FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Rostislav Slevínský

Reduced communication algoritms: theory and practice

Department of Numerical Mathematics

Supervisor of the bachelor thesis:	prof. Ing. Miroslav Tůma, CSc
Study programme:	Mathematics
Study branch:	General Mathematics

Prague 2019

I am most grateful to my supervisor for his time, patience and guidance. I would like to thank to my family for their support.

Title: Reduced communication algoritms: theory and practice

Author: Rostislav Slevínský

Department: Department of Numerical Mathematics

Supervisor: prof. Ing. Miroslav Tůma, CSc, Department of Numerical Mathematics

Abstract: Development in the parallel computing environment in the last decade comes with the need of being able to use these in solving large algebraic systems. In this thesis, we focus on the Krylov subspace methods (namely the conjugate gradient method) as one of the most powerful tools and the possibilities of their parallelization. We discuss the communication avoiding Krylov subspace methods and various problems introduced by the parallelization e.g. loss of orthogonality or delay of convergence. Application of the Krylov subspace methods comes usually with some preconditioner, therefore part of this thesis is dedicated to the preconditioning in parallel computing environments.

Keywords: Krylov space methods parallel computer architectures communication avoiding reduced commmunication

Contents

Introduction

Science and engineering generally are nowadays connected with the field of computational science more than ever before. Computational science has become a vital part of scientific exploration and it not only allows mankind to study new phenomena but it also makes science progress cheaper and less time-consuming. This connection puts emphasis on the further development of new computational methods as well as on the improvement of the older methods. To achieve either, one must have a clear and thorough understanding of what the performance bottlenecks are, what are the possibilities and ways to overcome them and, most importantly, what effects do the changes to the methods have on the numerical behaviour i.e. numerical stability, convergence and maximal attainable accuracy.

In this text, we are interested particularly in Krylov subspace methods (sometimes referred to as KSMs) as one of the most powerful tools for solving large scale algebraic systems

$$Ax = b. \tag{1}$$

Krylov subspace methods are members of iterative methods class, which makes them less suitable for parallelization as there is a lot of communication and synchronization needed on the global level. The classical approach to the performance analysis of a method originated from its computational complexity, i.e. the lesser the number of arithmetic operations performed is, the faster the method will be. However, with modern possibilities of parallelism, communication complexity is the better indicator. Hence, the goal is to reduce the number of synchronization points in the algorithms. There are several approaches to this, including so-called s-step methods or pipelined conjugate gradients.

Traditionally, the representation of Krylov subspace methods includes matrix-vector multiplication and dot products (inner products) in every iteration. Both matrix-vector multiplication and dot products are communication bounded operations, meaning that it is the data movement rather than the computational costs that are the performance-limiting factor. This gave rise to other class of Krylov subspace methods called communication avoiding Krylov Subspace methods (also referred to as CA-KSMs). The goal of this thesis is to give a state-of-the-art on part of the CA-KSMs family. In this thesis we will focus on conjugate gradients method and the various modifications to this method that reduce communication, we will point out the so-called s-step Krylov subspace methods and give a brief summary of recent work on their stability and convergence in finite precision. Then we will discuss possibilities to improving the performance of these methods, particularly the preconditioning.

1. Conjugate gradient method

In this section, we will introduce the conjugate gradient method as one of the main Krylov subspace methods. Before doing so, we define some important terms. All vectors and matrices are considered real in this thesis in order to make notation simpler.

Definition 1. *A symmetric real matrix A is said to be positive definite if*

$$x^T A x > 0 \tag{1.1}$$

for every $x \in \mathbb{R}^n$.

Definition 2. *Methods which use the Krylov subspaces*

$$\mathcal{K}_k(A, v) \equiv \operatorname{span}\{v, Av, \ldots, A^{k-1}v\}, \tag{1.2}$$

to search for the approximate solution x_k are said to be the Krylov subspace methods.

The n-th iteration of a Krylov subspace method can be viewed as a projection onto the n-th Krylov space orthogonal to some other so-called constraint space. Given initial approximation x_0 and initial residual vector $r_0 = b - Ax_0$, the n-th approximation holds $x_n \in x_0 + S_n$ where the S_n is called the search space and the corresponding $r_n \perp C_k$ is said to be the constraint space. The choice of this search and constraint space determines various Krylov subspace methods. For the conjugate gradient method the choice is

$$S_k = C_k = \mathcal{K}_k(A, r_0). \tag{1.3}$$

1.1 Standard conjugate gradient method

Conjugate gradient method is an algorithm that is used to solve systems of linear equations (1) where the matrix A is symmetric positive-definite and not necessarily but usually sparse as well. This method was first introduced by Hestenes and Stiefel in [9] who noticed the connection between solving the system and minimizing a functional along some line. The functional, sometimes called *the error function*, has the following form:

$$F(x) = \frac{1}{2} x^T A x - x^T b. \tag{1.4}$$

In its classical form here represented as introduced by Hestenes and Stiefel in the original work it is required to compute two dot products per iteration resulting in high communication costs. Now we will present the HS algorithm.

Algorithm 1.1: Hestenes - Stiefel CG

```
input : SPD matrix $A \in \mathbb{R}^{n\times n}$, vector $b \in \mathbb{R}^n$, initial approximation
        $x_0 \in \mathbb{R}^n$, stopping criterion
output: Approximate solution $x_n$
1 $r_0 = b - Ax_0$;
2 for $i = 1$ to $nmax$ do
3     $\alpha_{i-1} = \frac{(r_{i-1},r_{i-1})}{(p_{i-1},Ap_{i-1})}$;
4     $x_i = x_{i-1} + \alpha_{i-1}p_{i-1}$;
5     $r_i = r_{i-1} - \alpha_{i-1}Ap_{i-1}$;
6     $\beta_i = \frac{(r_i,r_i)}{(r_{i-1},r_{i-1})}$;
7     $p_i = r_i + \beta_i p_{i-1}$;
8 end
```

From line 5 in algorithm 1.1 we can see that in order to compute the coefficient β_i one needs to have the coefficient α_i already precomputed. Hence lines 3 and 6 cannot be computed in parallel, resulting in two global synchronization points per iteration. This represents performance bottleneck on parallel systems.

One of the possible ways to reduce the number of global synchronization points is to use some additional equalities to compute the dot products.

