A Fast "Parallelized" Single Pass Bandwidth Optimizer for Sparse Matrices

Detlev Zundel, Willi Schönauer
Rechenzentrum der Universität Karlsruhe
D-76128 Karlsruhe, Germany
E-mail: schoenauer@rz.uni-karlsruhe.de

Keywords: bandwidth, sparse matrix, (I)LU preconditioning, parallelization
AMS: 65F50, 65F05, 65F10, 65Y05

Abstract

This paper is a report of the diploma thesis of D. Zundel [10] and further work of the authors. For (I)LU preconditioning in the LINSOL program package several bandwidth optimizers have been implemented and compared: the reverse Cuthill-McKee (RCM), the Gibbs, Poole and Stockmeyer (GPS) and the Schönauer single pass (SSP) algorithms. These algorithms are inherently recursive, but they work on data stored in a distributed memory parallel computer.

1 Motivation

The background for this work is the implementation of an (I)LU preconditioner for the LINSOL [4, 3, 8] program package designed to solve large sparse linear systems $Ax = b$ on distributed memory parallel computers. LINSOL uses fast iterative solvers giving an approximation x' to the exact solution x satisfying a given upper bound of the residual $r = Ax' - b$.

The presently available fourteen generalized CG-solvers in LINSOL include PRES20 [6], BICO [6], AT-PRES also called CGNR [9], and a selection of polyalgorithms trying to automatically select the best one for a given problem. The matrix for the LU factorization in LINSOL is stored in single packed rows that are wrapped around over the processors for load balancing. However, it is well known that indexed storage schemes are inefficient on pipelined processors. Therefore the active part of the matrix is unpacked in a sliding buffer window as shown in fig. 1 and direct addressing with column pivoting is used. As fill-in occurs between the outermost diagonals, an efficient bandwidth optimization is essential.

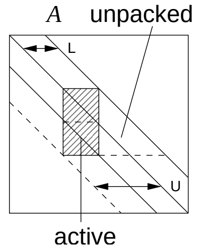


Figure 1: LU window.

The considered bandwidth optimizers are inherently recursive resulting in a serial stage in an otherwise parallel computation. Therefore speed will be more important than minor differences in the quality of the optimization and the total time for bandwidth optimization and factorization is decisive. The algorithms considered in this paper are the Reverse Cuthill-McKee (RCM) algorithm, the Gibbs-Poole-Stockmeyer (GPS) algorithm, both described in [2] and the Schönauer single pass (SSP) algorithm [10].

2 Introductory example

To illustrate the process of bandwidth optimization we will consider a trivial example that is typical for a matrix generated by a finite difference method. The 30×30 test matrix shown on the left of fig. 2 was subjected to the random permutation of the rows and columns given in table 1 to form the result shown on the right in the figure. The task now is to start with the permuted matrix and try to generate a permutation that regains the original pattern or at least the original bandwidth.

old	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
new	6	16	15	25	5	26	10	1	17	16	23	4	9	24	2
old	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
new	22	8	27	13	21	30	7	29	19	12	20	3	28	11	18

Table 1: Random permutation used in the example.

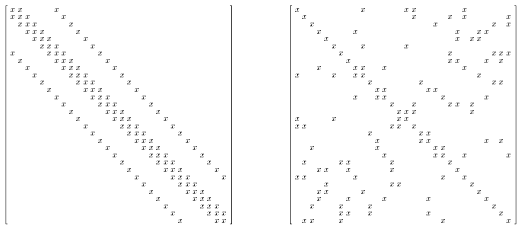


Figure 2: Original and scrambled example matrix.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1	x									x						x	x							x						
2		x															x					x		x				x		x
3			x						x											x						x	x			x

Figure 3: First three rows from the scrambled matrix.

