Distributed Schur complement solvers for real and complex block-structured CFD problems

A. Basermann[a,*], H.-P. Kersken[a]

[a]*German Aerospace Center (DLR), Simulation and Software Technology, Porz-Wahnheide, Linder Höhe, Cologne, Germany*

Abstract

At the German Aerospace Center, the parallel simulation system TRACE (Turbo-machinery Research Aerodynamic Computational Environment) has been developed for the calculation of internal turbo-machinery flows. The finite volume approach with block-structured grids for solving the Navier-Stokes equations results in large, sparse real or complex systems of linear equations. For the parallel iterative solution of these equation systems, FGMRes with Distributed Schur Complement (DSC) preconditioning for real or complex matrix problems has been integrated into TRACE. The DSC preconditioner replaces the original block-local preconditioning methods in the simulation system. Numerical and performance results of DSC methods are presented for typical TRACE problems on many-core architectures together with an analysis of the pros and cons of the complex problem formulation. The DSC preconditioned iterative solvers for the complex problem formulation distinctly outperform the solvers for the real formulation. Reasons are that the complex formulation results in lower problem order, more advantageous matrix structure, has higher data locality and a better ratio of computation to memory access.

Keywords:
Distributed Schur Complement preconditioning, parallel iterative solvers, complex arithmetics, many-core architectures, CFD, Navier-Stokes equations

*Corresponding author; phone: +49/2203/601-3326; fax: +49/2203/601-3070
Email addresses: `achim.basermann@dlr.de` (A. Basermann), `hans-peter.kersken@dlr.de` (H.-P. Kersken)

1. Introduction

At the Institute for Propulsion Technology of the German Aerospace Center (DLR), the parallel simulation system TRACE (Turbo-machinery Research Aerodynamic Computational Environment) has been developed specifically for the calculation of internal turbo-machinery flows. The finite volume approach with block-structured grids for solving the Navier-Stokes equations results in large, sparse real or complex systems of linear equations. For the parallel iterative solution of these equation systems, FGMRes [6] with Distributed Schur Complement (DSC) preconditioning [8] for real or complex matrix problems has been investigated. Fig. 1 shows simulation results generated with TRACE. Turbulent loss areas of a low pressure turbine within a real aeroengine are illustrated.

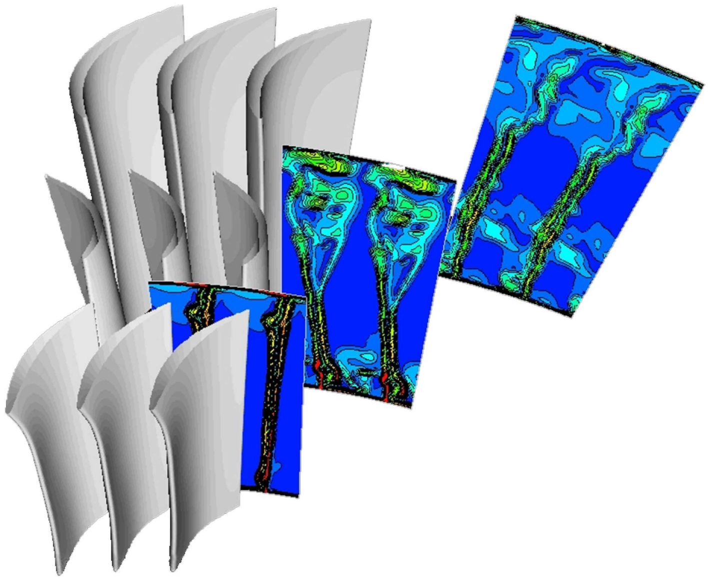

Figure 1: Turbulent loss areas of a low pressure turbine within a real aeroengine computed with TRACE.

The DSC method requires adaquate partitioning of the matrix problem since the order of the approximate Schur complement system to be solved depends on the number of couplings between the sub-domains. Graph partitioning with ParMETIS [5] from the University of Minnesota is suitable since a minimization of the number of edges cut in the adjacency graph of the matrix corresponds to a minimization of the number of the coupling variables between the subdomains. The latter determine the order of the approximate Schur complement system used for preconditioning. Since even the matrix pattern is non-symmetric for block-structured TRACE problems it has to be symmetrized so that the corresponding matrix adjacency graph becomes undirected und ParMETIS can be applied.

Matrix permutations like Reverse Cuthill-McKee (RCM), Minimum Degree (MD) or Nested Dissection are employed per sub-domain in order to reduce fill-in in incomplete factorizations which are part of the DSC preconditioner. These reordering methods together with a threshold strategy for incomplete factorizations decrease the costs of local approximate LU decompositions within the DSC method.