1.2 Three-term recurrence

Stiefel in [14] applied three-term recurrences for orthogonal polynomials with a certain choice of density function on the residual polynomials to obtain the three-term recurrences for the residual vectors. Let us assume that the solution approximations are updated as $x_{i+1} = x_i + \Delta x_i$, where the quantity Δx_i can be rewritten as

$$\Delta x_i = \frac{1}{q_i} r_i, \tag{1.5}$$

where the values of q_i distinguish different methods. Now we can use these coefficients to form a polynomial of degree k:

$$R_n(\lambda) = (1 - \frac{\lambda}{q_0})(1 - \frac{\lambda}{q_1})\dots(1 - \frac{\lambda}{q_{n-1}}). \tag{1.6}$$

By induction we get

$$x_n = \frac{1 - R_n(A)}{A} b \tag{1.7}$$

and finally using relation $Ax_n - b = -R_n(A)b$ we obtain the residual polynomial:

$$r_n = R_n(A)b \ , \text{ where } R_n(0) = 1. \tag{1.8}$$

Stiefel then used a recurrence from [15] to force orthogonality between the residual polynomials:

$$\lambda R_i(\lambda) = -q_i R_{i+1}(\lambda) + t_i R_i(\lambda) - p_i R_{i-1}(\lambda) \tag{1.9}$$

Note not to confuse the p_i with the direction vectors from the CG method. Set $t_i = p_i + q_i$, $R_0 = 1, p_0 = 0$. Applying this to the $x_{i+1} = x_i + \Delta x_i$ formula we obtain a three-term recurrence for the approximate solution:

$$x_{i+1} = x_i + \frac{1}{q_i}(r_i + p_i(x_i - x_{i-1})). \tag{1.10}$$

This result was later used by Rutishauser in [12] and transformed into the following theorem.

Theorem 1. *([12], Theorem 1) For a gradient method, the residual vector r_k may be represented as*

$$r_k = R_k(A)r_0, \tag{1.11}$$

where the $R_k(\lambda)$ are polynomials of degree k; they are called the residual polynomials of the method.

In Algorithm 1.2 we present a variant of the conjugate gradient method from [1], where the approximate solution x_i and the residual r_i are computed using the three term recurrences.

```
Algorithm 1.2: three-term CG
   input  : SPD matrix $A \in \mathbb{R}^{n\times n}$, vector $b \in \mathbb{R}^n$, initial approximation
            $x_0 \in \mathbb{R}^n$, stopping criterion
   output: Approximate solution $x_n$
1  $r_0 = b - Ax_0, x_{-1} = x_0, r_{-1} = r_0, e_{-1} = 0$;
2  for $i = 1$ to $nmax$ do
3      $q_{i-1} = \frac{(r_{i-1}, Ar_{i-1})}{(r_{i-1}, r_{i-1})} - e_{i-2}$;
4      $x_i = x_{i-1} + \frac{1}{q_{i-1}}[r_{i-1} + e_{i-2}(x_{i-1} - x_{i-2})]$;
5      $r_i = r_{i-1} + \frac{1}{q_{i-1}}[-Ar_{i-1} + e_{i-2}(r_{i-1} - r_{i-2})]$;
6      $e_{i-1} = q_{i-1}\frac{(r_i, r_i)}{(r_{i-1}, r_{i-1})}$;
7  end
```

Following observation from [1] will allow us to rewrite these recurrences in the notation from the Algorithm 1.1. This will be useful for two reasons:

(i) to show mathematical equivalence with the standard conjugate gradient method,

(ii) and to state the relations between other modifications of the CG method, that are all, in fact, using the same three-term recurrence.

Claim 2. *Algorithms 1.1 and 1.2 produce the same sequences of iterates x_i and residuals r_i for $i = 0, \ldots$. Moreover, we have $q_i = 1/a_i$ and $e_i = q_i\beta_{i+1}$ for $i = 0, \ldots,$ where α_i and β_{i+1} are coefficients from Algorithm 1.1.*

Proof. ([1], Observation 2.1) Consider q_i and e_i from Algorithm 1.2. It is clear that $q_0 = 1/\alpha_0$ and $e_0 = \beta_1 q_0$. Let $i > 1$, then using notation from Algorithm

1.1, induction and orthogonality relations between the quantities (in exact arithmetics), we get

$$q_{i-1} = \frac{(r_{i-1}, Ar_{i-1})}{(r_{i-1}, r_{i-1})} - e_{i-2} = \frac{(p_{i-1}, Ap_{i-1})}{(r_{i-1}, r_{i-1})} + \frac{(\beta_{i-1}p_{i-2}, \beta_{i-1}Ap_{i-2})}{(r_{i-1}, r_{i-1})} - q_{i-2}\beta_{i-1}$$
$$= \frac{1}{\alpha_{i-1}} + \frac{\beta_{i-1}^2}{\beta_{i-1}\alpha_{i-2}} - q_{i-2}\beta_{i-1} = \frac{1}{\alpha_{i-1}}.$$

Now we can use from Algorithm 1.1 line (4) the relation for approximate solution $x_i = x_{i-1} + \alpha_{i-1}p_{i-1}$ and using

$$\begin{aligned} x_{i-1} - x_{i-2} &= x_{i-2} + \alpha_{i-2}p_{i-2} - x_{i-2} \\ \alpha_{i-2}p_{i-2} &= x_{i-1} - x_{i-2} \\ p_{i-2} &= \frac{x_{i-1} - x_{i-2}}{\alpha_{i-2}}, \end{aligned}$$

we can obtain the three-term recurrence for the approximate solution x_i

$$\begin{aligned} x_i &= x_{i-1} + \alpha_{i-1}p_{i-1} \\ &= x_{i-1} + \alpha_{i-1}r_{i-1} + \alpha_{i-1}\beta_{i-1}p_{i-2} \\ &= x_{i-1} + \alpha_{i-1}r_{i-1} + \alpha_{i-1}\beta_{i-1}\frac{x_{i-1} - x_{i-2}}{\alpha_{i-2}} \\ &= x_{i-1} + \alpha_{i-1}[r_{i-1} + \frac{\beta_{i-1}}{\alpha_{i-2}}(x_{i-1} - x_{i-2})] \\ &= x_{i-1} + \frac{1}{q_{i-1}}[r_{i-1} + e_{i-2}(x_{i-1} - x_{i-2})]. \end{aligned}$$

The three-term recurrence for approximate residual can be obtained the same way. □

Note that this equivalence does not hold in the finite arithmetics.

This allows us to compute α_i from β_{i-1} with formula 1.12 which eliminates the global synchronization phase bounded with (p_i, Ap_i) dot product.

$$\alpha_i = \frac{(r_i, r_i)}{(Ar_i, r_i) - (\beta_{i-1}/\alpha_{i-1})(r_i, r_i)}. \tag{1.12}$$

One of the early approaches proposed to use the following formula

$$(r_i, r_i) = \alpha_{i-1}^2(Ap_{i-1}, Ap_{i-1}) - (r_{i-1}, r_{i-1}). \tag{1.13}$$

Which allows us to compute β_i using α_i :

$$\beta_i = \frac{\alpha_i^2(Ap_{i-1}, Ap_{i-1}) - (r_{i-1}, r_{i-1})}{(r_i, r_i)}. \tag{1.14}$$

This eliminates the global synchronization phase bounded with the other dot product (r_i, r_i). However, it has been observed by Saad in [13] that this leads to an unstable algorithm.

