Using Random Butterfly Transformations in Parallel Schur Complement-Based Preconditioning

Marc Baboulin
Université Paris-Sud
91405 Orsay, France
Email: baboulin@lri.fr

Aygul Jamal
Université Paris-Sud
91405 Orsay, France
Email: jamal@lri.fr

Masha Sosonkina
Old Dominion University
Norfolk, VA, 23529, United States
Email: msosonki@odu.edu

***Abstract*—We propose to use a randomization technique based on Random Butterfly Transformations (RBT) in the Algebraic Recursive Multilevel Solver (ARMS) to improve the preconditioning phase in the iterative solution of sparse linear systems. We integrated the RBT technique into the parallel version of ARMS (pARMS). The preliminary experimental results on some matrices from the Davis' collection show an improvement of the convergence and accuracy of the results when compared with existing implementations of the pARMS preconditioner.**

I. Introduction

With the evolution of recent computer architectures, the growing gap between communication and computation efficiency makes communication very expensive (at a cost of one communication we can generally perform thousands of arithmetical operations). This requires the rethinking of most of numerical libraries in order to take advantage of current parallel architectures which are commonly based on multicore processors [1], possibly with accelerators [2], such as Graphics Processing Units (GPU) or Intel Xeon Phi.

In this work we are concerned with the solution of linear systems $Ax = b$ where A is an $n \times n$ real matrix (dense or sparse), b is a real n-vector and x is the n-vector of unknowns. This operation is at the heart of many applications in high-performance computing (HPC) and is usually solved using either direct or iterative methods.

Direct methods [3] usually solve a linear system of equations $Ax = b$ using factorization techniques depending on the properties of the original matrix A. For a general system, we compute an LU factorization of A that decomposes the input matrix A into the product $L \times U$, where L is a lower triangular matrix and U is an upper triangular matrix. When A is positive definite, then we decompose the matrix A into the product $A = L \times L^T$ (Cholesky decomposition, which requires half the number of flops of the LU factorization). In both cases (LU or Cholesky), the solution is then obtained by solving successively 2 triangular systems.

Another possibility to solve $Ax = b$ is to use an iterative method to compute an approximate solution. These methods involve passing from one iteration to the next one by modifying one or a few components of an approximate vector solution at a time. Classical examples of iterative methods are the Jacobi, Gauss-Seidel, Successive Over-Relaxation (SOR), or Krylov subspace methods [4].

The Algebraic Recursive Multilevel Solver (ARMS) is one of the solvers which applies the iterative Krylov subspace methods in sparse linear systems, it relies on multilevel partial elimination. The preconditioning separates the entries into two parts, the first part called fine set which is composed of block independent set, and the second part called coarse set which contains the rest of the entries. The coarse set can be used to built the Schur complement, which allows us to perform a block LU factorization. The inter-level LU factorization can be built from the upper level LU factorization and the fine set, up to the first level.

Parallel ARMS (pARMS) is a distributed-memory implementation of ARMS, which relies on distributed group independent sets. It provides a set of standard preconditioners such as Additive Schwartz, Schur complement and Block Jacobi, which allow to run performance tests.

When solving square linear systems $Ax = b$ using Gaussian elimination (e.g., in LU factorization), we commonly use partial pivoting to avoid having zero or too-small numbers on the diagonal. This technique is implemented in current linear algebra libraries and ensures stability [5]. However, partial pivoting requires communication (search for pivots, swapping of rows). For example, on a hybrid CPU/GPU system, the LU algorithm in the MAGMA library [2] spends more than 20% of the factorization time in pivoting even for a large random matrix of size 10, 000 $\times$ 10,000 [6].

As an alternative to pivoting, an approach based on randomization called Random Butterfly Transformation (RBT) [7] was recently revisited. Following the RBT method, A is transformed into a matrix that would be sufficiently random to avoid pivoting (with a probability close to 1). RBT is a random transformation of A which can avoid pivoting and then can reduce the amount of communication. We can obtain

This work used resources of the National Energy Research Scientific Computing Center (NERSC), supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. Sosonkina was supported in part by the Air Force Office of Scientific Research under the AFOSR award FA9550-12-1-0476, by the National Science Foundation grants NSF/OCI—0941434, 0904782, 1047772, and by the U.S. Department of Energy, Office of Advanced Scientific Computing Research, through the Ames Laboratory, operated by Iowa State University under contract No. DE-AC02-07CH11358.

satisfying accuracy with an additional computational cost, which is negligible compared to the cost of factorization. This method has been successfully applied to dense linear systems for either general [6] or symmetric indefinite [8] systems, in the context of direct methods based on matrix factorization.

