Stage de Master 2 Recherche en Informatique

Using random butterfly transformations in iterative methods for sparse linear systems

Auteur :
Aygul JAMAL

Maître de stage :
Pr. Marc BABOULIN

Organisme d'accueil :
INRIA

Secrétariat - tél : 01 69 15 75 18 Fax : 01 69 15 42 72
courrier électronique : m2-info-nsi.sciences@u-psud.fr

1 avril – 30 septembre 2014

Contents

Abstract

In this internship, we integrate the randomization technique based on Random Butterfly Transformations (RBT) into the Algebraic Recursive Multilevel Solver (ARMS) to improve the preconditioning phase in the iterative solution of sparse linear systems. After obtaining satisfying experimental results in a sequential implementation using Matlab, we integrated the RBT technique into the parallel version of ARMS (pARMS). By analyzing experimental results we conclude that integrating RBT into pARMS enables us to obtain more accurate results and to reduce the number of iterations required for convergence.

Keywords

Linear systems, Krylov subspace methods, sparse linear algebra, randomization, preconditioning, Random Butterfly Transformation, ARMS, pARMS.

CHAPTER 1

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 [8], possibly with accelerators [2] like 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 [10] 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 $A = L \times U$, where L is a lower triangular matrix and U is an upper triangular matrix. When A is positive definite, then we decompose 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 [22] 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), and Gradient Methods [13].

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 [16]. However, partial pivoting requires communication (search a pivot, swapping of rows). For example, on a hybrid CPU/GPU system, the LU algorithm in the MAGMA library [6] spends over 20% of the factorization time in pivoting even for a large random matrix of size $10,000 \times 10,000$ [26].

As an alternative to pivoting, an approach based on randomization called Random Buttefly Transformation (RBT) [21] was revisited during the recent years. Following the RBT method, A is transformed into a matrix that would be sufficiently random (with a probability close to 1) to avoid the need of pivoting. RBT is a random transformation of A which can avoid pivoting and then can reduce the amount of communication. We can obtain 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 [5] or symmetric indefinite [3] systems, in the context of direct methods based on matrix factorization.

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 seperates 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 build the Schur complement, which allows us to perform a block LU factorization. The interlevel 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 Restrictive Additive Schwarz, Schur complement and Block Jacobi, which allows us to run performance tests.

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.

We now briefly introduce the contents of each chapter in this report. Chapter 2 presents the preconditioned Krylov subspace method (PKSM), which helps us to learn the iterative methods for solving sparse linear systems. Chapter 3 describes the randomization technique RBT, which will be integrated into the Algebraic Recursive Multilevel Solver (ARMS) and its parallel version pARMS. Chapter 4 describes the process of integrating RBT into ARMS, and the experimental results. Chapter 5 describes the process of integrating RBT into pARMS, and the experimental results. Chapter 6 presents the conclusions obtained after this internship.

CHAPTER 2

Preconditioned Krylov Subspace Method

2.1 Preconditioning

In general, a preconditioner is any kind of implicit or explicit modification of an original linear system which makes it "easier" to solve by a given iterative method. In terms of preconditioned Krylov subspace method, preconditioned means we use explicit modification of orginal linear system as form :

$$Ax = b \rightsquigarrow W^T AVy = W^T r_0, \ where \ r_0 = b - Ax_0 \tag{2.1}$$

and let Krylov subspace method to solve the linear system.

2.2 Krylov Subspace Method

Two kinds of iterative methods exist, the first one is Algebraic MultiGrid (AMG) Method, and the other one is Krylov subspace method. AMG methods have been proposed to solve general problems, but their success is limited to solving Partial Differential Equations (PDE) problems [24]. Krylov subspace methods, using incomplete LU preconditioners, are considered general method to solve arbitrary sparse linear system. In this internship, we focus on Krylov subspace method. Krylov subspace mathod is named after the russian mathematician and engineer Alexei Krylov, who published this method in 1931. The fundamental theorem for this paper is the Cayley-Hamilton theorem, which states that the inverse of a matrix can be found in a linear combination of its powers.

Krylov subspace iterative method is based on projection processes, which are orthogonal and oblique onto Krylov subspaces, and which are spanned by the vectors of the form $p(A)v$, where p is polynomial. This technique approximates the solution $A^{-1}b$ by $q_{m-1}(A)b$, in which q_{m-1} is a polynomial of degree $m-1$. Let $V = [v_1, ..., v_m]$ an $n \times m$ matrix, whose column-vectors consist a basis of K. $W = [\omega_1, ...\omega_m]$, an $n \times m$ matrix whose column vectors consist a basis of L. An approximative solution is given by : $x = x_0 + Vy$ [22]. Orthogonal condition produces a new system of equations to vector y, $W^T AVy = W^T r_0$. Assuming $W^T AV$ is nonsingular, the following algorithm can approximate the solution $\tilde{x}$.

2.3 Algorithm

Algorithm 1 Krylov Subspace Algorithm [22]

Compute $r_0 = b - Ax_0$, $\beta := \|r_0\|_2$, and $v_1 := \frac{r_0}{\beta}$
Define the $(m+1) \times m$ Hessenberg matrix $\bar{H_m} = \{h_{ij}\}_{1 \le i \le m+1, 1 \le j \le m}$. $Set \bar{H_m} = 0$.
for j = 1, 2, ..., m **do**
  Compute $\omega_j := Av_j$
  for $i = 1, 2, ..., j$ **do**
    $h_{ij} := (\omega_j, v_i)$
    $\omega_j := \omega_j - h_{ij} v_i$
  end for
  $h_{j+1,j} = \|\omega_j\|_2$
  if $h_{j+1,j} = 0$ **then** set $m := j$ and goto last line,
  else $v_{j+1} = \frac{\omega_j}{h_{j+1,j}}$,
  end if
end for
Compute y_m, the minimizer of $\|\beta e_1 - \bar{H_m} y\|_2$, and $x_m = x_0 + V_m y_m$