1.3 Methods based on additional recurrences

The Algorithm 1.2 or the idea to compute the α_i from β_{i-1} have been reintroduced several times by different authors. All following algorithms use the same three-term recurrence, therefore, are all variations of the Algorithm 1.2.

In [6] authors set their goal to overleap the (p_i, Ap_i) dot product. Their Algorithm 1.3 again leads to computing α_i from β_{i-1}. However, the authors did not notice the connection to the work of Stiefel or the Algorithm 1.2.

$$\begin{aligned}(p_i, Ap_i) &= (r_i + \beta_i p_{i-1}, Ar_i + \beta_i Ap_{i-1})\\ &= (r_i, Ar_i) + \beta_i(r_i, Ap_{i-1}) + \beta_i(p_{i-1}, Ar_i) + \beta_i^2(p_{i-1}, Ap_{i-1})\\ &= (r_i, Ar_i) + 2\beta_i(r_i, Ap_{i-1}) + \beta_i^2(p_{i-1}, Ap_{i-1}). \qquad (1.15)\end{aligned}$$

We have used the fact that the matrix A is symmetric positive definite and using another property of conjugate gradient method - the orthogonality of the residual vectors i.e. $(r_i, r_{i+1}) = 0$, this can be simplified even further:

$$\begin{aligned}r_i &= r_{i-1} + \alpha_i Ap_{i_1}\\ (r_i, r_i) &= (r_i, r_{i-1}) + \alpha_i(r_i, Ap_{i-1})\\ &= -\alpha_i(r_i, Ap_{i-1}) \qquad (1.16)\end{aligned}$$

Algorithm 1.3: D'Avezedo and Romine

```
   input : SPD matrix $A \in \mathbb{R}^{n\times n}$, vector $b \in \mathbb{R}^n$, initial approximation
           $x_0 \in \mathbb{R}^n$, stopping criterion
   output: Approximate solution $x_n$
1  initialization by performing one iteration of classical conjugate gradient
     algorithm;
2  $r_1 = b, \gamma_1 = (r_1, r_1), p_1 = r_1, v_1 = Ap_1, \sigma_1 = (p_1, v_1), x_2 = (\gamma_1/\sigma_1)p_1$;
3  for $k = 2$ to $nmax$ do
4      $s_k = Ar_k$;
5      $\gamma_k = (r_k, r_k)$;
6      $\delta_k = (r_k, s_k)$;
7      $\beta_k = \frac{\gamma_k}{\gamma_{k-1}}$;
8      $p_k = r_k + \beta_k p_{k-1}$;
9      $v_k = s_k + \beta_k v_{k-1}$;
10     $\sigma_k = \delta_k - \beta_k^2 \sigma_{k-1}$;
11     $\alpha_k = \frac{\gamma_k}{\sigma_k}$;
12     $x_{k+1} = x_k + \alpha_k p_k$;
13     $r_{k+1} = r_k - \alpha_k v_k$;
14 end
```

Note the additional recurrence to avoid computing the A-multiples of the direction vectors $\{p_j\}$ directly. This also requires slightly more storage.

Authors Chronopoulos and Gear in [5] introduced another algorithm based on the three-term variant. This variant is in fact very similar to the Algorithm 1.3

because both use additional recurrence to compute A-multiples of the direction vectors

$$Ap_i = Ar_i + \beta_{i-1}Ap_{i-1}, \tag{1.17}$$

which can be an additional source of instability and can cause loss of orthogonality. Chronopolous and Gear noticed this modification has potential for parallel computing because both dot products required in each step can be computed simultaneously and based their s-step method on it. Following algorithm 1.4 as given in [5] (Chapter 2, Algorithm 2.3) is, in fact, s-CG algorithm where s = 1.

Algorithm 1.4: ChG

```
input : SPD matrix $A \in \mathbb{R}^{n\times n}$, vector $b \in \mathbb{R}^n$, initial approximation
        $x_0 \in \mathbb{R}^n$, stopping criterion
output: Approximate solution $x_n$
1  $p_0 = r_0 = b - Ax_0$, $a_0 = \frac{(r_0,r_0)}{(Ar_0,r_0)}$, $b_{-1} = 0$;
2  for $i = 1$ to $nmax$ do
3      $p_i = r_i + \beta_{i-1}p_{i-1}$;
4      $Ap_i = Ar_i + \beta_{i-1}Ap_{i-1}$;
5      $x_{i+1} = x_i + \alpha_i p_i$;
6      $r_{i+1} = r_i + \alpha_i Ap_i$;
7      Compute and Store $Ar_{i+1}$;
8      Compute $(r_{i+1}, r_{i+1}), (Ar_{i+1}, r_{i+1})$;
9      $\beta_i = \frac{(r_{i+1},r_{i+1})}{(r_i,r_i)}$;
10     $\alpha_{i+1} = \frac{(r_{i+1},r_{i+1})}{(Ar_{i+1},r_{i+1})-(\beta_i/\alpha_i)((r_{i+1},r_{i+1}))}$;
11 end
```

1.4 S-step method

In the so-called s-step Krylov subspace methods, the process is divided into an outer loop and an inner loop. In the outer loop, we use s linearly independent directions to make a new Krylov subspace basis (or, in terms of projection process, we increase the Krylov subspace dimension by a factor of s) and then in the inner loop, we do the vector updates. This allows us to avoid the local orthogonality in order to achieve better data locality, hence keeping the ratio $(MemoryReferences)/(FloatingPointOperations)$ as low as possible.

The s-step conjugate gradient method was introduced in [5] and [4]. The idea was to use $\{r_i, ..., A^{s-1}r_i\}$ to increase the dimension of the i-th step Krylov subspace $\{r_0, ..., A^{is}r_0\}$ by a factor of s. These vectors are then used to get the new s direction vectors $\{p_i^1, ..., p_i^s\}$. In order to obtain these, we make vectors $\{r_i, ..., A^{s-1}r_i\}$ A-conjugate to the preceding directions $\{p_{i-1}^1, ..., p_{i-1}^s\}$. Eventually, the functional 1.4 is minimized simultaneously in all new s directions. The new residual r_{i+1} is then computed directly from the approximate solution x_{i+1}, so there is no need to compute the vectors Ap_i^j which can be used to compute r_{i+1} from r_i recursively. The Algorithm 1.5 is given as in the original text [5](Chapter 4, Algorithm 4.1).