The *bandwidth* of a matrix A is defined as $\beta(A) = \max_{a_{ij} \neq 0} |i - j|$ (distance of the outermost diagonal from the main diagonal). For the original matrix we have $\beta = 6$ and for the scrambled matrix $\beta = 28$. In order to motivate the working of the SSP algorithm let us see what information we can deduce from the new pattern. Fig. 3 shows the first three rows of the matrix. Let us identify row i with a vertex of index i and any non-zero entries in it as connections to adjacent vertices. This identification is well suited to our problem because the adjacency relation is invariant under the permutation, as can be easily seen. The vertices thus give us a representation of the matrix independent of permutations. It is also clear from fig. 3 that the bandwidth of the matrix corresponds to the maximal difference of adjacent indices to the row index i.

This stimulates working on the graph instead of working on the matrix rows. The situation for the first two rows is depicted in fig. 4. For non-symmetric matrices we effectively symmetrize the matrix by using only undirected edges in the graph. The problem of reducing the bandwidth of the matrix is thus converted into finding a renumbering of the graph vertices such that the index differences are minimal. It is known that this problem is NP-hard so that we are forced to use heuristic algorithms. For a formal introduction of the concepts used here see [10].

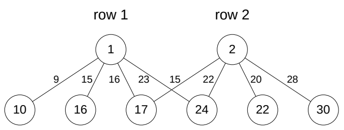


Figure 4: Subgraph for first two rows.

Let us now study how SSP proceeds in finding such a numbering. A suitable place to start is a vertex of minimal degree (number of elements in row), so SSP chooses vertex number 6. Fig. 5 shows the graph of the test matrix arranged such that the start vertex is the topmost vertex and the vertical position of a vertex corresponds to the time at which it is processed. Inter-level connections are omitted in the graph as they are unimportant for the numbering. This is also called a rooted level structure in which vertices belong to the same level if they are horizontal neighbours (in the spatial sense). So assigning the new number 1 to vertex number 6 we now have two candidates to assign the next numbers to, namely vertices 16 and 10. The two heuristics used by SSP are the fact that connections from the "early" side of a level to the "late" side of a level account for the maximal bandwidth and vertices with many "forward-connections" should be numbered last for the same reason. In the decision between the two candidate vertices, the first heuristic holds for both so we turn to the second heuristic and check the number of forward connections. As can be seen from fig. 5 vertex 16 has the fewest forward connections so we number it first followed by vertex 10. The candidates in the next level are now vertices number $1, 9, 15$ and 26.

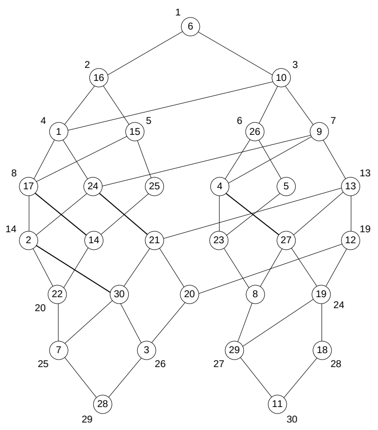


Figure 5: Level structure.

Now we use the first heuristic and concentrate on those candidates having a connection to the lowest numbered vertex in the previous level, namely vertex 16. This leaves vertices 1 and 15 to choose from. We now iterate the heuristic and check which one has a connection to the second-lowest numbered vertex in the previous level. Now we are left with only one candidate and assign the new number 4 to vertex 1, followed by number 5 for vertex 15. The first heuristic generally is iterated three times followed by the second heuristic which breaks ties arbitrarily if more than one candidate has a minimal number of "forward-connections". The result of the whole process is depicted in fig. 5 by the numbers outside the vertices representing the new numbers. Where the space is scarce the new numbers have been omitted but the new numbers ascend consecutively from left to right and from top to bottom. It should now be also clear why we choose a vertex of minimal degree as the start vertex. As was shown in [10] the proof originally given in [5] for RCM also applies to SSP and thus the reversed numbering is finally used to possibly improve the profile for no extra cost.