CHAPTER 3

Random Butterfly Transformation

3.1 Context

In the linear systems, to solve $Ax = b$, we compute the factorization $A = L \times U$, with L unit-lower triangular and U upper triangular. Then we solve two linear systems of type $Ly = b$, $Ux = y$, which is much faster than to solve directly the system $Ax = b$. However, this factorization has difficulties in the triangular matrix L and U, since it is forbidden to have small numbers or zeros on the diagonal, as we use these numbers in divisions. In this case, we avoid this problem by swapping rows or columns (called pivoting). Unfortunately, the operation of pivoting is expensive in communication, for this reason, we use randomization instead of pivoting. In this way we save communication time, which can be use to perform thousands of computations [15].

3.2 Randomization

A butterfly matix is an $n \times n$ matrix of the form :

$$B = \frac{1}{\sqrt{2}} \begin{pmatrix} R_0 & R_1 \\ R_0 & -R_1 \end{pmatrix} \tag{3.1}$$

where $n \geq 2$ and R_0 and R_1 are random diagonal and nonsingular $n/2 \times n/2$ matrices.

A recursive butterfly $n \times n$ matrix of depth d is a product of the form [5]:

$$W^{<n,d>} = \begin{pmatrix} B_1^{<n/2^{d-1}>} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & B_{2^{d-1}}^{n/2^{d-1}} \end{pmatrix} \times \cdots \times \begin{pmatrix} B_1^{<n/4>} & 0 & 0 & 0 \\ 0 & B_2^{<n/4>} & 0 & 0 \\ 0 & 0 & B_3^{<n/4>} & \\ 0 & 0 & 0 & B_4^{<n/4>} \end{pmatrix} \times \begin{pmatrix} B_1^{<n/2>} & 0 \\ 0 & B_2^{<n/2>} \end{pmatrix} \times B_1^n, \tag{3.2}$$

where $B_i^{<n/2^{k-1}>}$ are butterflies of size $n/2^{k-1} \times n/2^{k-1}$, $k = 2, ..., d$ and $B^{<n>}$ is butterfly of size $n \times n$.

A Random Butterfly Transformation (RBT) of depth d of a $n \times n$ matrix A is the product of $A_r = U^T AV$. It consists of a multiplicative preconditioning $U^T AV$ where the matrices U and V are selected from a particular class of recursive butterfly matrices. Then we use Gaussian Elimination with No Pivoting (GENP) [4] on the matrix $U^T AV$ and instead of solving $Ax = b$, we solve $(U^T AV)y = U^T b$ followed by $x = Vy$.

3.3 Algorithm

Algorithm 2 Random Butterfly Transformation Algorithm [5]

Transform the original matrix to a dense matrix whose size is power of 2
Generate recursive butterfly matrices U and V
Performe randomization to update the matrix A and obtain the matrix $A_r = U^T AV$
Factorize the randomized matrix with GENP
Compute $U^T b$ to solve $A_r y = U^T b$, then solve $x = Vy$
Cut down the vector whose size is a power 2 to its original size

CHAPTER 4

Integration of RBT in the Algebraic Recursive Multilevel Solver (ARMS)

4.1 Sequential implementation of ARMS

Implementation of Preconditioned Krylov Subspace Methods (PKSM)

1. A Preconditioned Krylov Subspace Method (PKSM) is used to solve the linear system $Ax = b$, which consists of a preconditioner and an accelerator [23]. In the following equation system, M is a preconditioning matrix, then the right-preconditioned system is

$$AM^{-1}y = b, \ where \ x = M^{-1}y, \tag{4.1}$$

is solved instead of the original system $Ax = b$. Furthermore, the accelerator is an iterative method, which applies Krylov subspace methods. To compute the residual $r_0 = b - Ax_0$ [1], we should assign an approximate value to the initial x_0, then compute the right-preconditioned Krylov subspace method. This algorithm gives an approximate solution from the affine space :

$$x_m = x_0 + Span\{r_0, AM^{-1}r_0, ..., (AM^{-1})^{m-1}r_0\} \ [7], \tag{4.2}$$

which statisfies certain conditions. For instance, the FGMRES algorithm requires that the residual $r_m = b - Ax_m$ has a minimal 2-norm.

The preconditioning matrix M is obtained from an incomplete LU factorization, which is an approximate Gaussian Elimination (GE) [12] process. When we apply GE to a sparse matrix A, it produces nonzero entries at the place of orginal zero elements. Fortunately, these fill-in entries are usually small and could be dropped, according to the different "dropping strategy". During the elimination process, if this "dropping strategy" relies on levels, then the preconditioner is called a level-of-fill ILU. For instance, ILU(0) is obtained by performing the LU factorization of A and dropping all fill-in elements generated during the process. Conversely, if the fill-ins are dropped according to their numerical values, then the preconditioner is ILU factorization with the threshold (ILUT) of dropping relatively small entries.

Multilevel ILU factorization

In general, we use a method named block incomplete LU factorization (ILU-factorization) to precondition a linear system, which consists of an approximate GE process based on separating the original unkowns into a "coarse" and a "fine" set. The idea of independent or "group independent" sets is exploited to define this seperation. Block independent set orderings permute the original linear system $Ax = b$ into the form :

$$\begin{pmatrix} B & F \\ E & C \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix}, \tag{4.3}$$

in which the submatrix B represents independent set reorderings, which generates a diagonal matrix B [17]. Thus, it is convenient to eliminate the x variable to obtain a system with only y variable. The coefficient matrix for this "reduced system" is the Schur complement $S = C - EB^{-1}F$ [9]. Recursion can now be exploited, such that dropping is supplied to S to limit the fill-ins followed by the reordering of the resulting reduced system into the form 4.3 by independent sets. This process is repeated for several levels until the system is small enough or until a maximum number of levels

is reached. Then the system is solved by a direct sparse solver or an ILUT-GMRES combination. This procedure is called Algebraic Recursive Multilevel Solver (ARMS).