In order to compute the coefficients $b_i^{(j,l)}$ and a_i^j we need to solve $s+1$ systems

Algorithm 1.5: s-step CG

```
  input : SPD matrix $A \in \mathbb{R}^{n \times n}$, vector $b \in \mathbb{R}^n$, $x_0 \in \mathbb{R}^n$, stopping
          criterion $nmax$
  output: Approximate solution $x_n$
1 $p_0^1 = r_0 = b - Ax_0$, ..., $p_0^s = A^{s-1}r_0$;
2 for $i = 0$ to $nmax$ do
3     select $a_i^j$ to minimize functional 1.4 in $x_{i+1} = x_i + a_i^1 p_i^1 + ... + a_i^s p_i^s$
       over span$\{x_i + \Sigma_{j=1}^s a_i^j p_i^j\}$;
4     compute $r_{i+1} = b - Ax_{i+1}$, $Ar_{i+1}$, ...,$A^{s-1}r_{i+1}$;
5     select $\{b_i^{j,l}\}$ to force A-conjugacy of $\{p_{i+1}^1, ..., p_{i+1}^s\}$, $\{p_i^1, ..., p_i^s\}$;
6     $p_{i+1}^1 = r_{i+1} + b_i^{(1,1)} p_i^1 + ... + b_i^{(1,s)} p_i^s$
      ⋮
      $p_{i+1}^s = A^{s-1}r_{i+1} + b_i^{(s,1)} p_i^1 + ... + b_i^{(s,s)} p_i^s$
7 end
```

of equations of order s. Let $W = \{(p_i^j, p_i^l)\}$, $1 \le j, l \le s$, then in the i-th iterate inner block, the systems are of form

$$\begin{aligned} W_{i-1}\mathbf{b}^j + \mathbf{c}^j &= 0 \\ W_i\mathbf{a} &= m_i, \end{aligned}$$

where $\mathbf{a} = [a_i^1, \ldots, a_i^s]$ is a vector of steplengths, $\mathbf{b}^j = [b_i^{j,1}, \ldots, b_i^{j,s}]$ and let us define matrix $B = [\mathbf{b}^1, \ldots, \mathbf{b}^s]$. The matrix W_i is an important element in the analysis of loss of orthogonality between the direction vectors. It turns out, that if the condition number of this matrix is big, one can expect larger errors in computing the coefficient $b_i^{(j,l)}$, therefore, this might cause orthogonality loss. It has been observed, that for $s > 5$ the convergence can be very slow, as the loss of the orthogonality between the direction vectors becomes significant. This phenomenon seems to be difficult to avoid while using the monomial bases. Possible solutions will be discussed later in Chapter 3.

Now we will introduce a theorem that has two major consequences:

- the s-CG algorithm converges,
- the s-CG finds the same solution as classical CG.

Theorem 3. *([5], Theorem 4.1) Let m be the degree of the minimal polynomial of r_0, and assume $m > (i+1)s$. Then the direction space P_i and the residuals R_i generated by the s-CG process for $i = 0, 1, ...$ satisfy the following relations:*

(i) *P_i is A-conjugate to P_j for $j < i$.*

(ii) *R_i is A-conjugate to R_j for $j < i - 1$.*

(iii) *$P_j, R_j, j = 0, ..., i$ form bases for the Krylov subspace $V_i = \{r_0, Ar_0, ..., A^{(i+1)s-1}r_0\}$.*

(iv) *r_i is orthogonal to V_{i-1}*

Proof. (follows original proof from [5]) For i=1 the theorem follows from definition of the quantities and the fact, that the algorithm minimizes the error function 1.4, which implies $r_{i+1} \perp \{r_i, \dots, A^{s-1}r_i\}$, which implies (iv) for i=1. Now we will use induction on i. Let us assume that the theorem holds for $i > 1$. Because $P_i = R_i + P_i B$ is by definition A-conjugate to the P_{i-1} it suffices to show that R_i is A-conjugate to $P_0, \dots P_{i-2}$. If we write:

$$P_j = R_j + \sum_{k=0}^{j-1} l_k[R_{k-1}],$$

where $l_k[R_{k-1}]$ is a linear combination of the vectors $R_0, \dots R_{j-1}$, the proof of (i) has been reduced to proving (ii).
Now $r_i = r_{i-1} - Ap_{i-1}\mathbf{a}$ is orthogonal to P_{i-1} (by definition) and to $P_0, \dots P_{i-2}$ (by the induction hypothesis). Hence, by (iii) r_i is orthogonal to V_{i-1} which proves (iv). To prove (ii) we must show that R_i is A-conjugate to R_j, $j = 0, \dots i-2$, or equivalently that r_i is orthogonal to $\{r_j, Ar_j, \dots, A^{2s-1}r_j\}$. This holds if $\{r_j, Ar_j, \dots, A^{2s-1}r_j\} \subset V_{i-1}$. And this holds (by the induction hypothesis on (iii)) if the degree $(A^{2s-1}r_j) \le is - 1$, or $(j+2)s - 1 \le is - 1$, or $j \le i - 2$. This proves (ii).
The vectors P_j and R_j for $j = 0, \dots, i$ are A-conjugate by blocks and they belong to the Krylov space V_i. Within each block, the vectors are linearly independent. If the contrary is assumed then there exists a polynomial $p(\lambda)$ of degree m such that $p(A)r_0$ and $m < (i+1)s$. Which is a contradiction. This proves (iii). □

Corollary 3.1. *In the exact arithmetic and given the same starting approximation x_0 for both s-CG and standard CG, the approximation x_i generated by s-CG is the same as x_{is} produced by standard CG. Specially, s-CG method converges in at most N/s steps.*

In this part, we briefly introduced the s-step CG. The Theorem 3 shows that the method converges. The goal to increase the number of the floating point operations over the number of memory references to improve the performance on parallel systems is achieved. However, the outcoming algorithm includes more computational work and has higher memory demands. We can also expect the convergence to delay if the condition number of the matrix W_i is big. This can be partially solved if the method is used together with some preconditioning method. This is still to be investigated further.

1.5 Pipelined methods

Ghysels and Vanroose in [8] introduced a method that uses additional recurrences and computations to overleap the inner products and seems to be similar to the Algorithm 1.4 (again, it is a modification of the three-term variant) as both use same update formulas for α_i, β_i and the additional recurrence for Ap_i. However, this method reduces the number of the global synchronization points even further. In the Algorithm 1.6 (Algorithm 2.3 [1]) we present the pipelined conjugate gradient method, how this method is called.