We present a parallel iterative FGMRes algorithm with DSC preconditioning [8] which achieves an accuracy of the solution similar to a direct solver but usually is distinctly faster for large problems. In [1], Basermann et al. demonstrated the superiority of the DSC preconditionier over (approximate) block-Jacobi preconditioning. Block-local preconditioning methods like block-Jacobi are the standard preconditioners in TRACE. In [2], Basermann et al. described and demonstrated the effect of matrix (re-)partitioning and reordering on the DSC algorithm in detail. Wubs and Thies investigated a hybrid direct/iterative solver approach involving Schur Complement methods for real fluid flow problems [12]. Compared with this algorithm, our method is fully iterative, and the software package is suitable for real and complex CFD problems. Hybrid direct/iterative solver approaches were also developed by Giraud and Haidar for real 3D convection-diffusion problems [3].

In this paper, we particularly discuss numerical and performance results of DSC methods for typical complex TRACE CFD problems on many-core architectures together with an analysis of the pros and cons of the complex problem formulation, e.g. regarding the ratio of calculation operations to memory accesses. In section 2, we describe the algorithmic background of the DSC method. In section 3, we present MATLAB results regarding the effect of reordering methods on preconditioner construction and iterative solution in real and complex arithmetics as well as performance measurements on a many-core cluster with the real and complex DSC software developed, including scalability studies.

2. The Distributed Schur Complement Method

In the following, techniques for the iterative solution with DSC preconditioning for CFD equation systems are sketched. More details, in particular on the theory, can be found in [8].

2.1. 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

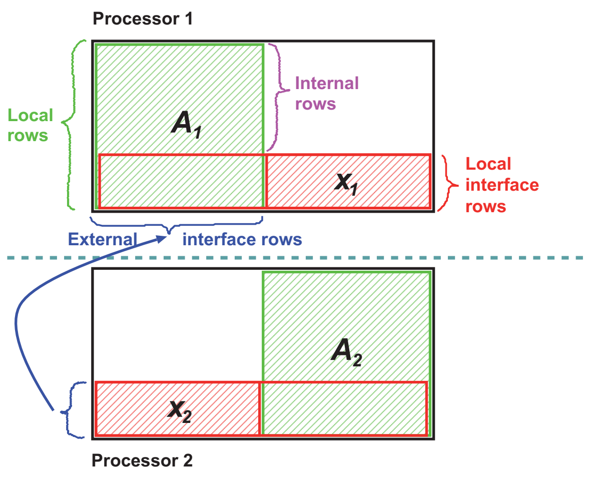

Figure 2: DSC definitions: Matrix distributed to two processors.

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 the sparsity pattern of TRACE CFD matrices usually is non-symmetric (due to special edge treatments in the block-structured grids) 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.

2.2. Algorithm

Fig. 3 gives a schematic survey of the DSC algorithm per processor. On

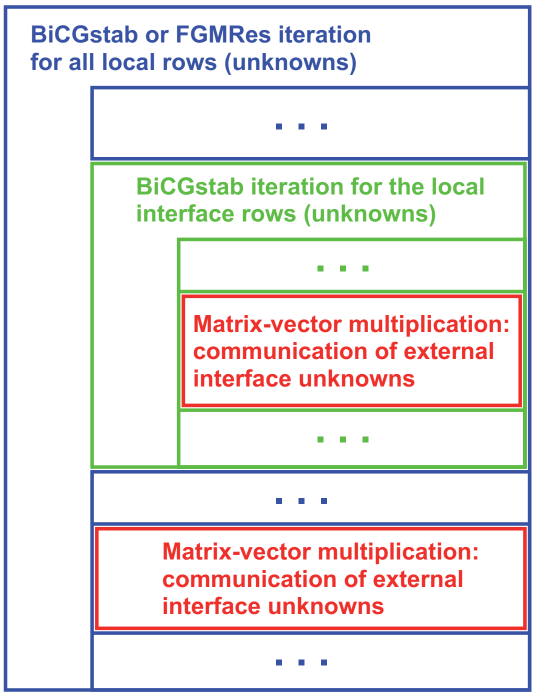
BiCGstab or FGMRes iteration
for all local rows (unknowns)
. . .
BiCGstab iteration for the local
interface rows (unknowns)
. . .
Matrix-vector multiplication:
communication of external
interface unknowns
. . .
. . .
Matrix-vector multiplication:
communication of external
interface unknowns
. . .

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

each processor an outer BiCGstab [11] or FGMRes [6, 7, 8] iteration is performed for all local rows (unknowns). 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 BiCGstab of FGMRes iteration, an inner BiCGstab 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.