Using ILUs in PKSMs may cause poor scalability. For instace, the ILUT construction phase is largely a sequential process, and parallel variants of ILUT are not effective in terms of convergence. The preconditioned iteration phase itself scales poorly with the problem size. New computer architectures also challenge PKSM implementations to adapt to larger processor counts, nonuniform memory, and novel communication paradigms. Remarkably, sparse direct methods have had a better history of adaptation to new computing environments. The main reason for this is that direct methods have integrated blocking techniques to exploit the dense parts of elimination and memory hierarchies in the computing platforms. In contrast, developers of PKSMs tend to keep fill-ins at bay as much as possible to gain in speed and memory usage, so end up with no or little dense computation phases. This strategy has a disadvantage when exploiting new accelerator architectures, such as general-purpose GPUs. So far, it has been exceedingly difficult to obtain good speeds for sparse iterative methods on GPUs. While some dense computations can extract a good portion of 60%, of the peak rate on an NVIDIA Telsa board for example, only 3% of the peak rate can be achieved for something as mundane as a matrix-vector product (sparse matrix, dense vector).

4.2 Integration of RBT into ARMS

Now we are familiar with Algebraic Recursive Multilevel Solver (ARMS) and Random Butterfly Transformation (RBT), the next step is to integrate our RBT technique into the ARMS solver. Our goal is to find the last level of preconditioning and then replace the original ILUT factorization by our RBT preprocessing. 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. After randomize the last schur complement A with recursive butterfly matrices U and V, then we reconvert the dense matrix back into a sparse format to do the following computations. Moreover, RBT requires the size of the matrix to be a power of 2, which can be always obtained by "augmenting" the matrix A with additional 1's on the diagonal.

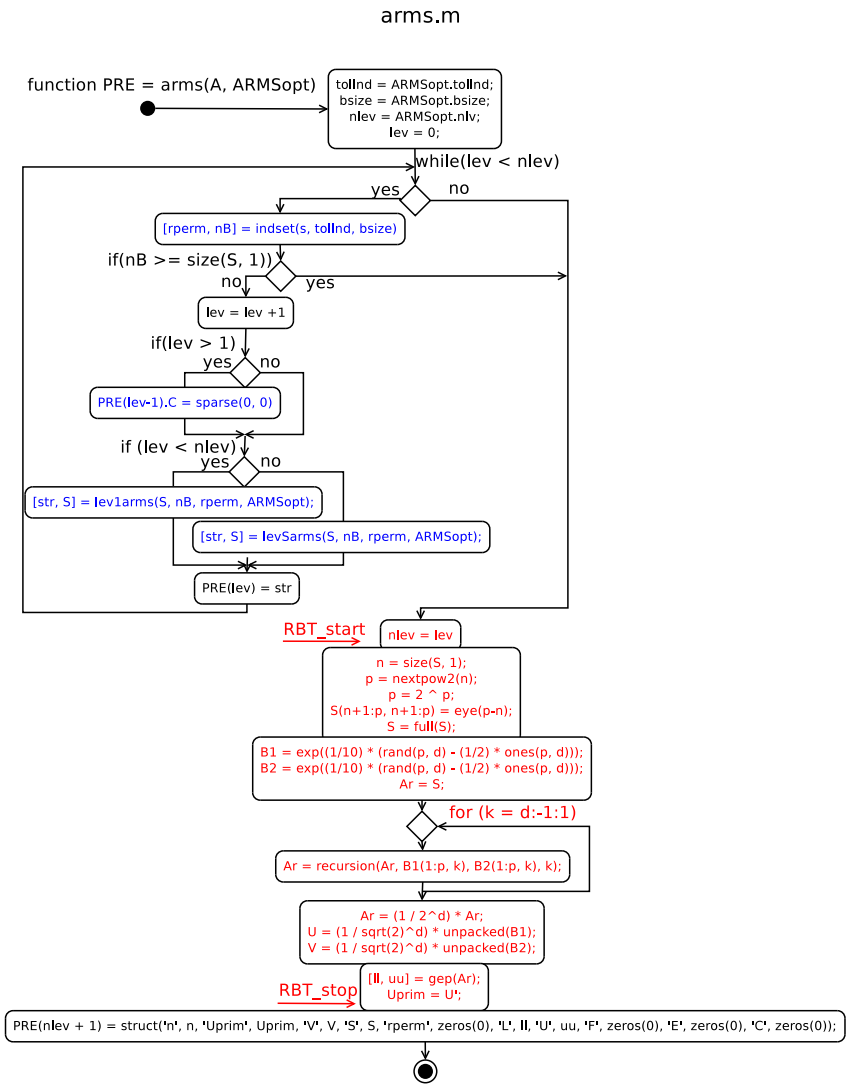

Figure 4.1: Integration of RBT in ARMS

4.3 Comparison of solving time

In this experiment the parameters are as follows : the block size in ARMS reduction is 200; the number of levels in ARMS is 4; the maximum number of outer steps is 1000.