Algorithm 1.6: pipelined conjugate gradient method

```
input  : SPD matrix $A \in \mathbb{R}^{n \times n}$, vector $b \in \mathbb{R}^n$, initial approximation
         $x_0 \in \mathbb{R}^n$, stopping criterion
output: Approximate solution $x_n$
1  $r_0 = b - Ax_0, p_0 = r_0, s_0 = Ap_0, w_0 = Ar_0, z_0 = Aw_0, \alpha_0 = \frac{(r_0,r_0)}{(p_0,s_0)}$;
2  for $i = 1$ to $nmax$ do
3      $x_i = x_{i-1} + \alpha_{i-1} p_{i-1}$;
4      $r_i = r_{i-1} + \alpha_{i-1} s_{i-1}$;
5      $w_i = w_{i-1} + \alpha_{i-1} z_{i-1}$;
6      $q_i = Aw_i$;
7      $\beta_i = \frac{(r_i,r_i)}{(r_{i-1},r_{i-1})}$;
8      $\alpha_i = \frac{(r_i,r_i)}{(w_i,r_i)-(\beta_i/\alpha_{i-1})((r_i,r_i))}$;
9      $p_i = r_i + \beta_i p_{i-1}$;
10     $s_i = w_i + \beta_i s_{i-1}$;
11     $z_i = q_i + \beta_i z_{i-1}$;
12 end
```

The main difference between Algorithm 1.4 and Algorithm 1.6 is that in the pipelined method uses one additional recurrence to compute Ar_i. This is beneficial in parallel processing because the computations in the lines (6) - (7) in algorithm 1.6 can be computed simultaneously as the dot products do not depend on the Ar_i and therefore only one point of global synchronization is needed. However, these modifications (namely the additional recurrences) that enable overlapping the inner products and reorganization to remove sequential dependancy between matrix-vector products and inner products, are potential source of numerical instabilities, as they introduce an additional loss of orthogonality of the original conjugate gradient method.

2. CA-KSMs

In this Chapter, we discuss the concept behind avoiding communication in Krylov subspace methods, which is sort of similar to the idea behind the s-step conjugate gradient method. The main source for this section is [2], where the author introduced new communication avoiding Krylov subspace methods including the nonsymmetric methods and discussed their stability and convergence.

Speaking strictly mathematically, all of these methods are equivalent to their classical counterparts. However, this equivalence does no longer hold in finite precision. Analyses of the numerical properties of the communication avoiding methods are yet to be done in many cases. Although, these methods share the same basic ideas and principles, therefore some of the results can be generalized.

This CA-KSMs approach to overcome the communication bottlenecks can be summarized into following steps:

(i) Krylov subspace base is generated,

(ii) these bases vectors are then orthogonalized in a block,

(iii) finally, s iterations are performed.

It seems that the item (i) in the list plays an important role in determining the numerical behaviour of each method. The s-step variant of conjugate gradient method presented earlier in the thesis uses a monomial basis $\{r_i, \dots, A^{s-1}r_i\}$ to lift the s dimensions out of the i-th step Krylov subspace. These bases usually have poor condition number and the convergence has been observed to be slow for $s > 5$. This is because for increasing s the monomial basis $\{r_i, \dots, A^{s-1}r_i\}$ converges to the eigenvector corresponding to the dominant eigenvalue of the matrix A. Therefore, the basis becomes ill-conditioned and prevents the method from convergence.

This lead to the proposal of using better conditioned polynomial bases. There are several possible strategies to obtain a polynomial basis for a Krylov subspace, for example, using Newton polynomials or Chebyshev polynomials.

In [2] it is assumed that these bases are computed via a polynomial three-term recurrence:

$$\begin{aligned} &\rho_0(z) = 1, \quad \rho_1(z) = (z - \hat{\alpha}_0)\rho_0(z)/\hat{\gamma}_0, \quad \text{and} \\ &\rho_j(z) = ((z - \hat{\alpha}_{j-1})\rho_{j-1}(z) - \hat{\beta}_{j-2}\rho_{j-2}(z))/\hat{\gamma}_{j-1}, \quad \text{for } j > 1 \end{aligned} \tag{2.1}$$

The basis vectors generation can be written in a matrix form:

$$A\underline{\mathcal{Y}}_k = \mathcal{Y}_k \mathcal{B}_k. \tag{2.2}$$

The matrix $\mathcal{B}_k \in \mathbb{R}^{(i+1)\times(i+1)}$ is of the form, where the recurrence coefficients depend on the strategy (Newton, Chebyshev,...)

$$\begin{bmatrix} \hat{\alpha}_0 & \hat{\beta}_0 & & & 0 \\ \hat{\gamma}_0 & \hat{\alpha}_1 & \ddots & & \vdots \\ & \hat{\gamma}_0 & \ddots & \hat{\beta}_{i-2} & 0 \\ & & \ddots & \hat{\alpha}_{i-1} & 0 \\ & & & \hat{\gamma}_{i-1} & 0 \end{bmatrix}. \tag{2.3}$$

By condition number of a basis, we mean a condition number of a matrix, that has the basis as its columns. Now we define the condition number of a matrix.

Definition 3. *Let A be a nonsingular square matrix. The number $\kappa(A)$ that satisfies*

$$\kappa(A) = \|A\|\|A^{-1}\|$$

is said to be the condition number of matrix A.

2.1 Lanczos method

Lanczos method is a special case of the Arnoldi method (applied to SPD matrix) which is a powerful tool for finding the eigenvalues and corresponding eigenvectors of a matrix or at least a subset of the spectrum of this matrix. To achieve this, the Arnoldi algorithm does the Gram-Schmidt orthogonalization on the Krylov subset basis, which is generated during the process. Let us first briefly recall the matrix form of the Arnoldi algorithm as seen in [16]:

$$AV_k = V_k H_k + h_{k+1,k} v_{k+1} e_k^T, \tag{2.4}$$

where $A \in \mathbb{R}^{n\times n}$, $H \in \mathbb{R}^{k\times k}$ is upper Hessenberg and the columns $v \in \mathbb{R}^n$ of the matrix $V \in \mathbb{R}^{n\times k}$ form the orthonormal base of the $k-th$ Krylov subspace $K_k(A, v) = \text{span}(v, Av, ..., A^{k-1}v)$. Now if we apply Arnoldi algorithm to a symmetric matrix $A \in \mathbb{R}^{n\times n}$ and multiply the relation 2.4 by matrix V_k^* from the left we can see that the matrix H_k is symmetric as well:

$$H_k = V_k^* A V_k = V_k^* A^* V_k = (V_k^* A V_k)^* = H_k^*.$$

Since the matrix H_k is upper Hessenberg and symmetric, it must be tridiagonal. This leads us to the matrix form of the Lanczos algorithm:

$$AV_k = V_k T_k + \beta_{k+1} v_{k+1} e_k^T. \tag{2.5}$$

Algorithm 2.1: Lanczos algorithm

```
  input : SPD matrix $A \in \mathbb{R}^{n\times n}$, starting vector $v_1 \in \mathbb{R}^n$, such that
          $\|v_1\|_2 = 1$
  output: Matrices $V_i$ and $T_i$ and vector $v_{i+1}$ satisfying 2.5
1 $u_1 = Av_1$;
2 for $i = 1$ to $nmax$ do
3     $\alpha_i = v_i^T u_i$;
4     $w_i = u_i - \alpha_i v_i$;
5     $\beta_{i+1} = \|w_i\|_2$;
6     $v_{i+1} = w_i / \beta_{i+1}$;
7     $u_{i+1} = Av_{i+1} - \beta_{i+1} v_i$;
8 end
```

The finite precision behaviour of classical Lanczos method is best described in a series of papers done by Paige (e.g. [11]). This analysis provided a methodology to Carson who gave the first analysis of the CA-Lanczos. In her work Carson extends various results for classical KSMs to the CA-KSMs including bounds on the maximal attainable accuracy, convergence, and loss of orthogonality. One of the important results is that these bounds can be rewritten in the same way as the bounds for the classical KSMs counterparts multiplied by a factor Γ, where Γ depends on the condition number of the Krylov base. This indicates that if the factor Γ is controlled and is *approximately one* then we can expect the same rate of the loss of orthogonality as for classical KSMs.