As basic outer iterative method, FGMRes [6, 7, 8], a flexible variant of GMRES [7], is in principle better suited for the DSC algorithm than BiCGstab since the preconditioner (the inner BiCGstab iteration) may change in each iteration of flexible methods. The disadvantage of FGMRes is that it has higher storage requirements since a subspace has to be stored while BiCGstab applies recursions. On the other hand, the outer BiCGstab method theoretically requires a very precise solution of the interface system to make sure good convergence of the DSC algorithm. In case of outer FGMRes, only a few iterations for the interface system are very often sufficient to achieve good convergence of the DSC algorithm [8]. With new non-symmetric flexible iterative methods [10], the disadvantage of FGMRes regarding the subspace storage may be overcome while the advantages regarding flexible preconditioner application are preserved.

From the mathematical point of view, each processer i solves the following 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 [8] for details), and (1) is reformulated:

$$A_i = \begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix} \to$$

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

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

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

$S_i = C_i - E_i\, B_i^{-1} F_i$ 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 system [8]:

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

2.3. 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) [7] of the local diagonal block ($L_i\, U_i$ 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 ($L_{i,S}\, U_{i,S}$ in Fig. 4).

In case of an outer BiCGstab iteration, the solution for the interface unknowns has to be very precise. A possibility is to use a significantly smaller limit in the stopping criterion of the inner iteration than in that of the outer iteration. This solution works well in all our test cases. Alternatively for FGMRes, a fixed small number of interface iterations is tested. In the FGMRes case, a flexible control of the interface iteration via the residual norm of the interface system is also possible but is not performed here.

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

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

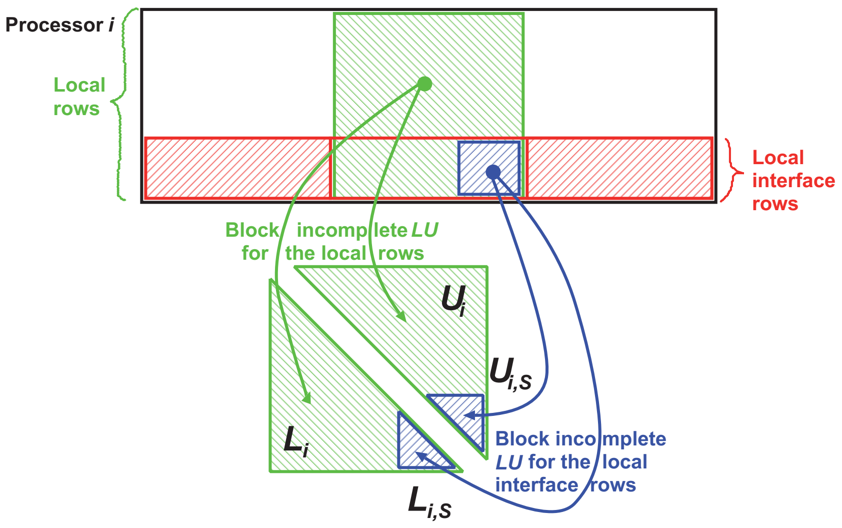
Processor *i*
Local rows
Local interface rows
Block incomplete *LU* for the local rows
U_i
$U_{i,S}$
L_i
$L_{i,S}$
Block incomplete *LU* for the local interface rows

Figure 4: Principle of preconditioning within the DSC algorithm.

with $B_i = L_{i,B}\,U_{i,B}$ the LU decomposition of B_i. By transforming the right hand side of (6) into

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

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

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)):

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

2.4. 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 [5]. Due to special treatments of certain edges in the block-strutured grids within the TRACE code, the sparsity pattern of the coefficient matrix of the emerging sparse systems of linear equations is non-symmetric. Since ParMETIS requires an undirected graph as input the non-symmetric pattern of the matrix has to be symmetrized for the matrix graph construction. Fig. 5 illustrates

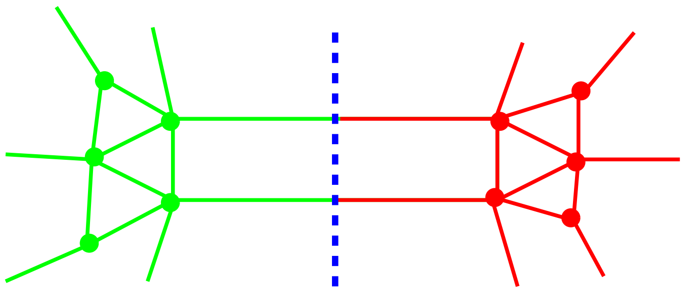

Figure 5: Sample graph partitioning (blue line) into two subdomains.

an advantageous partitioning (blue line) into two subdomains of a sample graph. Only two local interface rows have to be considered within the DSC algorithm per subdomain.