We compare the performance of ARMS with and without applying RBT. The test matrices arise from a 3-D convection/diffusion problem with the convection coefficients of 0.1 in all 3 directions, and with the Dirichlet Boundary conditions. The problem was discretized using a 7-point centered finite-difference scheme on a $n_x \times n_y \times n_z$ grid, excluding boundary points, where $n_x = n_y = 15$ and n_z is taken as 10, 20, or 30 to get the problems of varying total size 2250, 4500, or 6750 grid points, respectively. Table 4.1 contains the timing results for the construction of ARMS preconditioner (rows arms and armsprec) without and with RBT, respectively; and for the application of ARMS,

we observe (see bold-faced numbers in Table 4.1) an improvement in time of about 7.8%, 8.8%, or 2.8% respectively, when using RBT.

Table 4.1: Timing results for the test problems (seconds)

Algo	Size = 2250	Size = 4500	Size = 6750
arms	14.82	22.87	25.08
arms_rbt	**13.75**	**21.02**	**24.39**
armsprec	0.091	1.388	4.462
armsprec_rbt	**0.082**	**1.185**	**3.196**

Now, let us visualize the data in the following graph.

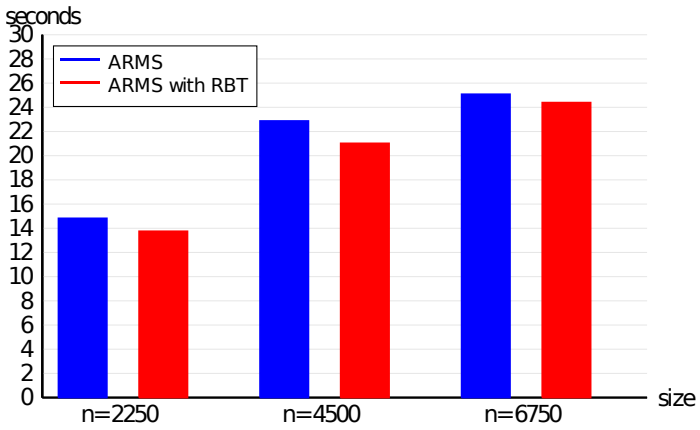

Figure 4.2: Timing results for arms and arms_rbt

4.4 Comparison of iterations required for convergence

In this experiment the parameters are as follows : the block size in ARMS reduction is 200; the number of levels in ARMS is 4; the maximum number of outer steps is 1000.

Table 4.2 shows the number of iterations (column #it) needed to reach an approximate solution with relative tolerance of 10^{-10}, and the corresponding (column fill-factor). The fill-factor is defined as the ratio of total ARMS preconditioner to fill-in to the number of nonzeros in the original matrix. We observe (see bold-faced numbers in Table 4.2) that the number of iterations required when ARMS with RBT is used is smaller than that for the ARMS without RBT. Hence, RBT may accelerate the convergence of ARMS.

Table 4.2: Convergence and fill-factor tests of different preconditioners

	Size = 2250		Size = 4500		Size = 6750	
Type	#it	fill-factor	#it	fill-factor	#it	fill-factor
ILUT	38	**2.53**	194	**2.62**	635	2.65
arms	**26**	2.82	70	2.95	272	3.36
arms_rbt	**26**	3.48	**67**	2.96	**200**	**2.53**

Now, let us visualize the data in the following graph.

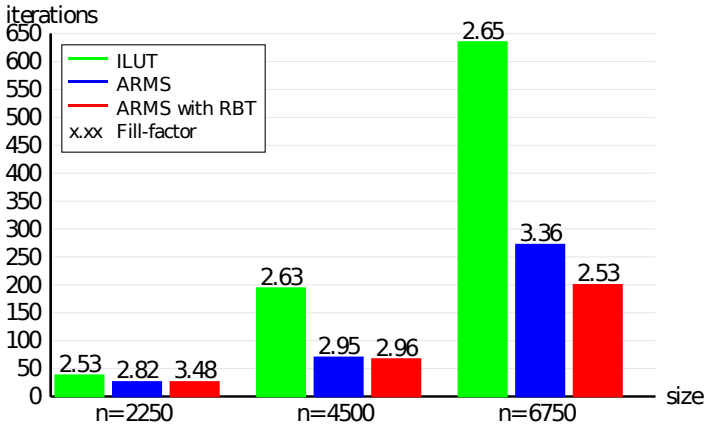

Figure 4.3: Convergence and fill-factor tests of different preconditioners

CHAPTER 5

Integration of RBT in the parallel Algebraic Recursive Multilevel Solver (pARMS)

The Figure 5.1 presents the fundamental process of pARMS [19]. At first, we have an initial matrix A, which is distributed among the processors, using domain decomposition methods. Then each processor approximately solves a part of the system independently, using local preconditioners. Exchange of information for shared data is made by a global preconditioning. This process is repeated until the solution is sufficiently accurate.