In the following section we will concisely derive the CA-KSM, for thorough introduction we refer to [2].

2.1.1 CA-Lanczos

From Algorithm 2.1 using mathematical induction on lines (6) and (7) it follows that after $(ks+1)$-th iteration (where $k \in \mathbb{N}$ and $0 \neq s \in \mathbb{N}$) for $j \in \{1, ..., s+1\}$ the vectors satisfy:

$$\begin{aligned} v_{sk+j} &\in \mathcal{K}_s(A, v_{sk+1}) + \mathcal{K}_s(A, u_{sk+1}), \\ v_{sk+j} &\in \mathcal{K}_{s+1}(A, v_{sk+1}) + \mathcal{K}_{s+1}(A, u_{sk+1}). \end{aligned} \tag{2.6}$$

For $k > 0$ let $\mathcal{V}_k$ and $\mathcal{U}_k$ be matrices $n \times (s+1)$ whose columns form bases of $\mathcal{K}_s(A, v_{sk+1})$ and $\mathcal{K}_s(A, u_{sk+1})$ respectively. This allows us to define a so-called *basis matrix*

$$\mathcal{Y}_k = [\mathcal{V}_k, \mathcal{U}_k]. \tag{2.7}$$

Then we can write the vectors v_{sk+j} and v_{sk+j} by their coordinates in the $\mathcal{Y}_k$:

$$\begin{aligned} v_{sk+j} &= \mathcal{Y}_k v'_{k,j}, \\ u_{sk+j} &= \mathcal{Y}_k u'_{k,j}, \end{aligned} \tag{2.8}$$

where vectors $v'_{k,j}$ and $u'_{k,j}$ are the coordinate vectors. Using the basis matrix $\mathcal{Y}_k$, we can define the Gram matrix

$$G_k = \mathcal{Y}_k^T \mathcal{Y}_k,$$

and then use this matrix and the coordinate vectors to rewrite the dot products from Algorithm 2.1, lines (3) and (5):

$$\begin{aligned} \alpha_{sk+j} &= v'^T_{k,j} G_k u'_{k,j} \\ \beta_{sk+j} &= ((w')^T_{k,j} G_k w'_{k,j})^{1/2}, \end{aligned} \tag{2.9}$$

where $w'_{k,j}$ is coordinate vector for the vector w_{sk+j} i.e. $w_{sk+j} = \mathcal{Y}_k w'_{k,j}$.

As discussed in the introduction of Chapter 2, the bases are generated using polynomial recurrences, which can be represented as a matrix $\mathcal{B}_k$ and the recurrence is then written as in (2.2). Then we can write:

$$Av_{sk+j+1} = \mathcal{Y}_k \mathcal{B}_k v'_{k,j+1}. \tag{2.10}$$

To perform one inner iteration, we have to generate the basis matrix $\mathcal{Y}_k$ and then compute the Gram matrix G_k. Thus, iterations $sk+2$ through $sk+s+1$

have the same communication costs as one iteration of the classical method. The CA-Lanczos algorithm presented in this thesis in Algorithm 2.2 is as given by [2].

Algorithm 2.2: CA-Lanczos algorithm

```
input  : SPD matrix $A \in \mathbb{R}^{n\times n}$, starting vector $v_1 \in \mathbb{R}^n$, such that
         $\|v_1\|_2 = 1$
output: Matrices $V_{sk+s}$ and $T_{sk+s}$ and vector $v_{sk+s+1}$ satisfying 2.5
1  $u_1 = Av_1$;
2  for $k = 1$ to $nmax$ do
3      Compute $\mathcal{Y}_k$ witch change of basis $\mathcal{B}_k$ according to 2.10;
4      Compute $G_k = \mathcal{Y}_k^T \mathcal{Y}_k$;
5      $v'_{k,1} = e_1$;
6      if $k$=$0$ then
7          $u'_{0,1} = \mathcal{B}_0 e_1$
8      end
9      else
10         $u'_{k,1} = e_{s+2}$
11     end
12     for $j = 1$ to $s$ do
13         $\alpha_{sk+j} = (v'_{k,j})^T G_k u'_{k,j}$;
14         $w'_{k,j} = u'_{k,j} - \alpha_{sk+j} v'_{k,j}$;
15         $\beta_{sk+j+1} = ((w')^T_{k,j} G_k w'_{k,j})^{1/2}$;
16         $v'_{k,j+1} = w'_{k,j} / \beta_{sk+j+1}$;
17         $v_{sk+j+1} = \mathcal{Y}_k v'_{k,j+1}$;
18         $u'_{k,j+1} = \mathcal{B}_k v'_{k,j+1} - \beta_{sk+j+1} v'_{k,j}$;
19         $u_{sk+j+1} = \mathcal{Y}_k u'_{k,j+1}$
20     end
21 end
```

The main result of the CA-Lanczos analysis in finite precision in [2] is the Theorem 1 (Chapter 5,page 86). The quantity $\bar{\Gamma}_k$ from this theorem which is a measure of the conditioning of the Krylov bases generated at the beginning of each outer loop. This suggests that if one could handle $\bar{\Gamma}_k$ to make it fulfil $\bar{\Gamma}_k = \mathcal{O}(s)$ then the bounds given by this theorem would be similar to those given earlier by Paige. This means we can expect similar rate of loss of orthogonality. We could use a different definition for $\bar{\Gamma}_k$ in order to obtain tighter bounds, however, it is to be pointed out that the value of these bounds is in the insight they provide rather in their tightness.