For local, incomplete decompositions, we tested Column Minimum Degree (MD) and Reverse Cuthill-McKee (RCM) reordering for fill-in reduction with MATLAB. Column MD had to be used because of the non-symmetric matrix pattern. In the parallel DSC software developed, we use METIS nested dissection reordering to reduce fill-in in the factors [5]. 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. Detailed experiments on load balance improvements by nested dissection reordering are described in [1, 2].

Tests with other reordering strategies like the matrix permutation methods from [9, 4] are devoted to future investigations.

2.5. Complex and Real Problem Formulation

In the parallel simulation system TRACE, many linear problems are formulated in complex arithmetics. A complex system of linear equations $Ax = b$ can be easily split into its real and complex part:

$$Ax = b \quad \Leftrightarrow \quad (C + iD)(y + iz) = c + id \ . \tag{8}$$

The corresponding real system $Gw = e$ to solve is then

$$\begin{pmatrix} C & -D \\ D & C \end{pmatrix} \begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} c \\ d \end{pmatrix} \quad \Leftrightarrow \quad Gw = e \ . \tag{9}$$

Since most available software packages for solving sparse linear systems are made for real arithmetics it may be useful to transform the complex TRACE problems into real ones. The performance implications of the complex and real problem formulation on many-core systems are examined in the following section.

3. Results

The real and complex versions of the DSC solver were implemented in Fortran and C. MPI (Message Passing Interface) was used for inter-process communication. Beside performance examinations on many-core clusters with this software, tests with MATLAB were performed in order to investigate the implications of different matrix permutations for fill-in reduction for TRACE matrix problems as well as the effect of the real and complex problem formulation on the execution times of the MATLAB solver routines.

3.1. Experiments with MATLAB

MATLAB experiments were performed with a small complex TRACE matrix of order 28,120 with 1,246,200 non-zeros and estimated condition number $6.7 \cdot 10^6$. MATLAB was executed on a quad-core Intel Xeon CPU L5420 workstation.

Fig. 6, left, displays the sparsity pattern of this matrix. The structure of the matrix is quite regular as it is the case for many block-structured CFD problems. Fig. 6, right, shows the pattern of the corresponding small real

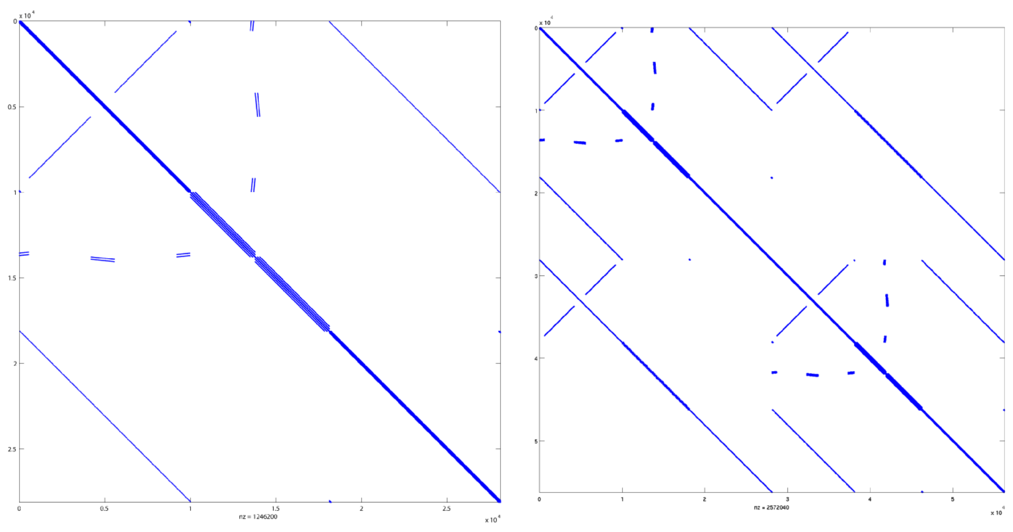

Figure 6: Sparsity pattern of a small complex TRACE matrix of order 28,120 (left) and of the corresponding small real TRACE matrix of order 56,240 (right).

TRACE matrix of order 56,240 generated according to (8) and (9). The structure is more complicated than that of the complex matrix in Fig. 6, left. The number of non-zeros is 2,572,040; the condition number estimation gives $8.4 \cdot 10^6$. Note that the number of non-zeros is only a little higher than twice the number of non-zeros of the associated complex matrix. This means that most imaginary parts of the complex matrix are zero.

Fig. 7 illustrates the sparsity patterns of the complex matrix after Column MD (left) and RCM (right) reodering for fill-reduction. Fig. 8 illustrates the sparsity patterns of the associated real matrix after Column MD (left) and RCM (right) reodering. Note that the small bandwidth for RCM reodering in Figs. 7 and 8 also increases the locality for memory accesses to the matrix data.