3 Differences between RCM and SSP

The "classical" RCM algorithm effectively searches over the permutations generated by different start vertices (given by a certain rule) to be more robust. This is based on the intuition that the selection of a start vertex completely determines the rooted level structure and thus gives a lower bound for the bandwidth attainable. The search, usually tunable by parameters, is very expensive in our context, however, so rather than trying to optimize the number of search candidates, SSP strictly uses only one pass through the graph of the matrix. In this sense SSP is minimal as would be a single pass RCM.

When numbering candidate vertices in a level RCM chooses vertices adjacent to the lowest numbered vertices in the previous level breaking ties by considering the degree of the vertex. SSP uses a slightly different heuristic by considering vertices adjacent to three, two and one low numbered vertices in the previous level, also breaking ties by considering the "forward-degree" of the vertex. Thus SSP tries to replace the expensive search for many start vertices with more refined local decisions in the numbering phase.

4 The GPS algorithm

Just like RCM and SSP, GPS works on the undirected graph of the matrix. In contrast to those two algorithms, it tries to improve on their performance by generating a more general level structure heuristically superior to the rooted level structures used by them. Starting with a minimal degree node, it builds the corresponding rooted level structure just like RCM and SSP do. For all vertices in the deepest level, the corresponding "reverse" level structures are generated and if a deeper structure can be found the process iterates. The two level structures found in this process are combined into a new level structure serving as a basis for the numbering process as depicted in fig. 6. For this each vertex is assigned a level pair consisting of the level

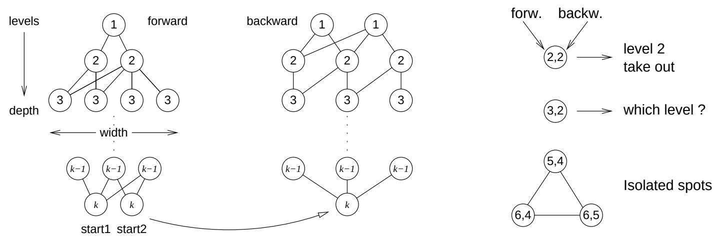
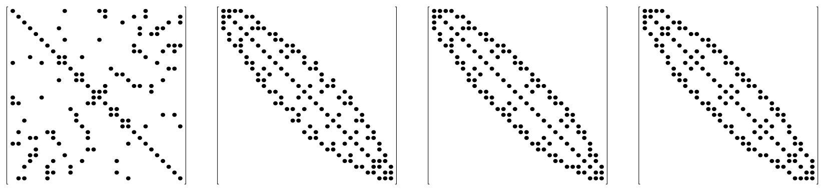


Figure 6: Level structures used by GPS.

numbers in the forward and the reverse level structure. The level numbers of the reverse structure are inverted such that both level numberings ascend in the same direction. As depicted in the right hand side of fig. 6 vertices for which both structures propose the same level are assigned directly. The remaining unassigned vertices form isolated spots in the graph. Each one of them is assigned as a whole according to the proposition yielding a smaller level width in the resulting structure. Based on this new structure, GPS now assigns numbers to the vertices basically like RCM, except that the non-rooted structure has to be handled differently in some cases.

5 Results for the test matrix

Fig. 7 shows the results of applying the three algorithms to the scrambled matrix of section 2. As is obvious none of them is able to reconstruct the original pattern although RCM and GPS yield a permutation with the same bandwidth as the unscrambled matrix.



Orig: $\beta = 28$ SSP: $\beta = 7$ GPS: $\beta = 6$ RCM: $\beta = 6$

Figure 7: Comparison for the test matrix.

6 Parallelization

The parallelization of the algorithms, SSP in particular, deserves some closer study. The renumbering in all of them is inherently recursive barring an efficient parallelization. Thus the "parallelization" that we refer to is on data stored in distributed memory. One processor is active in the numbering phase until it has $[n/p]\{+1\}$ rows, then the next processor becomes active. In this phase the communication pattern is not known in advance as it depends on the structure of the matrix. This is solved by a sophisticated data exchange in which the passive processors are busy waiting for instructions on which data has to be sent to which processor. This data is collected in one block for each level and processor to minimize the communication overhead.