2.2 Conjugate gradient and Lanczos algorithm

It is well known that there is a strong relation between the conjugate gradients and the Lanczos algorithm, for more details see [16]. Applied to a symmetric positive-definite matrix A and vector $v = b - Ax_0 = r_0$, where x_0 is an initial approximation of the solution of the system $Ax = b$, the Lanczos algorithm will produce an orthonormal basis $\{v_1, \ldots, v_k\}$ of the k-th Krylov subspace $\mathcal{K}_k = (A, r_0)$ stored in matrix $V_k = [v_1, \ldots, v_k]$. Because the approximate solution x_k lies within the

$x_0 + \mathcal{K}_k = (A, r_0)$, we can write:

$$x_k = x_0 + V_k y_k$$

for some y_k. The residual r_k is orthogonal to $\mathcal{K} = (A, r_0) = span\{v_1, \dots, v_k\}$, therefore:

$$0 = V_k^T r_k = V_k^T (r_0 - AV_k y_k) = \|r_0\| e_1 - V_k^T AV_k y_k = \|r_0\| e_1 - T_k y_k,$$

where T_k is a tridiagonal Jacobi matrix storing the orthogonalization and normalization coefficients. Hence, we can express the conjugate gradient k-th approximation as:

$$x_k = x_0 + V_k y_k, \quad \|r_0\| e_1 = T_k y_k.$$

From the CG algorithm and Lanczos algorithm, we can see that both algorithms produce orthonormal base vectors u_j such that

$$w_{j+1} \in \mathcal{K}_{j+1} = (A, r_0), \quad \text{and} \mathcal{K}_j = (A, r_0).$$

for $j = 1, \dots k$. From the conjugate gradient algorithm, it immediately follows that $w_j = r_j$ and from comparison of the both algorithms we get $v_{j+1} = (-1)^j \frac{r_j}{\|r_j\|}$. There is a following relation between the coefficients from both algorithms:

$$\beta_{j+1} = \frac{\sqrt{\delta_j}}{\gamma_{j-1}}, \quad \alpha_j \frac{1}{\gamma_{j-1}} + \frac{\delta_{j-1}}{\gamma_{j-2}},$$

where $\gamma_k = \alpha_k$ and $\delta_k = \beta_k$ are renamed coefficients from Algorithm 1.1. There is, therefore, a connection between these two methods.

3. Techniques for improvement

There is a number of methods for improving stability and convergence for KSMs in finite precision. Carson in [2] has shown that many of these techniques including residual replacement, selective reorthogonalization or look-ahead method, can be used along with CA-KSMs as well. These methods are somewhat problem-dependent, however with some modifications and applied to suitable problem these methods improve the CA-KSMs numerical properties while preserving the communication savings.

The KSMs are rarely used in their unpreconditioned form. The CA-KSMs are designed to reduce the costs of every single iteration of the method. On the other hand, the purpose of preconditioning is to reduce the total amount of iterations that are needed to complete the computation. These two approaches are complementary in a way and appear to open potential for a powerful method. However, finding parallelism in the techniques for preconditioning and making them usable with the CA-KSMs can be challenging.

3.1 Preconditioning

Consider system (1) $Ax = b$, where the matrix A has a certain property (i.e. for the conjugate gradient method, the matrix A is symmetric positive-definite and say sparse). The goal is to modify the original system to obtain a new system $M^{-1}Ax = M^{-1}b$, where the new system should fulfill the same properties, where the matrix M is called the preconditioner. The original system can be written as

$$(M^{-1}AM^{-T})(M^Tx) = M^{-1}b, \tag{3.1}$$

define $\hat{A} = M^{-1}AM^{-T}, \hat{x} = M^Tx$ and $\hat{(b)} = M^{-1}b$. Now we can apply the KSMs to solve the new system $\hat{A}\hat{x} = \hat{b}$ and obtain the solution of the system (1) with the relation $x = M^{-T}\hat{x}$. If we apply this to the conjugate gradient method, where the matrix A is SPD, then the matrix $\hat{A}$ of the preconditioned system is also SPD and there is following relation between the errors:

$$\|\hat{x} - \hat{x}_k\|^2_{\hat{A}} = \|x - x_k\|^2_A. \tag{3.2}$$

This implies that if the method applied to the preconditioned system converges fast, the approximate solutions $x_k = M^{-T}\hat{x}_k$ of the original system converge to the exact solution fast as well. Note that the method applied to the preconditioned system solves a different task as it minimizes the error using the $K_k = (\hat{A}, \hat{b})$ measured in the $\hat{A}$-norm rather than using the A-norm and $K_k = (A, b)$. This raises the question of how to select a good preconditioner. The preconditioner should meet the following requirements:

(i) $MA \approx I$,

(ii) application of M^{-1} to a vector is inexpensive; the systems $My = z$ and $M^Ty = z$ are solvable in $O(n)$ operations,

(iii) if the matrix A is sparse, the preconditioner should be also sparse.

Selecting a good preconditioner is usually problem-dependent and it is not always clear what a good preconditioner would be. Selection $M \approx A^{-1}$ might seem like a good idea and will improve convergence rate, but even if the matrix A is sparse, the inverse is usually dense, which breaks locality.

3.1.1 Incomplete Cholesky and ILU factorization

One of the most important preconditioning techniques is to use a modification of the Cholesky factorization. For an SPD matrix A there exists a lower triangular matrix L with nonzero positive diagonal such that $A = LL^T$. However the sparsity of the matrix A is not necessarily transferred into the matrix L, thus the costs of computation with the matrix L are higher. Therefore, the idea behind the incomplete Cholesky factorization is to form a matrix C that is lower triangular with the sparsity property of the matrix A and is similar to the matrix L in the following sense: $A \approx CC^T$. This C can be then used for preconditioning the CG method. One can use a Cholesky factorization variant known under the marking IC(0), which computes Cholesky factorization and lays $c_{i,j} = 0$ if $a_{i,j} = 0$.

Using incomplete Cholesky factorization or incomplete LU factorization (ILU) together with a CA-KSMs have been less useful than with the classical KSMs. This is because computations involved in these methods are not generally suitable for straightforward parallelization and lead to the search for improvements. To use ILU along with parallel methods, it is necessary to search for possible ways of parallelism in the computations included in ILU process. There has been some progress towards parallelism in work of Hysom and Pothen [10], where authors assume special properties of the original problem and use these in developing a scalable parallel algorithm for computing the ILU. In [3] authors introduce their completely new algorithm for computing incomplete LU decomposition on parallel systems. Authors mention that the parallelism in their algorithm is fine-grained, which means that the individual entries of the factorization can be computed in parallel rather than the rows (compare to older methods) and is therefore suitable for modern architectures with large number of processors (or graphics processing units) that have been available only recently. The ILU is in a way related to the Gaussian elimination and preceding methods for computing ILU were equivalent to this. The new ILU is based on the observation that:

$$(LU)_{i,j} = (a)_{i,j}, \text{ for } (i,j) \in S, \tag{3.3}$$

where S is a set of matrix coordinates where the non-zeros are allowed. The condition 3.3 can be written as:

$$\sum_{k=1}^{min(i,j)} l_{i,k} u_{k,j} = a_{i,j}, \text{ for } (i,j) \in S. \tag{3.4}$$

Which leads to solving $|S|$ nonlinear equations for $|S|$ unknowns. There are methods for solving these systems in parallel and in addition, we do not need a very precise solution to obtain a good ILU preconditioner.