Fig. 9 illustrates the effect of reordering on the ILUT fill-in with threshold 10^{-3} (left) and on the ILUT construction time (right) for the small complex TRACE matrix. Both fill-in and ILUT construction time are significantly reduced by Column MD and RCM reordering. While fill-in is a little higher for RCM compared with MD reodering the ILUT construction time is slighty smaller for RCM reodering due to higher locality of memory accesses.

Fig. 10 shows similar results as in Fig. 9. Fill-in and ILUT construction

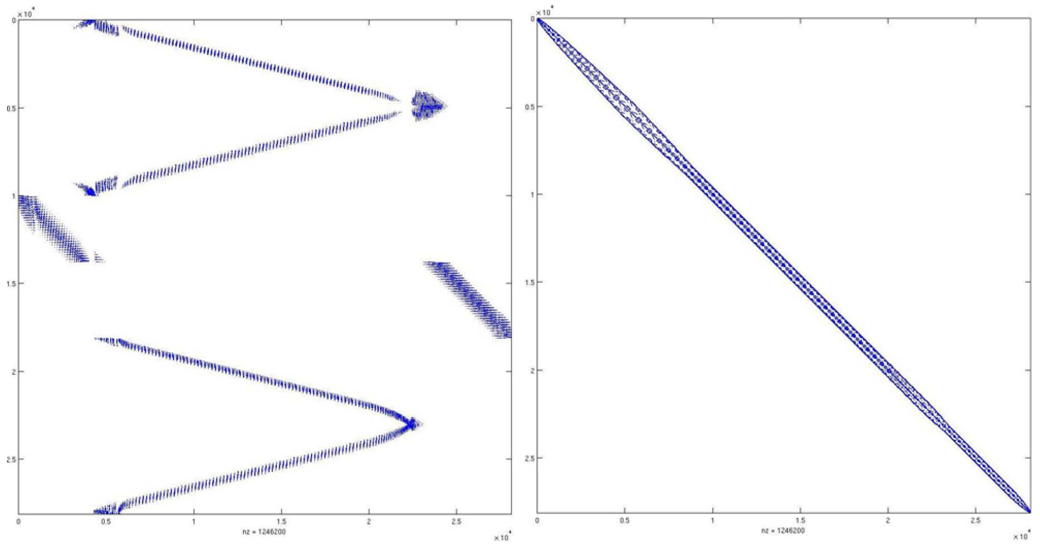

Figure 7: Sparsity pattern of a small complex TRACE matrix after Column MD (left) and RCM (right) reordering.

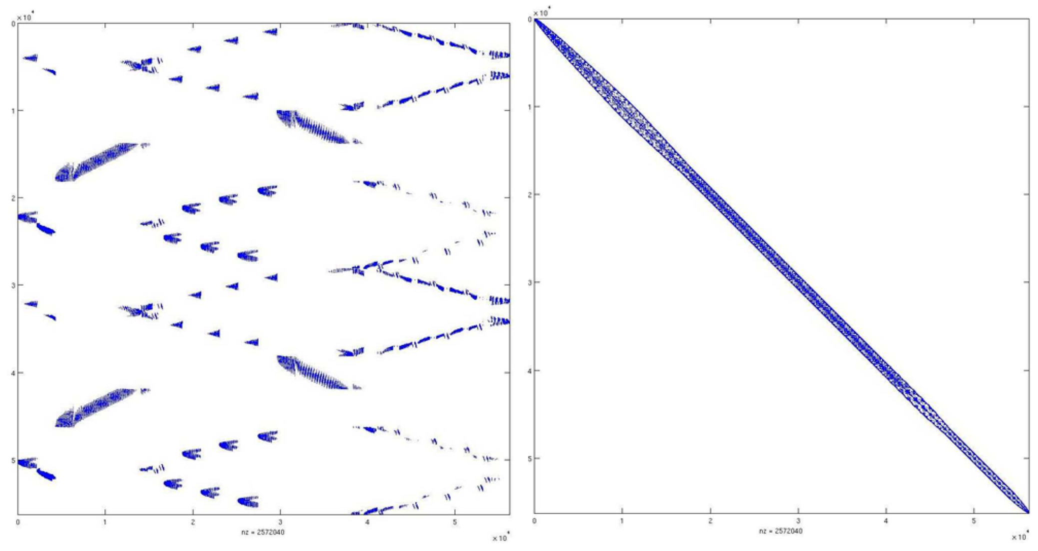

Figure 8: Sparsity pattern of the associated small real TRACE matrix after Column MD (left) and RCM (right) reordering.