In this work we want to study the possibility of using RBT in iterative linear system solvers based on Krylov Subspace methods, which are widely used in physical and industrial applications.

This paper is organized as follows. Section II presents the preconditioned Krylov subspace method (PKSM) and the parallel Algebraic Recursive Multilevel Solver (pARMS) for solving sparse linear systems. Section III explains how randomization through Random Butterfly Transformation can be integrated into pARMS. For the obtained solver, Section IV proposes performance and accuracy results. Conclusions are presented in Section V.

II. Preconditioned Krylov Methods and the pARMS Solver

A. Preconditioned Krylov Methods

A preconditioned Krylov subspace method (PKSM) is used to solve the linear system $Ax = b$, where A is square non-symmetric matrix, in general. If M is a preconditioning matrix, then the right-preconditioned system is may be expressed as:

$$AM^{-1}y = b, \text{ where } y = Mx\ , \tag{1}$$

which is solved instead of the original system $Ax = b$. To solve this system by using iterative methods, first, we compute the residual $r_0 = b - Ax_0$ [9] after initializing x_0, then we may use a right-preconditioned Krylov subspace method to find an approximate solution from the affine subspace [10]:

$$x_m = x_0 + \text{span}\{r_0,\ AM^{-1}r_0, \ldots, (AM^{-1})^{m-1}r_0\}\ , \tag{2}$$

which satisfies certain conditions. For instance, the GMRES algorithm [4] requires that the residual $r_m = b - Ax_m$ has a minimal 2-norm. The flexible GMRES is abbreviated as FGMRES [4]. Its implementation differs from that of GMRES mainly in storing the preconditioned vectors $z_j = M_j^{-1}v_j$ because the relation $AZ_m = V_{m+1}\bar{H}_m$ is used instead of a simpler one $(AM^{-1})V_m = V_{m+1}\bar{H}_m$ from GMRES.

One way to obtain the preconditioning matrix M is to use an incomplete LU (ILU) factorization. ILU is constructed by performing an approximate Gaussian Elimination (GE) [11] on a sparse matrix A and dropping certain nonzero entries of the factorization according to different dropping strategies. A dropping strategy that relies on levels of the matrix fill-in results in a factorization called $ILU(K)$.

The preconditioner $ILU(0)$ is obtained by performing the LU factorization of A and dropping all fill-in elements generated during the process. Conversely, if the nonzeros are dropped according to their numerical value magnitudes, then the resulting factorization is called ILU with the threshold or—if combined with the dropping strategy based on the number of remaining nonzero—with dual threshold ($ILUT$) and is performed as follows. In the algorithm $ILUT(k, \tau)$, there are two important rules. (1) If an element is less than relative tolerance τ_i (τ $\times$ the norm of the ith row), it is dropped. (2) Keep only the k largest elements in the L and U parts of the row along with the diagonal element.

In this work, we use a preconditioner called Algebraic Recursive Multilevel Solver (ARMS) [12], which is based on a block incomplete LU factorization with different dropping strategies. This block factorization consists of an approximate GE process separating the unknowns into two sets; and an idea of independent or "group independent" set is exploited to define the separation. Hence, the original linear system $Ax = b$ is permuted into the form:

$$\begin{pmatrix} B & F \\ E & C \end{pmatrix} \times \begin{pmatrix} u \\ y \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix}\ , \tag{3}$$

where the submatrix B corresponds to group-independent set reorderings, thereby generating a block-diagonal matrix B [13]. Thus, it is convenient to eliminate the u variable to obtain a system with only y variable. The coefficient matrix for the resulting "reduced system" is the Schur complement $S = C - EB^{-1}F$ [14]. A recursion can now be exploited, such that dropping is applied to S to limit the fill-ins followed by the reordering of the resulting reduced system into the form (3) by using the group-independent set reordering again. This process is repeated for several levels of recursion until the Schur-complement system is small enough or until a maximum number of recursion levels is reached. Then, the last Schur complement may be solved by a direct or an iterative solver. Note that the sparsification of the Schur complement may be undertaken at each level of recursion, to keep down the preconditioning costs.