There has been also a lot of work done by van der Vorst (e.g. ([17])) on finding parallelism in the preconditioning e.g. re-ordering (which can be further distinguished into the wavefront ordering, the multi-color ordering, multi-wavefront ordering) or domain decomposition. However, the suitability of each method is very problem-dependent and new preconditioning are still yet to be found.

Conclusion

In this thesis, we gave a state-of-the-art on the communication avoiding Krylov subspace methods. In chapter 1, multiple ways to overleap the communication in the conjugate gradient method were presented. Namely the three-term variant including the derivation of the three-term recurrence as given by Stiefel in [14] and the s-step CG for which we included proof of convergence. The three-term recurrence itself and other additional recurrences, however, introduce new sources of instabilities into the methods. We described how the s-step method is connected with modern approaches to avoiding communication in KSM. In chapter 2 we summarized and put into context with Chapter 1 the work of Carson, who gave the first finite precision analysis of the CA-Lanczos algorithm. The interesting outcome of this analysis is that when multiplied by a factor Γ, the bounds given by Paige for the classical Lanczos algorithm hold for the CA-Lanczos as well. This factor Γ depends on the condition number of the Krylov bases computed in each step of the method. This fact can be spotted in the relations between them and their classical counterparts for other CA-KSMs. In chapter 3 we introduced the possibilities of improving CA-KSMs. We discussed the topic of preconditioning as it is common for KSMs to be used with some kind of preconditioner. The main problem of using preconditioners with CA-KSMs is the need to compute the preconditioner on parallel systems as well. We presented one method of incomplete LU factorization by Chow and Patel in [3] that is based on fine-grained parallelism. There are as well some older techniques given by van der Vorst ([17]) or Hysom and Pothen ([10]) that are more or less suitable for various problems. The importance of the future work on preconditioning is also indicated by the recent benchmarks for parallel computers given in [7], where the performance is measured for conjugate gradient method preconditioned using a symmetric Gauss-Seidel preconditioner. Nevertheless, the problem of preconditioning CA-KSMs remains an open question and it is still possible that new techniques will be discovered.

Bibliography

[1] E. C. Carson, Miroslav Rozložnık, Zdeněk Strakoš, Petr Tichý, and Miroslav Tůma. "The Numerical Stability Analysis of Pipelined Conjugate Gradient Methods: Historical Context and Methodology". In: *SIAM J. Sci. Comput.* 40.5 (2018), A3549–A3580. DOI: 10.1137/16M1103361. URL: http://www.karlin.mff.cuni.cz/~mirektuma/ps/crstt16.pdf.

[2] Erin Carson. "Communication-Avoiding Krylov Subspace Methods in Theory and Practice". PhD thesis. EECS Department, University of California, Berkeley, Aug. 2015. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-179.html.

[3] Edmond Chow and Aftab Patel. "Fine-Grained Parallel Incomplete LU Factorization". In: *SIAM J. Sci. Comput.* (2015).

[4] A. T. Chronopoulos and Charles William Gear. "On the efficient implementation of preconditioned s-step conjugate gradient methods on multiprocessors with memory hierarchy". In: *Parallel Computing* 11.1 (Jan. 1989), pp. 37–53. ISSN: 0167-8191. DOI: 10.1016/0167-8191(89)90062-8.

[5] A Chronopoulos and C Gear. "S-step iterative methods for symmetric linear systems". In: *Journal of Computational and Applied Mathematics* 25 (Feb. 1989), pp. 153–168. DOI: 10.1016/0377-0427(89)90045-9.

[6] Ed D'Azevedo, Victor Eijkhout, and Chris Romine. "Lapack Working Note 56 Conjugate Gradient Algorithms with Reduced Synchronization Overhead on Distributed Memory Multiprocessors". In: (Jan. 2000).

[7] Jack J. Dongarra, Michael A. Heroux, and Piotr Luszczek. "HPCG Benchmark : a New Metric for Ranking High Performance Computing Systems ". In: 2015.

[8] P. Ghysels and W. Vanroose. "Hiding Global Synchronization Latency in the Preconditioned Conjugate Gradient Algorithm". In: *Parallel Comput.* 40.7 (July 2014), pp. 224–238. ISSN: 0167-8191. DOI: 10.1016/j.parco.2013.06.001. URL: http://dx.doi.org/10.1016/j.parco.2013.06.001.

[9] M. R. Hestenes and E. Stiefel. "Methods of conjugate gradients for solving linear systems". In: *Journal of research of the National Bureau of Standards* 49 (1952), pp. 409–436.

[10] David Hysom and Alex Pothen. "A Scalable Parallel Algorithm for Incomplete Factor Preconditioning". In: *SIAM J. SCI. COMPUT* 22 (2000), pp. 2194–2215.

[11] Christopher C. Paige. "Error Analysis of the Lanczos Algorithm for Tridiagonalizing a Symmetric Matrix". In: 1976.

[12] H. Rutishauser. "Theory of Gradient Methods". In: *Refined Iterative Methods for Computation of the Solution and the Eigenvalues of Self-Adjoint Boundary Value Problems.* Basel: Birkhäuser Basel, 1959, pp. 24–49. ISBN: 978-3-0348-7224-9. DOI: 10.1007/978-3-0348-7224-9_2. URL: https://doi.org/10.1007/978-3-0348-7224-9_2.

[13] Youcef Saad. “Practical Use of Polynomial Preconditionings for the Conjugate Gradient Method”. In: *Siam Journal on Scientific and Statistical Computing* 6 (Oct. 1985). DOI: `10.1137/0906059`.

[14] E. Stiefel. “Relaxationsmethoden bester Strategie zur Lösung linearer Gleichungssysteme”. In: *Commentarii Mathematici Helvetici* 29.1 (Dec. 1955), pp. 157–179. ISSN: 1420-8946. DOI: `10.1007/BF02564277`. URL: `https://doi.org/10.1007/BF02564277`.

[15] G. Szegö. *Orthogonal Polynomials.* American Math. Soc: Colloquium publ v. 23. American Mathematical Society, 1939. ISBN: 9780821810231.

[16] Jurjen D. Tebbens, Iveta Hnětynková, Martin Plešinger, Zdeněk Strakoš, and Petr Tichý. *Analýza metod pro maticové výpočty – Základní metody.* 1. vydani. Praha: Matfyzpress, 2012. ISBN: 978-80-7378-201-6.

[17] Henk Van der Vorst. “Parallel Iterative Solution Methods for Linear Systems arising from Discretized PDE’s”. In: *Special Course on Parallel Computing in CFD* (June 1995).

List of Algorithms