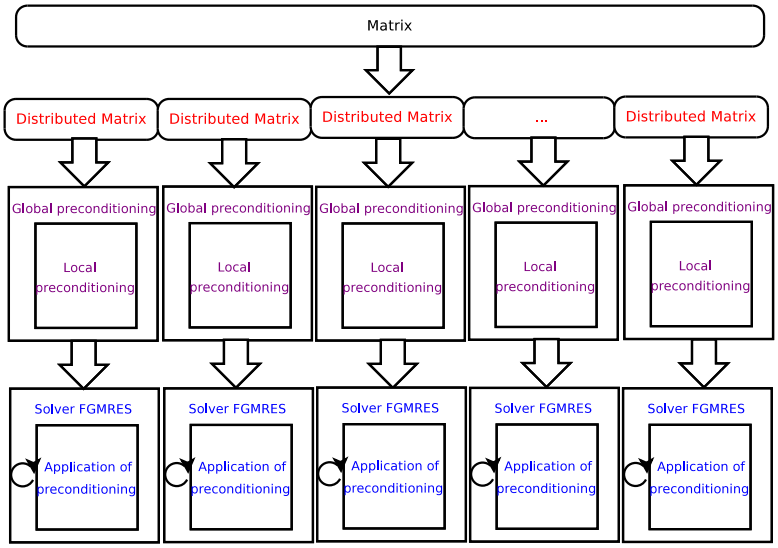

Figure 5.1: The fundamental process of pARMS

5.1 Domain Decomposition method

When considering the parallel implementation, it is important to mention the domain decomposition method [25]. In pARMS, we use a special domain decomposition algorithm that we discuss now. In this program, each processor reads the whole matrix, which is assumed to be in Harwell-Boeing format. Matrix graph is then partitioned using Distributed Site Expansion (DSE), a simple partitioning routine, and scatters the local matrices to each processor. Once these submatrices are received, each processor solves the problem using preconditioned FGMRES preconditioned with : Restrictive Additive Schwarz preconditioner (RAS), Schur complement based preconditioner (SCHUR) and Block-Jacobi preconditioner (BJ).

Independent sets

To move to a parallel implementation of ARMS, we need to generalize the notion of Group-Independent Sets [19] to Distributed Group-Independent Sets (DGIS) [19]. DGIS means unkowns of different groups (with and across processors) are not coupled, which is easily calculated by further

subdividing these sets of interior nodes. The local matrix is represented by three sets : IS - the points in distributed group-independent sets, I1 - the set of local interface points, and I2 - the set of interdomain interface points.

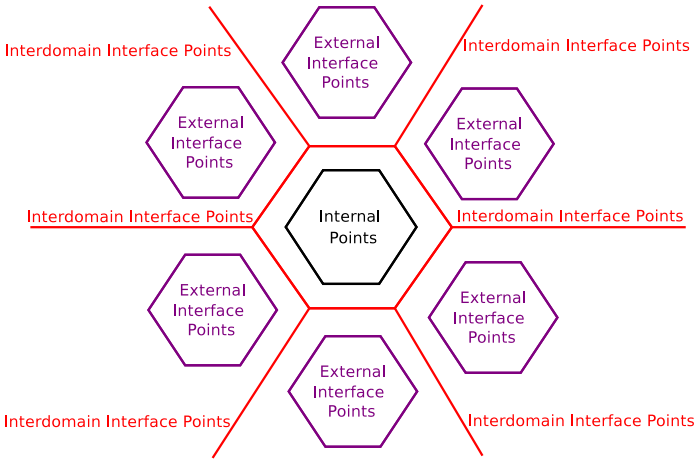

Figure 5.2: Distributed Group-Independent Sets

5.2 Global preconditioners in pARMS environment

There are three global preconditioners available in pARMS. The first one is an Restrictive Additive Schwarz procedure in which the local solver uses a local ARMS preconditioner. The second one is a Schur complement technique, in which the Schur complement relates to equations associated with local and inter-processor interface points. The third one applies if the system is very large, in this case it is necessary to extend the ARMS reordering [18] for the interdomain variables since the Schur complement system becomes costly.

Now we talk about preconditioning of distributed sparse linear systems. For instance, RAS, SCHUR, BJ.

RAS 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. Forward: calculate right hand side (rhs) $g'_i = g_i - E_i B_i^{-1} f_i$
2. Solve global Schur complement system $Sy = g'$, with

$$S_i y_i + \sum_{j \in N_i} E_{ij} y_j = g_i - E_i B_i^{-1} f_i \equiv g'_i, \tag{5.1}$$

3. Backward : calculate u_i with $B_i u_i = f_i - E_i y_i$.

BJ preconditioner

1. Forward: calculate right hand side (rhs) $g'_i = g_i - E_i B_i^{-1} f_i$
2. Solve global Schur complement system $Sy = g'$, with

$$S_i y_i + \sum_{j \in N_i} E_{ij} y_j = g_i - E_i B_i^{-1} f_i \equiv g'_i, \rightsquigarrow y_i + S_i^{-1} \sum_{j \in N_i} E_{ij} y_j = S_i^{-1} \left[g_i - E_i B_i^{-1} f_i \right]. \quad (5.2)$$

3. Backward : calculate u_i with $B_i u_i = f_i - E_i y_i$.

Parallel Schur complement preconditioning

Let us look deep into the framework of parallel Schur complement [14]. The ARMS procedure disscussed above yields a comprehensive framework that can be exploited to develop other well-known algorithms. In fact, the only operations required are : 1. the descend and ascend procedure, 2. the generation of the Schur complement. In ARMS, these are defined from incomplete GE. In Algebraic Multigrid (AMG), they are defined from generic restrictions and prolongation operators. Much of AMG research is centered around defining good restrictions and prolongations. If the ARMS software is written in an inclusive manner, it is perfectly possible to have a unique meta-algorithm which yields as particular cases either AMG or various ARMS techniques. It is important to have such polymorphism available for conducing research on robust algorithms.

The versatility of the ARMS framework may be clearly seen in its handling of the last-level Schur complement. In this work, we propose to focus on solving the linear system defined by this Schur complement A_k, where k is the last level. As we have stated earlier, the matrix A_k may be very poorly conditioned because the existing ARMS multilevel procedure pushes "bad" nodes to the end. Currently, ARMS uses dropping to sparsify this system followed by an incomplete LU solve (with partial pivoting). Possibly convergence-impaired, this procedure, however, enables the matrix A_k to be relatively large due to its controlled sparsity. In the realm of exsacale computing power, we propose to keep the last Schur complement as dense (or dense block) for the sake of both convergence and effcient execution using accelerator technologies, such as GPUs. In addition, we propose to use the RBT to precondition matrix A_k, to avoid partial pivoting and to achieve efficient implementations on GPUs and/or MIC architectures.