In this paper, we are interested in parallelizing the iterative methods rather than direct methods. There are two reasons can explicate our choices. First, the direct methods are scale poorly with problem size, when the problem size augment rapidly, the iterative methods are the only choice, which can compute the approximate solution of linear system $Ax = b$. Second, it is hard to parallelize the direct methods which need more space and time to compute, while iterative methods involve passing from one iteration to the next one by modifying one or a few components of an approximate vector solution at a time and it is easy to parallelize.

B. Parallel Implementation of ARMS

Figure 1 outlines distributed linear system solution using pARMS [15]. First, the initial matrix A is distributed among the processors, using a graph partitioning method. In Figure 1, each column of blocks depicts one processor, hence there are five processors shown. Second, each processor solves its part of the system in parallel to construct its portion of the global preconditioner. Then FGMRES solves the preconditioned system with a given accuracy.

When considering the parallel implementation, it is important to specify how the matrix is distributed and handled in parallel. In particular, our pARMS implementation partitions

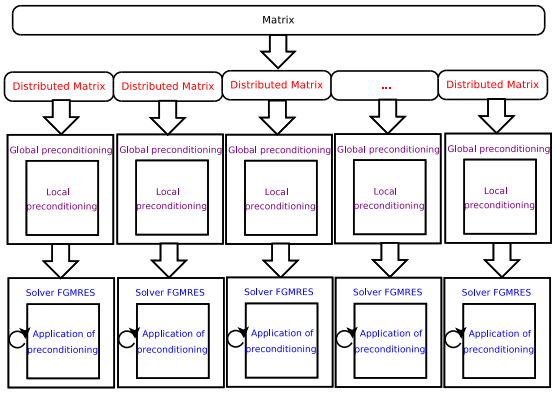


Fig. 1. Sketch of the distributed linear system solution using pARMS on five processors.

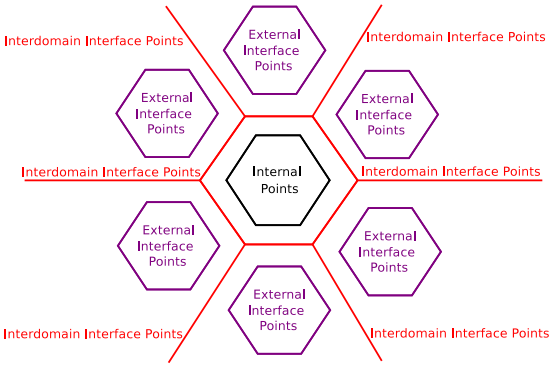


Fig. 2. Per-subdomain view of equation variables-points.

the whole matrix on a single processor using a distributed site expansion (DSE) technique, which is rather simple yet effective in constructing well-balanced subdomains with small interfaces [16]. Although partitioning the entire matrix by a single processor lacks scalability, we note here that this is done by the driver routine, which may be adapted to an application matrix size and format at hand. Given a distributed matrix, Figure 2 shows the per-subdomain division of variables into internal, interdomain interface, and external (residing on the neighboring processors) sets.

We outline now three global preconditioners types available in pARMS: Block-Jacobi preconditioner (BJ), Schur complement preconditioner (SCHUR), and Schur-complement based Restrictive Additive Schwartz preconditioner (SchurRAS). BJ is the simplest global preconditioner because it does not take into account the interface information between neighboring subdomains [17]. SCHUR relates equations associated with the local and interdomain interface points [18]. SchurRAS is constructed from the local ARMS preconditioners in each subdomain using an overlap similar to a standard RAS preconditioner [19] and acting on the Schur complement system as shown in [20]. Specifically, for each of the three preconditioner types, the following algorithms may be implemented in each subdomain.

BJ preconditioner:
1. Update local residual: $r_i = (b - Ax)_i$,
2. Solve: $A_i \delta_i = r_i$,
3. Update local solution: $x_i = x_i + \delta_i$.

SCHUR preconditioner:
1. From (3) compute: $g_i' = g_i - E_i B_i^{-1} f_i$,
2. Solve: $S_i y_i + \sum_{j \in N_i} E_{ij} y_j = g_i'$, where $S_i = C_i - E_i B_i^{-1} F$ and N_i is a set of neighboring subdomains,
3. Back substitute: u_i with $B_i u_i = f_i - E_i y_i$.