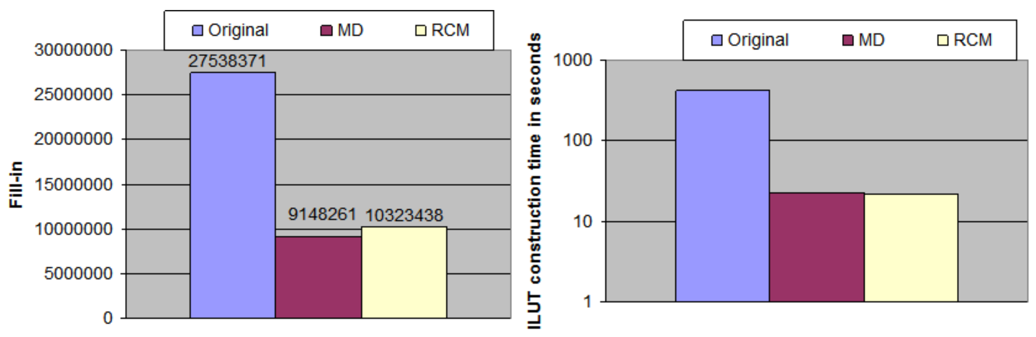

Figure 9: Effect of reordering on the ILUT fill-in with threshold 10^{-3} (left) and on the ILUT construction time (right) for the small complex TRACE matrix.

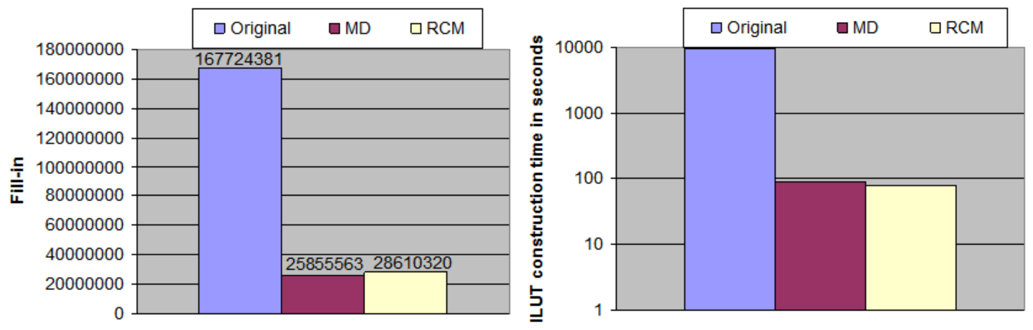

Figure 10: Effect of reordering on the ILUT fill-in with threshold 10^{-3} (left) and on the ILUT construction time (right) for the associated small real TRACE matrix.

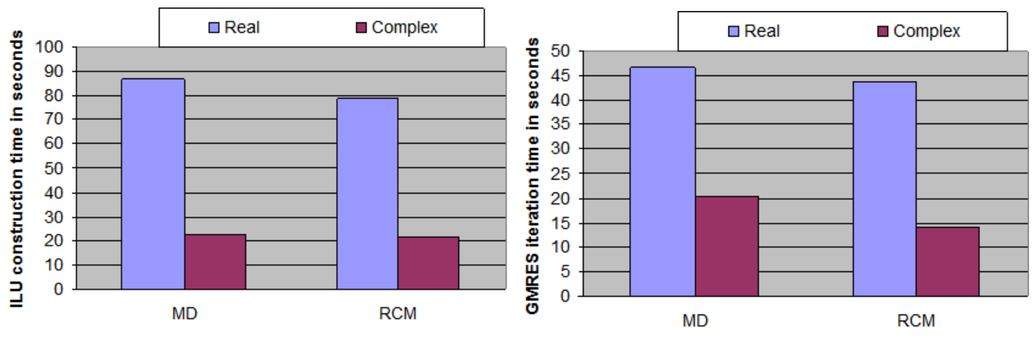

Figure 11: MATLAB execution times for ILUT construction (left) and an ILUT preconditioned GMRES iteration (right) for the real and the complex case with MD or RCM reordering.

time are distinctly reduced by Column MD and RCM reordering. Compared with RCM, MD again results in less fill-in but in a little higher ILUT construction time.

Fig. 11 compares the MATLAB execution times for ILUT construction (left) and an ILUT preconditioned GMRES iteration (right) for the real and the complex case with MD or RCM reordering. The GMRES iteration was stopped when the current residual norm divided by the initial residual was smaller than 10^{-10}. For both ILUT construction and the preconditioned GMRES iteration, the complex formulation results in distinctly shorter execution times, i.e., significantly higher performance on the quad-core Intel Xeon CPU. Reasons are that the complex formulation leads to lower problem order, a more advantageous matrix structure, has higher data locality (storage of real and imaginary part in adjacent memory cells) and a better ratio of computation to memory access due to complex arithmetics in comparison with the real formulation.