5.3 Local preconditioners in pARMS environment

In pARMS, we can use three local Incomplete LU preconditioners [20], such as Incomplete LU factorization level 0 (ILU0), Incomplete LU factorization level k (ILUK), Incomplete LU factorization with Threshold (ILUT), and one multilevel preconditioner such as Algebraic Recursive Multilevel Solver (ARMS). During this internship we added our local preconditioner ARMS with RBT, which we call ARMS_RBT.

ILU0 preconditioner

ILU0 means incomplete LU factorization technique with no fill-in, consisting of taking the zero pattern P to be precisely the zero pattern of A. Assuming L has the same structure as the lower part of A, U has the same structure as the upper part of A, if we take the product $L \times U$, the resulting matrix has the same pattern. In general, it is impossible to match A with product of $L \times U$, because product of $L \times U$ has extra diagonals, named diagonal with offsets $n_x - 1$ and $-n_x + 1$. The entries in these extra diagonals are called fill-in elements. If these fill-in elements are ignored, then it is possible to find L and U, which leads to A equal to product of $L \times U$ in the other diagonals.

ILUK preconditioner

To define ILU(k) with the same example as before, the ILU(1) factorization results from taking P to be the zero pattern of the product $L \times U$ of factors L, U obtained from ILU(0). In other words, the fill-in positions created in this product belong to the augmented pattern $NZ_1(A)$, but their actual values are zero. So with this regulation we could obtain ILU(k) factorization.

ILUT preconditioner

The generic ILU algorithm with threshold can be obtained from the IKJ version of Gaussian elimination by including a set of rules for dropping small elements. Applying a dropping rule to an element means replacing the element by zero if it satisfies a set of criteria. A dropping rule can be applied to a whole row by applying the same rule to all the elements of the row.

ARMS preconditioner

ARMS-solve(A_l, b_l) – Recursive Multi-Level Solution [19]

1. Solve $L_l \; f'_l = f_l$
2. Descend, i.e., compute $h'_l := h_l - E_l U_l^{-1} \; f'_l$
3. If $l = last\_lev$ then
4. Solve $A_{l+1} z_l = h'_l$ using ILUT factors
5. Else
6. Call ARMS-solve(A_{l+1}, h'_l)
7. Endif
8. Ascend, i.e., compute $f''_l = f'_l - L_l^{-1} F_l \; z_l$
9. Back-Substitute $y_l = U_l^{-1} f''_l$

ARMS_RBT preconditioner

ARMSRBT-solve(A_l, b_l) – Recursive Multi-Level Solution with RBT

1. Solve $L_l \; f'_l = f_l$
2. Descend, i.e., compute $h'_l := h_l - E_l U_l^{-1} \; f'_l$
3. If $l = last\_lev$ then
4. Solve $A_{l+1} z_l = h'_l$ using RBT + LU
5. Else
6. Call ARMSRBT-solve(A_{l+1}, h'_l)
7. Endif
8. Ascend, i.e., compute $f''_l = f'_l - L_l^{-1} F_l \; z_l$
9. Back-Substitute $y_l = U_l^{-1} f''_l$

5.4 Integration of RBT into pARMS

Analyzing the whole framework

Our second part of internship consits of procedures as below. To begin with, we learn the struture used in pARMS. It has three types of structure, for instance *parms_arms_data*, *p4ptr*, *ilutptr*, and another special stucture to store the matrix called *SparRow*. Now, let me explain the roles of these structure. The first structure *parms_arms_data* stores the interlevel structure *p4ptr*, the last level *ilutptr*, parameters for solving the pARMS system, and information about the size of the matrix, the number of matrix levels and the number of nonzeros in the original and preconditioned matrices. The second structure *p4ptr* consists of information about interlevel matrices, each has four parts : its decompotion as L, U, E, F matrices, the permutations as rperm, perms, sysperms, the current rhs, and the pointers prev to previous level and next to the next level. The third structure *ilutptr* contains information about the last level matrix, there is the last Schur complement matrix C, its decomposition as L and U matrices, the permutations as rperm, perms, perm2, and information about the current rhs. In the end, the special structure *SparRow* has the number of lines, an array containing the length of each row, an array of pointers to store column indices and an array of pointers to store nonzero entries.

Analyzing the code with global vision, which helps understanding the global structure of the source code. We used Callgrind to analyze it, and we learned that pARMS managed the parallel part by using global preconditioning with MPI instructions, while the local part, more precisely the local preconditioning does not use the MPI instructions and is therefore standalone. Thus in our work, we do not need to take care about parallelism, since pARMS does it by itself. The next step was to analyze the code locally.

Then, we analyze the code in details, and we found that it was corresponding to the local preconditioning, such as *ilu*0, *ilut*, *iluk* and *arms*. Then we focused on our target arms, because it is the basis for *arms_rbt*. When we learned how did *arms* work, then we could integrate our RBT into *arms* and make it behave like arms. In fact, by analyzing *arms*, we have created similar functions which related to *arms*, in this way we keep coherence with the orginal code and take profit from the original framework. The essential part resides in the last Schur complement, where we implemented RBT so we could give the preconditioned matrix to the fgmres solver to solve the linear system. The rest of the code behaves exactly as *arms*.

5.5 Experimental results

After having integrated RBT in the sequential ARMS solver, we continued our work by considering the parallel solver pARMS. However, when we integrated RBT in pARMS, we did not accelerate the solution time, but we decreased the number of iterations to reach convergence (see Figure 5.3), at the same time we improved the accuracy of the solution (see Figure 5.4). The reason for not accelerating the solution time is that in the last Schur complement, we convert the matrix from sparse to dense format.

Despite we cannot reduce the solving time, even if we do not permute the lines and rows which needs communications among processors, we can solve the system with less iterations and better accuracy, which means that convergence is faster and that the obtained solution is more accurate. This can be illustrated by Figure 5.3 and Figure 5.4. Considering the performance between arms and *arms_rbt*, the latter with RBT performs much better than alone.