SchurRAS preconditioner:
1. Compute local right-hand side g_i'.
2. Solve local Schur-complement system extended with rows for all external variables $y_{i,ext}$.
3. Back substitute: u_i with $B_i u_i = f_i - E_i y_i$.

Note that the local solves in step 2 of BJ, SCHUR, and SchurRAS may be accomplished using incomplete LU or ARMS procedures, mentioned in section II-A. In this work, we apply ARMS enhanced with Recursive Butterfly Transformations (RBT) in step 2 of SCHUR to alleviate the extra work associated with pivoting that may be required in the Schur-complement matrix S_i due to its poor conditioning.

III. Overview of Random Butterfly Transformations and Implementation

In this section we recall the main definitions related to RBT and how it can be applied to pARMS.

A. Randomization

Random Butterfly Transformation (RBT) is a randomization technique initially described by Parker [7] and recently revisited in [6] for general dense systems and [8] for symmetric indefinite systems. It has also been applied recently to a sparse direct solver in a preliminary paper [21]. The procedure to solve $Ax = b$, where A is a general matrix, using a random transformation and the LU factorization is summarized in Algorithm 1. The random matrices U and V are chosen among a particular class of matrices called *recursive butterfly matrices*. A *butterfly matrix* is a *random* $n \times n$ matrix of the form

$$B^{<n>} = \frac{1}{\sqrt{2}} \begin{bmatrix} R_0 & R_1 \\ R_0 & -R_1 \end{bmatrix},$$

where R_0 and R_1 are random diagonal $\frac{n}{2} \times \frac{n}{2}$ matrices. A *recursive butterfly matrix* of size n and depth d is defined recursively as

$$W^{<n,d>} = \begin{bmatrix} B_1^{<n/2^{d-1}>} & & \\ & \ddots & \\ & & B_{2^{d-1}}^{<n/2^{d-1}>} \end{bmatrix} \cdot W^{<n,d-1>}$$

with $W^{<n,1>} = B^{<n>}$ where the $B_i^{<n/2^{d-1}>}$ are butterflies of size $n/2^{d-1}$, and $B^{<n>}$ is a butterfly of size n.

In the original work by Parker, $d = \log_2 n$; it is proved that, given two recursive butterfly matrices U and V, the matrix $U^T AV$ can be factored into LU without pivoting with probability 1 in exact arithmetic, or with probability $1 - O(2^{-t})$ using t-bit floating point numbers. RBT was extensively studied for dense matrices and it was shown in [6] that in practice, $d = 1$ or 2 is enough to obtain a satisfactory accuracy (in most cases a few steps of iterative refinement can recover the digits that have been lost). It has been showed that random butterfly matrices are cheap to store and to apply ($O(nd)$ and $O(dn^2)$ respectively) and they proposed implementations using the dense linear algebra PLASMA and MAGMA. As was demonstrated in the related papers, the preprocessing by RBT can be easily parallelized and provides good scalability.

Algorithm 1 Random Butterfly Transformation Algorithm

Generate recursive butterfly matrices U and V
Perform randomization to update the matrix A and obtain the randomized matrix $A_r = U^T AV$
Factorize the randomized matrix with no pivoting [22]
Compute $U^T b$ and solve $A_r y = U^T b$, then $x = Vy$

B. Integration of RBT into pARMS

We describe in this section how Random Butterfly Transformations can be integrated into pARMS. Our goal is to find the last level of preconditioning and then replace the original $ILUT$ factorization by the RBT pre-processing. Note that RBT usually concerns dense linear systems, while ARMS addresses sparse linear systems. So we have to convert the last Schur complement which is a sparse matrix into a dense format, and after that we can use RBT. Then after randomizing the last Schur complement A with recursive butterfly matrices U and V, the dense matrix is factorized using a Lapack-like [23] routine that performs Gaussian elimination without pivoting, followed by two triangular solves. Note that RBT requires the size of the matrix to be a power of 2, which can be obtained by "augmenting" the matrix A with additional 1's on the diagonal.

The pARMS solver manages the parallel part by using global preconditioning with MPI instructions, while the local part of the code, more precisely the local preconditioning phase does not use MPI instructions. Then the parallelism is entirely managed by pARMS. The local preconditioning can be based on $ilu0$, $iluk$, $ilut$ or $arms$. The essential part resides in the last Schur complement, where we implemented RBT and the preconditioned matrix is then used in FGMRES in order to solve the linear system.