3.2. Performance Tests on a Many-Core Cluster

The following experiments were performed on DLR's AeroGrid many-core cluster (45 Dual-processor nodes; Quad-Core Intel Harpertown; 2.83 GHz; 16 GB main memory per node; InfiniBand interconnection network between the nodes).

Fig. 12 displays execution times of the real and complex DSC solver software developed on 4 to 32 processor cores of the AeroGrid cluster for the real and the complex small TRACE matrix problem (cf. Figs. 6, left, or 6, right). METIS Nested Dissection was used for reordering of the local diagonal blocks (cf. section 2.4). ILUT with threshold 10^{-3} was applied for factoring the local blocks. The FGMRes iteration was stopped when the current residual norm divided by the initial residual was smaller than 10^{-5}. As for the MATLAB experiments (cf. Fig. 11), the complex DSC solver version distinctly outperforms the real version both for preconditioner preparation and for the total DSC solver. This is again caused by the lower problem order and a more advantageous matrix structure in the complex case opposite to the real case. In addition, the complex formulation results in higher data locality (storage of real and imaginary part in adjacent memory cells) and a better ratio of computation to memory access due to complex arithmetics in comparison with the real formulation.

Fig. 12 also shows an advantageous strong scaling behavior of the DSC method on the many-core cluster, even for this small matrix problem.

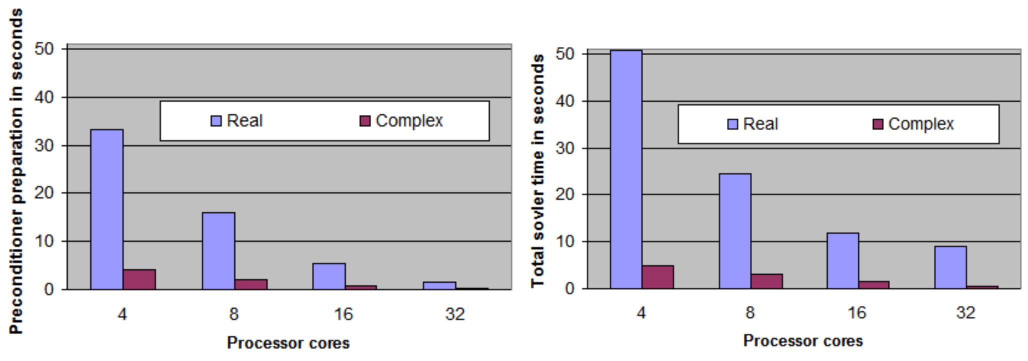

Figure 12: Execution times for DSC preconditioner preparation (left) and the total DSC solver (right) for the real and the complex small TRACE matrix problem on a many-core cluster.

Fig. 13, left, illustrates the strong scaling behavior of the total complex DSC method on 4 to 192 processor cores of the AeroGrid cluster for a medium size complex TRACE matrix problem of order 378,400 and with 45,456,500 non-zeros. The FGMRes iteration was stopped when the current residual norm divided by the initial residual was smaller than 10^{-5}.

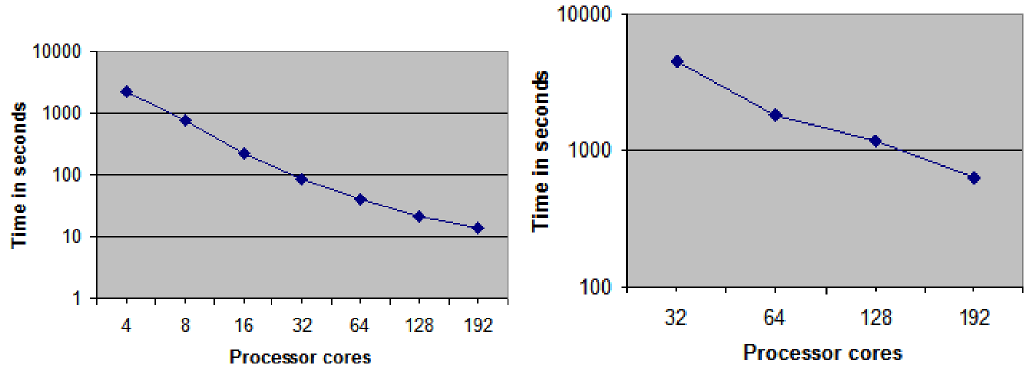

Figure 13: Execution times of the total complex DSC solver for a complex medium size TRACE matrix problem of order 378,400 (left) and for a complex large TRACE matrix problem of order 4,497,520 (right) on a many-core cluster.

Fig. 13, right, shows execution times of the total complex DSC method on 32 to 192 processor cores of the AeroGrid cluster for a large complex TRACE matrix problem of order 4,497,520 and with 552,324,700 non-zeros. In order to achieve the required accuracy in each component of the solution vector the FGMRes iteration was here stopped when the current residual norm divided by the initial residual was smaller than 10^{-10}.