In this experiment, we used one of the matrix from Davis' collection [11] to test the performance of different preconditioners. The test matrix called SHERMAN5 is a real unsymmetric matrix, of size 3312×3312, and with 20793 non zeros. SHERMAN5 matrix 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 important configurations are as follows : the tolerance for inner iteration is 0.01, the tolerance for outer iteration is 1.0e-6, the number of levels for ARMS is 1, the block size for block independent sets is 250, the tolerance used in independent set is 0.4, the krylov subspace size for outer iteration is 20, the outer fgmres iteration is 200, the droptol of matrices L, U, $L^{-1}F$ and EU^{-1} is 0.00001, the droptol of Schur complements at each level is 0.001, the droptol of ILUT in last level Schur complement is 0.001 for ARMS, while 0 for ARMS_RBT.

The platform we have features 48GB of RAM and two Intel Xeon E5645 at 2.40GHz, each with 6 cores and hyperthreading disabled, which gives a total of 12 cores. During this experiment we tested with various number of cores, 2, 4, 6, 8, 10, 12 respectively. In the following figures, we can compare the performance of different preconditioners in terms of iterations required for convergence and the accuracy. From these figures, we learn that solver ARMS with RBT performs better than solver ARMS alone in most cases.

Figure 5.3 shows the comparison of performance in terms of iterations required for convergence. We compare the results of each global preconditioners RAS, SCHUR and BJ respectively with four local preconditioners iluk, ilut, arms, arms_rbt. We observe that most of the time, arms_rbt performs better than all the other local preconditioners, and in the worst case, it performs as well as arms.

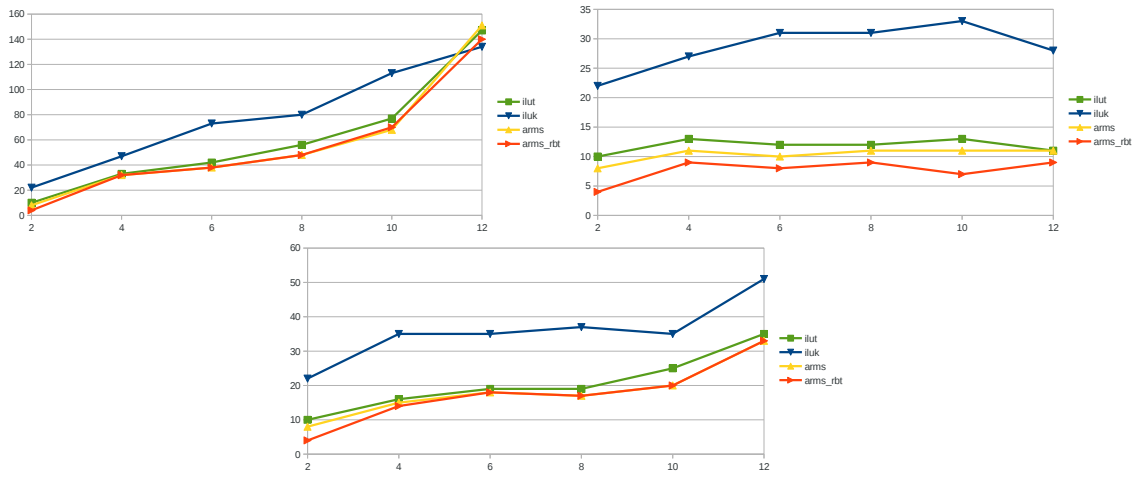

Figure 5.3: Iterations required for convergence

Figure 5.4 shows the comparison of performance in terms of accuracy of results. We compare the results of each global preconditioners RAS, SCHUR and BJ respectively with four local preconditioners iluk, ilut, arms, arms_rbt. We observe that when we compare to other preconditioners, arms_rbt performs better or similarly to arms when used with RAS and SCHUR global preconditioners, when dealing with BJ, the results are particular since accuracy depends on the number of processors used.

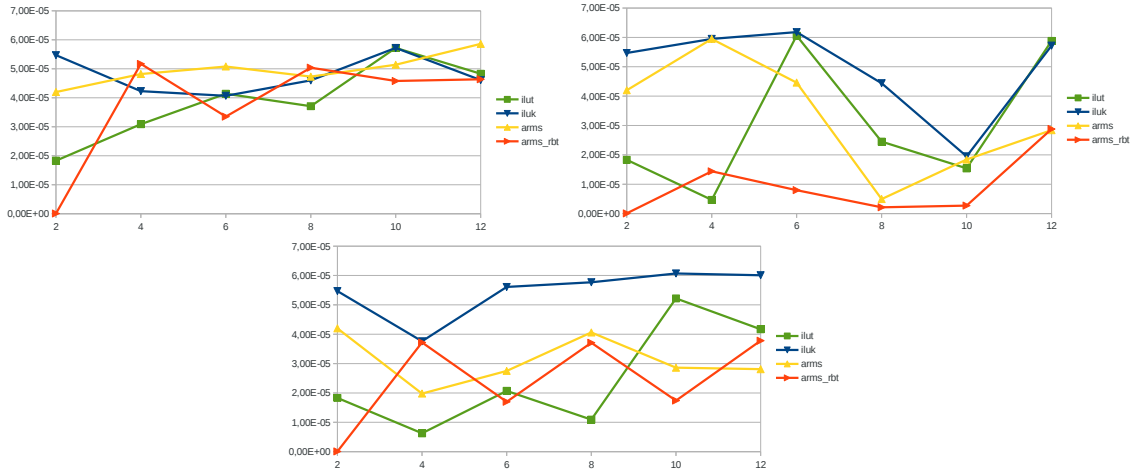

Figure 5.4: Accuracy of results

After testing with matrix SHERMAN5, we also did some experiments by using other matrices such as raefsky3 and SHERMAN3. With matrix raefsky3 we improved the performance by achieving better accuracy, but arms and arms_rbt required the same number of iterations for convergence. While with matrix SHERMAN3, we learned that integrating RBT into pARMS did not influence the iterations required for convergence, nor the accuracy of results.