IV. Numerical Experiments

This section describes preliminary results obtained by integrating RBT into the pARMS solver. The experiments have been carried out using one node (2 twelve-core AMD MagnyCours Opteron 6172 processors running at 2.10GHz) of the Hopper machine located at NERSC[1]. In these experiments, we used matrices from the Davis' collection [24] to test the performance of different preconditioners. The first matrix (`Sherman5`) is a real non-symmetric matrix of size $3,312$ (nnz = 20,793). `Sherman5` arises from a three dimensional simulation model on a $n_x \times n_y \times n_z$ grid using a seven-point finite-difference approximation with n_c equations and unknowns per grid block, where n_x is 16, n_y is 23, n_z is 3, n_c is 3. The second matrix (`Raefsky3`) is a real non-symmetric matrix of size $21,200$ (nnz =1,488,768), which arises from a fluid structure interaction turbulence problem. The third matrix (`Cant`) is a real symmetric matrix that comes from a 2D/3D FEM problem, of size 62,451 (nnz = 2,034,917). For the three matrices, we study the results obtained when using a global Schur complement-based preconditioner with the following local preconditioners: $ilu0$, $iluk$, $ilut$, $arms$, or $arms\_rbt$. The pARMS parameters are chosen for these matrices following the guidance for the local ARMS preconditioner, as explained in [12], for example. Certain parameters influence considerably the size and density of the last Schur Complement, which, in turn, affects greatly the performance of RBT. Since the RBT for dense matrices is used in this work, it is desirable that the last Schur Complement remains dense while being

[1] http://www.nersc.gov

relatively small. Hence, parameter values for the number of ARMS levels and the ARMS independent block size were chosen such that a small Schur Complement is obtained. In particular, the former parameter was small (equals two) while the latter was large (allowing to form the blocks up to the entire local matrix size). At the same time, the drop tolerance for the last Schur Complement was kept quite low (0.001) as well as all the other intermediate-level drop tolerances, so that there is close to none sparsification of the Schur Complement.

The experiments are performed using 4 to 12 cores, and we use one MPI process per core and no multi-threading. In Figure 3, we compare the number of iterations required for convergence. We observe that, for matrices `Sherman5` (fig. 3(a)), `Raefsky3` (fig. 3(b)), $arms\_rbt$ performs better than the other local preconditioners. For `Cant`, $arms\_rbt$ converges in fewer iterations than the other preconditioners do so when using up to eight cores. This observation suggests that $arms\_rbt$ may be a more versatile preconditioner to use for obtaining superior convergence. Note that, for different numbers of subdomains (one per MPI process) in a given matrix, the obtained parallel preconditioning varies leading to the differences in the number of iterations to converge. Note that for these preliminary tests, $arms\_rbt$ requires more time to solve the system since, in our preliminary implementation, a dense-matrix solver was used to solve the last Schur complement system in pARMS. In the future, we plan to develop a sparse RBT solver based on a sparse direct solver, such as SuperLU [25]. Figure 4 represents the residual obtained with the five local preconditioners. We observe that these preconditioners provide us with a similar accuracy, $arms\_rbt$ being more accurate for the matrix `Raefsky3`.

V. Conclusion and Future Work

We have investigated the feasibility of using RBT randomization in the pARMS solver and how RBT may enhance the iterative convergence. Most of our experiments showed an improvement in the number of iterations and accuracy of results. However, our integration of RBT in pARMS necessitates an implementation that may adjust the sparsity of the last Schur complement matrix based on the available memory and on the performance characteristics of its (direct) solver at hand. As a future work, we will integrate a sparse RBT direct solver based on SuperLU, which will also enable us to solve large-scale sparse linear systems.

References

[1] A. Buttari, J. Langou, J. Kurzak, J. Dongarra, A class of parallel tiled linear algebra algorithms for multicore architectures, Parallel Comput. 35 (1) (2009) 38–53.

[2] S. Tomov, J. Dongarra, M. Baboulin, Towards dense linear algebra for hybrid GPU accelerated manycore systems, Parallel Computing 36 (5&6) (2010) 232–240.

[3] T. A. Davis, Direct Methods for Sparse Linear Systems (Fundamentals of Algorithms 2), Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2006.