Figs. 12 and 13 demonstrate that the complex DSC algorithm scales very well on many-core clusters for differnt CFD matrix problems of varying size.

4. Conclusions

We discussed numerical, performance and scalability results of DSC methods for typical TRACE CFD problems on many-core architectures together with an analysis of the pros and cons of the complex problem formulation.

Matrix permutations were demonstrated to be crucial for incomplete factorizations within the DSC preconditioner as well as for iterative solver performance. The developed DSC preconditioned iterative solver software for the complex problem formulation distinctly outperformed the solvers for the real formulation on many-core clusters. Reasons are that the complex formulation results in lower problem order, more advantageous matrix structure, has higher data locality and a better ratio of computation to memory access.

Furthermore, the complex DSC algorithm implemented showed an advantageous strong scaling behavior on many-core architectures for large equation systems from Navier-Stokes CFD problems. This was achieved by communication-reducing partitioning of the symmetrized matrix graphs and by a load balance improvement for local incomplete factorizations through reordering of local matrix blocks.

Acknowledgements

This work has been supported by the German Federal Ministry of Education and Research (BMBF) under grant 01IH08012 A (HICFD – Highly Efficient Implementation of CFD Codes for HPC Many-Core Architectures; `http://www.dlr.de/sc/en/desktopdefault.aspx/tabid-5762/9332_read-17927/`).

References

[1] A. Basermann, F. Cortial-Goutaudier, U. Jaekel, K. Hachiya, Parallel solution techniques for sparse linear systems in circuit simulation, in: Proceedings of the 4th International Workshop on Scientific Computing in Electrical Engineering, Series: Mathematics in Industry, Springer, 2002.

[2] A. Basermann, U. Jaekel, M. Nordhausen, K. Hachiya, Parallel iterative solvers for sparse linear systems in circuit simulation, Future Generation Computer Systems 21 (2005) 1275–1284.

[3] L. Giraud, A. Haidar, Parallel algebraic hybrid solvers for large 3D convection-diffusion problems, Numerical Algorithms, 51 (2009) 151–177.

[4] M. Hagemann, O. Schenk, Weighted Matchings for Preconditioning Symmetric Indefinite Linear Systems, SIAM J. Sci. Comput. 28 (2006) 403–420.

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

[6] Y. Saad, A flexible inner-outer preconditioned GMRES algorithm, SIAM J. Sci. Statist. Comput. 14 (1993) 461–469.

[7] Y. Saad, Iterative Methods for Sparse Linear Systems, second ed., SIAM, Philadelphia, 2003.

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

[9] O. Schenk, S. Röllin, A. Gupta, The Effects of Nonsymmetric Matrix Permutations and Scalings in Semiconductor Device and Circuit Simulation, IEEE Trans. Computer-Aided Des. Integrated Circuits Syst. 23 (2004) 400–411.

[10] D.B. Szyld, J.A. Vogel, FQMR: A flexible quasi-minimal residual method with inexact preconditioning, SIAM J. Sci. Comput. 23 (2001) 363–380.

[11] 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) 631–644.

[12] F.W. Wubs, J. Thies, A robust two-level incomplete factorization for (Navier-)Stokes saddle point matrices, Science and Technology Facilities Council, Technical Report RAL-TR-2010-022,

ISSN 1358-6254, Rutherford Appleton Laboratories, July 2010. ftp://ftp.numerical.rl.ac.uk/pub/reports/wRAL2010022.pdf.

Vitae

Achim Basermann obtained his Ph.D. in Electrical Engineering from RWTH Aachen in 1995 followed by a post-doctoral position in Computer Science at Research Centre Jülich GmbH, Central Institute for Applied Mathematics. In 1997 he joined C&C Research Laboratories, NEC Europe Ltd., in Sankt Augustin, Germany and led a team of HPC application experts. Since 2009 he is with the Simulation and Software Technology institute of the German Aerospace Center (DLR) in Cologne, Germany, as team leader of the HPC group. Current research is focussed on parallel linear algebra algorithms, partitioning methods and optimization tools in the area of computational fluid dynamics.

Hans-Peter Kersken received his diploma in physics in 1989 and his Ph.D. in 1992 from the University of Bonn, Germany. From 1992 to 1997 he was with the Scientific Computing Group at the Computing Center of the Alfred-Wegener Institute for Polar and Marine Research at Bremerhaven, Germany and from 1997 to 1998 with the Computing Center of the University of Stuttgart, Germany. Since 1998 he is with the Simulation and Software Technology institute of the German Aerospace Center (DLR) in Cologne, Germany. His research interests are in the fields of numerical mathematics, parallel computing, and computational fluid dynamics.