7 Examples

In this section the application of the three algorithms to several examples will be presented. The comparison of the implementations can be used as a comparison of the algorithms because all three algorithms were implemented by us in the same quality. This does not imply that they are the fastest possible implementations, but comparing them is more expressive than comparing a new implementation of an algorithm against library routines that have been optimized over years.

The measurements were done on an IBM/SP2 (375 MHz) on 16 processors and are given in seconds. Note that for most of the examples a preconditioning would not be needed as the iterative solvers in LINSOL converge. However, it is interesting to study the results of the optimizations. The "classical" RCM algorithm employing a search over many start vertices was not able to complete for the examples in under 4 hours of computing time. A single-pass version of it produces comparable results to SSP, which turn out to be quite good. This is the reason why no parameter tuning experiments with RCM have been made and why it does not appear in the reported results.

7.1 Structural analysis

The example shown in fig. 8 was generated by us with the commercial MSC/NASTRAN FEM package. It stems from static load analysis of a console and has 44040 unknowns. As we can see from table 2 the very efficient Bi-CGSTAB2 iterative solver converges for this example so as we already stated, a preconditioner is not needed. It is interesting, however, that the combination of SSP and LU is able to speed up the calculation.

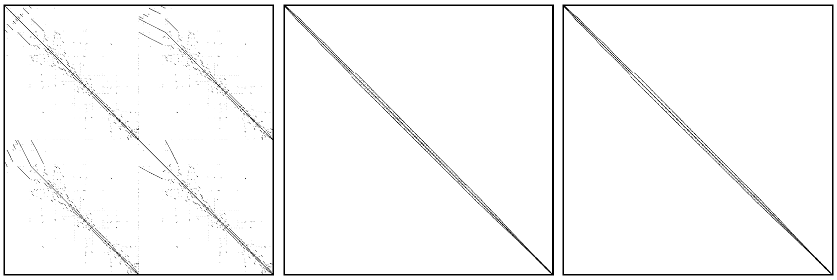


Figure 8: Console and SSP and RCM reorderings.

Method	$\beta(A_{opt})$	Opt. time	Fill-in	Total time
Bi-CGSTAB-2	33260	–	–	113.90
SSP + LU	683	22.62	16.0	70.99
GPS + LU	684	147.52	16.0	196.07

Table 2: Measurements for the console example.

7.2 Simulation of a buckling problem

The Simon-Venkat01 example in fig. 9 stems from the simulation of a buckling problem from NASA Ames Research Center and has 62424 unknowns [1]. As table 3 shows, again no preconditioning is needed and in this case is indeed drastically slower than the iterative solver. Remarkable is the fact that the time invested by GPS to achieve a better result pays off in the LU stage as the total time is less for GPS and LU compared to SSP and LU, despite the fact that SSP runs significantly faster.

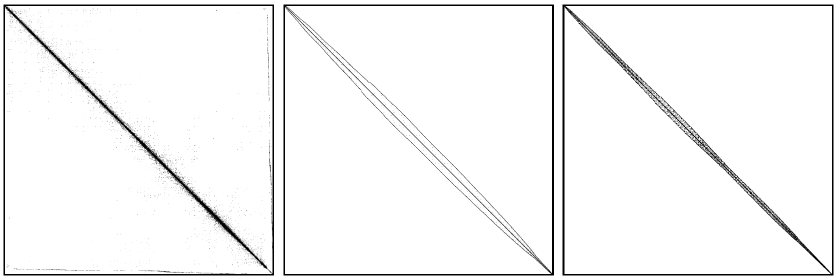


Figure 9: Simon-Venkat01 and SSP and RCM reorderings.

