Gru31	3008	0.0068	0.0011	0.575	0.139
Gru32	3268	0.0068	0.0011	0.531	0.166
Gru33	3008	0.0068	0.0011	0.575	0.139
Gru34	3083	0.0213	0.0106	0.346	0.112
Gru35	13436	0.0026	0.0002	0.395	0.142
LHR01	1477	0.0087	0.0013	0.142	0.342
LHR02	2954	0.0087	0.0013	0.071	0.500
LHR04	4101	0.0162	0.0011	0.152	0.325
LHR71	70304	0.0016	0.0001	0.041	0.408
ORANI678	2529	0.0729	0.0024	0.175	0.500
MAHINDAS	1258	0.0302	0.0138	0.374	0.280
WEST2021	2021	0.0039	0.0007	0.216	0.421
VAVASIS3	41092	0.0010	0.0000	0.182	0.380
PSMIGR1	3140	0.4817	0.0162	0.279	0.375

It can be easily observed that relative distance parameters clearly divide the test set of matrices into two groups. The first group is composed of Gruxx series of matrices. It is characterized by relatively high value of standard diagonal distance (typically above 0.5) and low value of orthogonal diagonal distance (below 0.2). The reverse is true for the rest of matrices – low standard diagonal distance and high orthogonal distance. Computation of standard and orthogonal diagonal distance is inexpensive in the sense of timing. It seems to be an usable auxiliary tool for classification of matrices into the set decomposable or not decomposable with the aid of SuperLU. Final judgement must rely on an attempt to perform LU decomposition.

IV. ALTERNATIVE SPARSE SOLVER WITH ENHANCED NUMERICAL STABILITY

Two solvers have been developed in the course of current research, as more stable alternatives for SuperLU solver. Both solvers implement "array of pointers" idea [2, 6]. First one has been developed in object oriented style using C++ language (GNU compiler package). Second solver is structural counterpart of the first one. It has been developed through some sort of downgrading C++ pointer solver to C language. This solver version has been used for comparing numerical and code generating efficiency of C++ compiler vs. C compiler [7].

Several thorough improvements in basic version of pointer solver [6] have been introduced in the course of current research. First of all, during LU decomposition, **L** matrix is generated as a separate entity. In order to speed-up access to its elements it is ordered columnwise. Second set of improvements is related to matrix **U**. This matrix replaces original rowwise matrix **A**. The rows of matrix **U** (despite fill-in) become shorter in the course of forward elimination. In single elimination step at first a product of pivot row and element of already assembled column of matrix **L** is computed. Next the memory allocated to current **A** matrix row is released and the pointer to this memory is assigned to this temporary structure. Thus row contents shifting is replaced by much faster dynamic memory allocation and deallocation.

Row swapping during pivoting is handled locally in pointer solver. In this case simple interchange of pointer nodes contents is sufficient.

V. COMPARISON OF SUPERNODAL AND POINTER SOLVER

Three solvers have been tested:

- supernodal solver SuperLU [1] available as open source software in C language;
- pointer solver developed in C++ language;
- pointer solver almost identical (algorithms) with the previous one, but downgraded to C language;

The tests have been performed on Linux RedHat 7.1 platform running on Pentium 800 MHz. The software has been compiled with the aid of C/C++ GNU compiler package. Both original uncompressed matrices and compressed versions (Mayoh's transposed matrix sum [3] and RCM method) have been used. In the case of SuperLU solver compression is available and consequently has been used in these tests.

One of most interesting factors in assessment of solver performance and usability is the computation time. Table III shows LU decomposition times for uncompressed matrices. SuperLU solver would be clear winner if "Gruxx" set of matrices have been excluded from test data. Supernodal solver outperforms pointer solver introduced here by the factor of 7 to 8 with respect to processing time. However in the case of "Gruxx" matrices SuperLU solver fails to finish correctly the LU decomposition.