[4] Y. Saad, Iterative Methods for Sparse Linear Systems, 2nd Edition, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003.

[5] N. J. Higham, Accuracy and Stability of Numerical Algorithms, 2nd Edition, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002.

[6] M. Baboulin, J. Dongarra, J. Herrmann, S. Tomov, Accelerating linear system solutions using randomization techniques, ACM Trans. Math. Softw. 39 (2) (2013) 8:1–8:13.

[7] D. S. Parker, Random butterfly transformations with applications in computational linear algebra, Tech. Rep. CSD-950023, University of California Los Angeles, CA USA (1995).

[8] M. Baboulin, D. Becker, G. Bosilca, A. Danalis, J. Dongarra, An efficient distributed randomized algorithm for solving large dense symmetric indefinite linear systems, Parallel Computing 40 (7) (2014) 213–223.

[9] M. Arioli, J. Demmel, I. Duff, Solving sparse linear systems with sparse backward error, SIAM Journal on Matrix Analysis and Applications 10 (2) (1989) 165–190.

[10] B. N. Bond, L. Daniel, Guaranteed stable projection-based model reduction for indefinite and unstable linear systems, in: Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design, ICCAD '08, IEEE Press, Piscataway, NJ, USA, 2008, pp. 728–735.

[11] S. Donfack, J. Dongarra, M. Faverge, M. Gates, J. Kurzak, P. Luszczek, I. Yamazaki, A Survey of Recent Developments in Parallel Implementations of Gaussian Elimination, Concurrency and Computation: Practice and Experience (2014) 18.

[12] Y. Saad, B. Suchomel, ARMS: an algebraic recursive multilevel solver for general sparse linear systems, Numerical Linear Algebra with Applications 9 (5) (2002) 359–378.

[13] Y. Bai, W. N. Gansterer, R. C. Ward, Block tridiagonalization of "effectively" sparse symmetric matrices, ACM Trans. Math. Softw. 30 (3) (2004) 326–352.

[14] Z.-H. Cao, Constraint schur complement preconditioners for nonsymmetric saddle point problems, Appl. Numer. Math. 59 (1) (2009) 151–169.

[15] Z. Li, Y. Saad, M. Sosonkina, pARMS: a parallel version of the algebraic recursive multilevel solver, Numerical Linear Algebra with Applications 10 (5-6) (2003) 485–509.

[16] Y. Saad, M. Sosonkina, Non-standard parallel solution strategies for distributed sparse linear systems, in: P. Z. *et al.* (Ed.), Parallel Computation: 4th International ACPC Conference, Vol. 1557 of Lecture Notes in Computer Science, Springer-Verlag, 1999, pp. 13–27.

[17] B. F. Smith, P. E. Bjørstad, W. D. Gropp, Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations, Cambridge University Press, New York, NY, USA, 1996.

[18] Y. Saad, M. Sosonkina, Distributed Schur Complement techniques for general sparse linear systems, SIAM J. Scientific Computing 21 (1999) 1337–1356.

[19] X.-C. Cai, M. Sarkis, A restricted additive schwarz preconditioner for general sparse linear systems, SIAM J. Sci. Comput. 21 (2) (1999) 792–797.

[20] Z. Li, Y. Saad, Schurras: A restricted version of the overlapping schur complement preconditioner, SIAM J. Sci. Comput. 27 (5) (2005) 1787–1801.

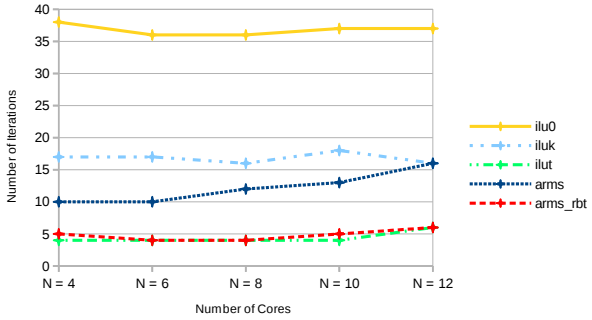
[21] M. Baboulin, X. S. Li, F.-H. Rouet, Using random butterfly transformations to avoid pivoting in sparse direct methods, in: Proceedings of VECPAR 2014, 2014.