7.3 Another buckling problem

The Simon-Raefsky4 example in fig. 10 also stems from the simulation of a buckling problem from NASA Ames Research Center and has 19779 unknowns [1]. Table 4 shows that again no preconditioning is needed but the overhead for the full LU is not as significant as in the previous example.

Method	$\beta(A_{opt})$	Opt. time	Fill-in	Total time
Bi-CGSTAB-2	60323	–	–	1.97
SSP + LU	2451	52.76	49.5	194.41
GPS + LU	1443	80.64	49.5	180.93

Table 3: Measurements for Simon-Venkat01.

It is interesting to note that for this example the seemingly superior GPS algorithm is closely beaten by a much faster SSP.

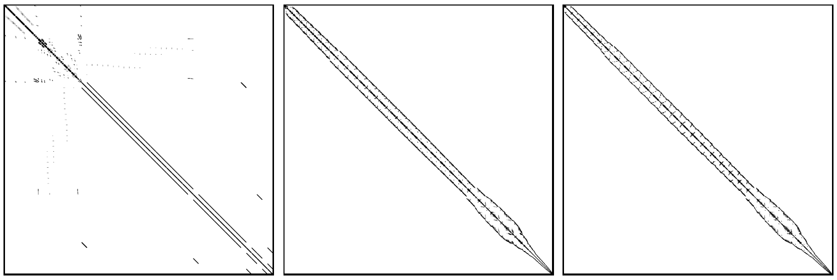


Figure 10: Simon-Raefsky4 and SSP and RCM reorderings.

Method	$\beta(A_{opt})$	Opt. time	Fill-in	Total time
Bi-CGSTAB-2	11786	–	–	26.76
SSP + LU	1050	18.21	17.4	48.47
GPS + LU	1053	68.67	17.2	94.66

Table 4: Measurements for Simon-Raefsky4.

7.4 A CFD-Discretization

The Bova-RMA10 example in fig. 11 stems from a CFD-Discretization of Charleston Harbor done at Nichols Research Corporation and has 46835 unknowns [1]. For the first time the fast Bi-CGSTAB2 iterative solver does not converge and the slow but robust ATPRES algorithm was used for the comparison. Table 5 shows that the preconditioning in this case is by a magnitude faster for both SSP and GPS. From the timings of the optimization phase we can also see that in this example GPS obviously makes good use of its parallelizable stage.

7.5 FDEM example

The r57963 example shown in fig. 12 has 57963 unknowns and stems from the FDEM [7] package after local unstructured mesh refinement. The 1-D domain decomposition applied to achieve load balancing in the parallel computation has the effect of a pre-optimization. Table 6 shows the results. The

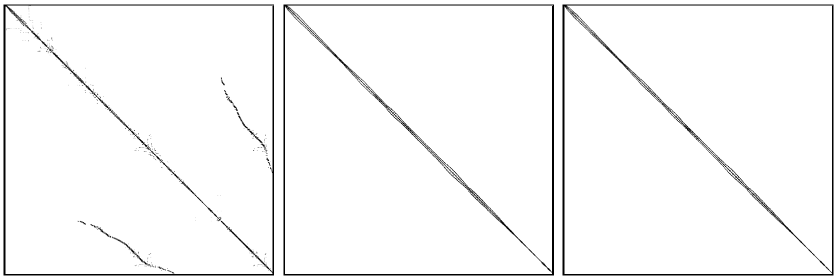


Figure 11: Bova-RMA10 and SSP and RCM reorderings.

Method	$\beta(A_{opt})$	Opt. time	Fill-in	Total time
AT-PRES	25141	–	–	6341.39
SSP + LU	603	21.44	17.4	64.13
GPS + LU	602	34.96	10.0	71.93

Table 5: Measurements for Bova-RMA10.

fast Bi-CGSTAB2 method again does not converge and the robust ATPRES method is slower by a magnitude than the application of the preconditioners. Here SSP is much faster than GPS without producing a significantly worse re-ordering. The total time once more confirms that in the parallelized environment speed in the bandwidth optimizing stage is more important than minor differences in quality.