TABLE III

LU DECOMPOSITION TIME (SEC) FOR UNCOMPRESSED MATRICES; PENTIUM 800 MHZ, LINUX REDHAT 7.1 OPERATING SYSTEM, GNU C/C++ COMPILER

	No. of equa-tions	No. of non-zeros	Pointer solver C++	Pointer solver C	SuperLU solver
WATT1	1856	11360	1	1	1
WATT2	1856	11550	1	1	1
Gru30	3268	27836	15	16	error
Gru31	3008	27576	3	3	error
Gru32	3268	27836	15	15	error
Gru33	3008	27576	4	4	error
Gru34	3083	21216	1	1	error
Gru35	13436	94926	12	11	error
LHR01	1477	18592	8	8	1
LHR02	2954	37206	16	16	1
LHR04C	4101	82682	290	288	32
LHR07C	7337	156508	1763	1779	245

The situation in the case of preliminary matrix compression (table IV) changes in some aspects. As in the case of uncompressed matrices SuperLU solver is the fastest one, if the LU decomposition is feasible. Also "Gruxx" matrices cannot be decomposed with the aid of this solver. Matrix compression reduces processsing time of all solvers in the case of "LHRxx" matrix series with reduction factor above 20 in some cases. For "Gruxx" matrix series, compression influence is surprisingly reversed. Sometimes very prominent processing time expansion is observed.

TABLE IV

LU DECOMPOSITION TIME (SEC) FOR COMPRESSED MATRICES; PENTIUM 800 MHZ, LINUX REDHAT 7.1 OPERATING SYSTEM, GNU C/C++ COMPILER

	No. of equa-tions	No. of non-zeros	Pointer solver C++	Pointer solver C	SuperLU solver
WATT1	1856	11360	1	1	1
WATT2	1856	11550	1	1	1
Gru30	3268	27836	30	32	error
Gru31	3008	27576	22	24	error
Gru32	3268	27836	30	32	error
Gru33	3008	27576	23	24	error
Gru34	3083	21216	5	5	error
Gru35	13436	94926	269	273	error
LHR01	1477	18592	2	2	1
LHR02	2954	37206	10	9	1
LHR04C	4101	82682	26	24	7
LHR07C	7337	156508	80	82	23

In the case of bandwidth compressed matrices pointer solver is slower by the factor of 3 to 10 (lower values correspond to larger matrices or compressed ones) than SuperLU solver. However it seems to be a justified price for reliability and meaningful results.

After this semi-quantitative analysis it is rather easy to explain the failure of SuperLU solver in the case of "Gruxx" series. For matrices of this type very extensive pivoting is necessary. SuperLU solver has traded numerical stability (equivalent to extensive pivoting) for speed.

VI. CONCLUSIONS

Main purpose of this paper consists in pointing out the limitations of supernodal approach to partial pivoting in sparse unsymmetric matrices. The SuperLU package, implementing this method should be used with some caution.

New pointer-type sparse solver, using object oriented programming technique, has proved its usefulness in the field of large unsymmetric linear equation systems. It features better numerical stability than competitive supernodal solver at the expense of acceptably longer processing time. Tests of C++ and C language versions have shown no practical differences in performance. Introduction of C++ language enables compacting source code to single version (in contrast to single precision, double precision and complex versions in C solvers) due to template facility of the language [4, 5].

Two new parameters have been introduced for classification of sparse matrices – standard mean diagonal distance of nonzeros and new parameter – mean orthogonal diagonal distance. Applied jointly they help to detect numerical difficulties and eventual problems with less robust solvers.

REFERENCES

[1] J.W. Demmel, S.C. Eisenstat, J.R. Gilbert, X.S. Li, J.W.H. Liu, "A supernodal approach to sparse partial pivoting", SIAM J. Matrix Anal. Appl., vol. 20, No. 3, pp. 720-755.

[2] K.S. Kundert, „Sparse matrix techniques" in „Circuit Analysis, Simulation and Design", Albert Ruehli (editor), North Holland, 1986.

[3] B.H. Mayoh, „A graph technique for inverting certain matrices", Mathematics of Computation, vol. 19 (1965), pp. 644-646.

[4] S. Pandit, S. A. Soman, S. A. Khaparde, „Design of generic direct sparse linear system solver in C++ for power system analysis", IEEE Transactions on Power Sysytems, vol. 16 (2001), iss. 4, pp. 647-652.

[5] J.G. Siek, „A Modern Framework for Portable High Performance Numerical Linear Algebra", Master Sc. Thesis, Notre Dame University, Indiana 1999.

[6] M.M. Stabrowski, „Software and hardware optimization of unsymmetrical sparse linear equation solvers", Proceedings of XII ISTET Conference , Warsaw 2003, vol. II, pp.159-162.

[7] T.L. Veldhuizen, M.E. Jernigan, „Will C++ be faster than Fortran", in "Scientific Computing in Object-Oriented Parallel Environments". ISCOPE, December 1997.

[8] S.E. Zitney, J.U. Mallya, T.A. Davis, M.A. Stadherr, „Multifrontal vs frontal techniques for chemical process simulation on supercomputers", Computers in Chemical Engineering, vol. 20 (1996), pp. 614-646.