CHAPTER 6

Conclusion

The objective of this internship was to integrate randomization using RBT in the ARMS solver, which requires advanced knowledge of both ARMS and RBT. This knowledge was acquired by learning state-of-the-art methods for solving linear systems.

As a proof of concept, we obtained preliminary results using sequential ARMS. Our experiments showed an acceleration of the convergence and better time results.

Our next task was to implement RBT into parallel ARMS. We expected to have a speed up in the computations, while it was not always the case. Indeed, our integration of RBT in pARMS requires a conversion of sparse matrices to a dense format. Depending on the sparsity of the last Schur complement, we may obtain a bigger matrix which takes more time to process, anihilating the benefit of using RBT.

However, we obtained an improvement in terms of number of iterations and accuracy of results. This confirms the interest of using RBT to solve sparse linear systems using iterative methods.

Acknowledgements

I am really thankful to my advisor Pr. Marc Baboulin. He is the constructor of the implementation of statistical technique RBT. He supported my work with a lot of patience. Since I was a beginner in the domain of numerical linear algebra, he taught me a lot of background knowledge.

I am also thankful to my other advisor Pr. Masha Sosonkina. She is one of the authors of the software pARMS that I used during this internship.

Bibliography

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

[2] M. Baboulin, J. Dongarra, and S. Tomov. Some issues in dense linear algebra for multicore and special purpose architectures. In *9th International Workshop on State-of-the-Art in Scientific and Parallel Computing (PARA'08)*, volume 6126-6127 of *Lecture Notes in Computer Science*. Springer-Verlag, 2008.

[3] Marc Baboulin, Dulceneia Becker, George Bosilca, Anthony Danalis, and Jack Dongarra. An efficient distributed randomized algorithm for solving large dense symmetric indefinite linear systems. *Parallel Computing*, 2013.

[4] Marc Baboulin, Simplice Donfack, Jack Dongarra, Laura Grigori, Adrien Rémy, and Stanimire Tomov. A class of communication-avoiding algorithms for solving general dense linear systems on CPU/GPU parallel machines. Rapport de recherche RR-7854, INRIA, January 2012.

[5] Marc Baboulin, Jack Dongarra, Julien Herrmann, and Stanimire Tomov. Accelerating linear system solutions using randomization techniques. *ACM Trans. Math. Softw.*, 39(2):8:1–8:13, February 2013.

[6] Marc Baboulin, Jack Dongarra, and Stanimire Tomov. Some issues in dense linear algebra for multicore and special purpose architectures. lapack working note 200, 2008.

[7] Bradley N. Bond and Luca 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, pages 728–735, Piscataway, NJ, USA, 2008. IEEE Press.

[8] Alfredo Buttari, Julien Langou, Jakub Kurzak, and Jack Dongarra. A class of parallel tiled linear algebra algorithms for multicore architectures. *Parallel Comput.*, 35(1):38–53, January 2009.

[9] Zhi-Hao Cao. Constraint schur complement preconditioners for nonsymmetric saddle point problems. *Appl. Numer. Math.*, 59(1):151–169, January 2009.

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

[11] Timothy A. Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25, December 2011.

[12] Simplice Donfack, Jack Dongarra, Mathieu Faverge, Mark Gates, Jakub Kurzak, Piotr Luszczek, and Ichitaro Yamazaki. On Algorithmic Variants of Parallel Gaussian Elimination: Comparison of Implementations in Terms of Performance and Numerical Properties. Rapport de recherche, Innovative Computing Laboratory - ICL , HiePACS - INRIA Bordeaux - Sud-Ouest, 2013.

[13] James E. Gentle, Wolfgang Härdle, and Yuichi Mori, editors. *Handbook of Computational Statistics*. Springer, 2004.

[14] Luc Giraud, Azzam Haidar, and Yousef Saad. Sparse approximations of the Schur complement for parallel algebraic hybrid linear solvers in 3D. Rapport de recherche RR-7237, INRIA, March 2010.

[15] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 1996. Third edition.

[16] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2002.

[17] William Kahan. Accurate eigenvalues of a symmetric tri-diagonal matrix. Technical report, Stanford, CA, USA, 1966.

[18] Daniel Kressner. Block algorithms for reordering standard and generalized schur forms. *ACM Trans. Math. Softw.*, 32(4):521–532, December 2006.

[19] Zhongze Li, Yousef Saad, and Masha Sosonkina. pARMS: a parallel version of the algebraic recursive multilevel solver. *Numerical Linear Algebra with Applications*, 10(5-6):485–509, 2003.

[20] Scott MacLachlan, D. Osei-Kuffuor, and Yousef Saad. Modification and compensation strategies for threshold-based incomplete factorizations. *SIAM J. Scientific Computing*, 34(1), 2012.

[21] D. Stott Parker. Random butterfly transformations with applications in computational linear algebra. Technical Report CSD-950023, University of California Los Angeles, CA USA, 1995.

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

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

[24] Barry F. Smith, Petter E. Bjørstad, and William D. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, New York, NY, USA, 1996.

[25] Barry F. Smith, Petter E. Bjørstad, and William D. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, New York, NY, USA, 1996.

[26] Stanimire Tomov, Jack Dongarra, and Marc Baboulin. Towards dense linear algebra for hybrid GPU accelerated manycore systems. *Parallel Comput.*, 36(5-6):232–240, June 2010.