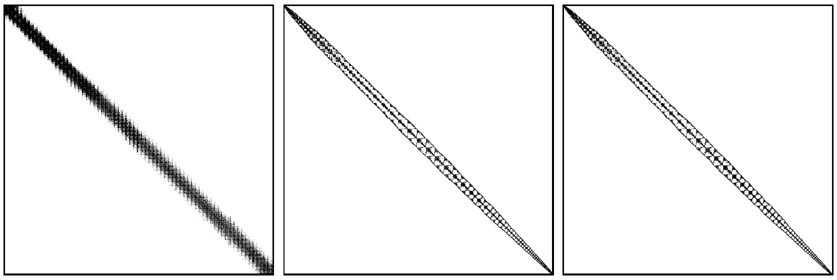


Figure 12: r57963 and SSP and RCM reorderings.

8 Concluding remarks

In a parallel environment the computation time for a recursive bandwidth optimization may need a significant part of the total solution time. In this paper three bandwidth optimizations have been compared under quite equal conditions: RCM, SSP and GPS. The "classical" RCM is too expensive for such large problems. The fastest algorithm is the SSP (a single-pass RCM would behave quite similar). The most robust GPS needs significantly more computation time and is more efficient only where it reduces the bandwidth

Method	$\beta(A_{opt})$	Opt. time	Fill-in	Total time
AT-PRES	3431	–	–	11881.84
SSP + LU	2196	72.88	83.6	1168.95
GPS + LU	2159	226.53	83.6	1324.17

Table 6: Measurements for r57963.

much better than SSP so that the LU factorization is considerably less expensive.

References

[1] T. Davis. University of Florida Sparse Matrix Collection, http://www.cise.ufl.edu/research/sparse/matrices ftp://ftp.cise.ufl.edu/pub/faculty/davis/matrices, NA Digest, vol. 92, no. 42, October 16, 1994, NA Digest, vol. 96, no. 28, July 23, 1996, and NA Digest, vol. 97, no. 23, June 7, 1997.

[2] Norman E. Gibbs, William G. Poole Jr., and Paul K. Stockmeyer. An algorithm for reducing the bandwidth of a sparse matrix. *SIAM Journal of Numerical Analysis*, 13(2):236–249, April 1976.

[3] Hartmut Häfner and Willi Schönauer. The integration of different variants of the (I)LU algorithm in the LINSOL program package. *to appear in Applied Numerical Mathematics.*

[4] Hartmut Häfner, Willi Schönauer, and Rüdiger Weiss. The program package LINSOL: Basic concepts and realization. *Applied Numerical Mathematics*, 30:213–224, 1999.

[5] Gerard Meurant. *Computer Solution of Large Linear Systems*, volume 28 of *Studies in Mathematics and its applications*. North Holland, 1999.

[6] Willi Schönauer. *Scientific Computing on Vector Computers*. North Holland, Amsterdam, 1987.

[7] Willi Schönauer and Torsten Adolph. The FDEM (Finite Difference Element Method) program package: Fully parallelized solution of elliptic and parabolic PDE's. *CD-ROM ECCOMAS 2000, Laboratori de Calcul de la Facultat d'Informatica de Barcelona (FIB, UPC), ISBN 84-89925-70-4, Barcelona, 2000, 20 pages.*

[8] Willi Schönauer and Hartmut Häfner. Numerical experiments to optimize the (I)LU preconditioning in the iterative linear solver package LINSOL. *to appear in Applied Numerical Mathematics.*

[9] Rüdiger Weiss. *Parameter-Free Iterative Linear Solvers*. Akademie Verlag, Berlin, 1996.

[10] Detlev Zundel. Implementation and comparison of three bandwidth optimizing algorithms on a distributed memory parallel computer. Diploma thesis. Available online at http://www.uni-karlsruhe.de/Uni/RZ/Forschung/Numerik/linsol/litps/bandoptim.ps.gz, 2000.