[22] M. Baboulin, S. Donfack, J. Dongarra, L. Grigori, A. Rémy, S. Tomov, A class of communication-avoiding algorithms for solving general dense linear systems on cpu/gpu parallel machines, in: International Conference on Computational Science (ICCS 2012), Vol. 9 of Procedia Computer Science, Elsevier, 2012, pp. 17–26.

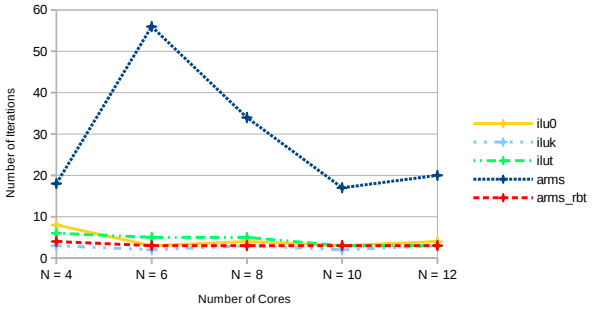
[23] E. Andersen, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchev, D. Sorensen, LAPACK user's guide, 3rd. Ed. 1999 SIAM, Philadelphia (1999).

[24] T. A. Davis, Y. Hu, The University of Florida Sparse Matrix Collection, ACM Trans. Math. Softw. 38 (1) (2011) 1–25.

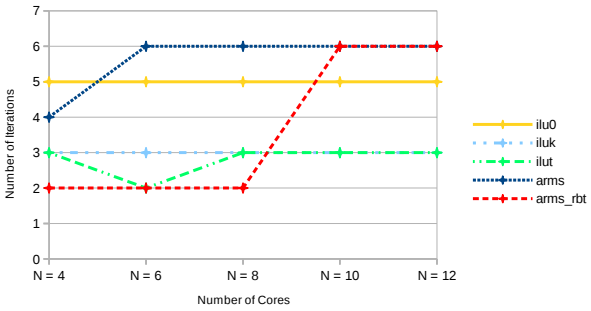
[25] X. S. Li, An overview of SuperLU: Algorithms, implementation, and user interface, ACM Transactions on Mathematical Software 31 (3) (2005) 302–325.



(a) Sherman5

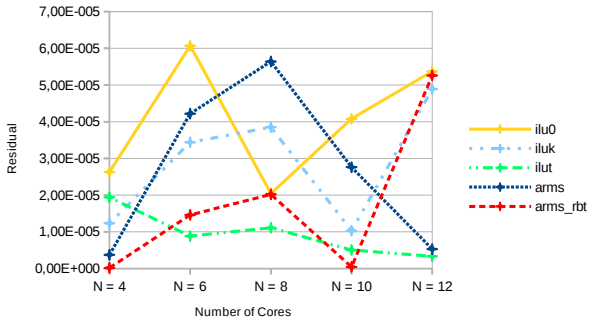


(b) Raefsky3

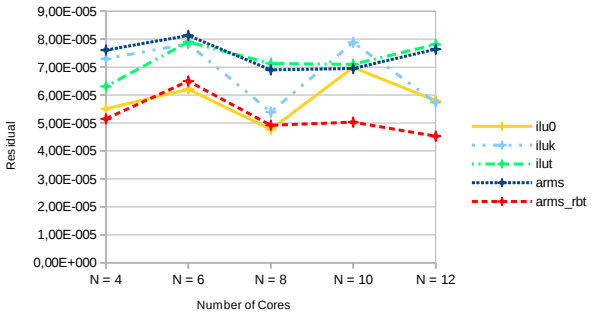


(c) Cant

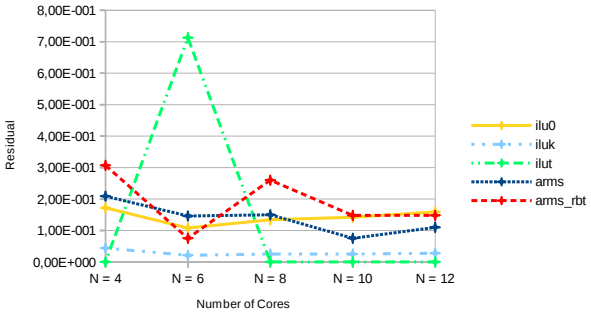
Fig. 3. Iterations required for convergence with five choices of local preconditioner.



(a) Sherman5



(b) Raefsky3



(c) Cant

Fig. 4. Residual for test problems with five choices of local preconditioner.