Abstract

Classical iterative methods require data movement in each iteration, yielding a bottleneck in parallel processing. The cyclic formulations, requiring lagged parameters from previous iterations, can theoretically reduce the global synchronizations. The pioneering work was done three decades ago by Barzilai and Borwein, then leading to a framework called gradient methods with retards. On the other hand, the s-step formulations act as a well-known communication-avoiding strategy through s simultaneously formed directions. We can break the data dependencies of inner products and perform other operations one after another within s iterations. This method was originally proposed for minimizing the A-norm error over a s-dimensional plane, and then it has efficiently evolved over the years for other projection methods such as the minimal residual method and several Krylov subspace methods. In this paper we are interested in the communication-avoiding formulations of the gradient-type methods. Sequential and parallel algorithms are given and numerical experiments are conducted to illustrate the efficiency.

Keywords: gradient methods with retards, cyclic gradient methods, s-step methods, minimizing communication, parallel computing, message passing interface.

1 Introduction

The computational cost is a function of both arithmetic operations and communication. On modern computational environments, communication costs are much higher than the computation costs, and this trend is likely to accelerate in future systems. One way to remedy such problem consists in breaking the data dependency, leading probably to a chaotic iterative process if the control of data transmission is fully abandoned, which is usually called asynchronous iterations (see, e.g, [1]). In this case, large over-

lap of arithmetic operations in different iterative cycles may exist among several cores or machines, resulting in a reduction of waiting time. On the other hand, if one still manages explicitly the data transmission but modifies the algorithm in order to minimize communication, better parallel performance may be obtained. We call this a communication-avoiding (CA) algorithm. Several works of CA have been done over ten years for the Krylov and the factorization methods (see, e.g., [2, 3]). The original ideas such as the s-dimensional gradient method, however, can date back to the 1950s (see, e.g, [4]). For a recent overview of communication avoidance, see [5] and the references therein.

Krylov subspace methods are often the choice in practice which have finite termination property in exact arithmetic (see, e.g., [6]). For example, the conjugate gradient (CG) method [7] gives the optimal result for the symmetric positive definite (SPD) systems. The s-step variant was proposed in [8] in order to reduce the frequency of data exchange between slow and fast devices. Nevertheless, when low precision is required, CG sometimes performs not good [9]. Any derivation such as a small non-quadratic term in the quadratic function or nonsymmetric error in the Hessian matrix can seriously degrade its performance [10]. In such cases, the basic gradient methods may also become competitive [11]. There exist several promising formulations since Barzilai and Borwein proposed the lagged gradient method [12]. We introduce in this paper some variants of gradient methods for the sake of avoiding communication, providing sequential algorithms and parallel implementations based on the message passing interface (MPI), which is a commonly used protocol for the communication among several computational nodes.

In the next section, we introduce the preliminaries of gradient methods with retards. The cyclic iterative strategy and the s-step formulations are illustrated. In Section 3, we provide the parallel implementation of our methods, including the discussions of performance. Section 4 gives some numerical experiments for the new methods in the distributed environment with an MPI-based programming library. Finally, we draw some conclusions in Section 5.

2 Gradient and s-step gradient methods

2.1 Preliminaries

Consider the system of linear equations

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

where A is an N-dimensional SPD matrix and b is an N-dimensional vector. We assume that all coefficients in system (1) are real. To solve this equation, we consider the gradient iterative method

$$x^{(n+1)} = x^{(n)} - \alpha^{(n)} g^{(n)} \tag{2}$$

started from a given initial vector $x^{(0)}$. Here, n is the number of the current step. One executes in each step an advance along the gradient direction $g^{(n)} = Ax^{(n)} - b$, for which the distance equals $\alpha^{(n)}$. With basic substitution, the gradient vector can be updated by

$$g^{(n+1)} = g^{(n)} - \alpha^{(n)} A g^{(n)}. \tag{3}$$

Thus, the sequence $\{x^{(n)}\}$ is generated by a specific method characterized by the formula of $\alpha^{(n)}$, which converges to the exact solution x^*. In finite-precision environment, the sequence should approximate to x^* within a given threshold ε in finite number of iterations.

Steepest descent (SD) plays a major role in the history of iterative methods, which can be written in the following form

$$\alpha_{\mathrm{SD}}^{(n)} = \frac{(g^{(n)})^{\intercal} g^{(n)}}{(g^{(n)})^{\intercal} A g^{(n)}}, \tag{4}$$

where $u^{\intercal}$ denotes the transpose of vector u. It minimizes the A-norm error $\left\|e^{(n)}\right\|_A = \left\|x^* - x^{(n)}\right\|_A$ in each iteration, thus showing some optimal property in the beginning. It is widely known that, however, the SD method is inefficient since it reduces the quadratic function

$$f^{(n)} = \frac{1}{2}(x^{(n)})^{\intercal} A x^{(n)} - b^{\intercal} x^{(n)} \tag{5}$$

along a zigzag path and converges asymptotically into a two-dimensional subspace. Barzilai and Borwein [12] proposed a similar form with a one-step delay compared to the SD method

$$\alpha_{\mathrm{BB}}^{(n)} = \frac{(g^{(n-1)})^{\intercal} g^{(n-1)}}{(g^{(n-1)})^{\intercal} A g^{(n-1)}}, \tag{6}$$

which was inspired by the quasi-Newton method. Numerical experiments show that BB is much more efficient than SD, while few theoretical results support such assertion due to the difficulties in analyzing the nonmonotone convergence behavior. The global convergence has been proved in [13] by an ingenious induction.

2.2 Cyclic steepest descent

We observe that both SD and BB require one matrix-vector multiplication and two dot product operations in each iteration. In [9], a general convergence framework called gradient methods with retards (GMR) was proposed

$$\alpha_{\mathrm{GMR}}^{(n)} = \frac{(g^{(\tau(n))})^{\intercal} A^{\rho(n)} g^{(\tau(n))}}{(g^{(\tau(n))})^{\intercal} A^{\rho(n)+1} g^{(\tau(n))}}, \tag{7}$$

with

$$\tau(n) \in \{\bar{n},\, \bar{n}+1,\, \ldots,\, n-1,\, n\}, \quad \rho(n) \in \{q_1,\, \ldots,\, q_m\}, \quad q_j \geq 0,$$

where m is a given positive integer and $\bar{n} = \max\{0,\ n - m\}$. Here, $\tau(n)$ acts as a delayed iteration number. There exist m possible matrix powers fulfilling the number of retards. The convergence analysis has been conducted in the same manner as that was done in [13] for the BB method. The significance can be explained by the fact that all sequences of solution vectors generated by the gradient methods which satisfy Equation (7) converge to x^*. For example, we could always choose the minimal SD steplength in the past m iterations to execute the current update of solution and gradient vectors. Then the sequence $\{x^{(n)}\}$ converges theoretically regardless of the stability issue due to the oscillating behavior.

From model (7) we can trivially derive the cyclic steepest descent (CSD) method

$$\alpha_{\text{CSD}}^{(n)} = \alpha_{\text{SD}}^{(\tau(n))}, \quad \tau(j) = \max\left\{j \leq n : j \bmod d = 0\right\}, \tag{8}$$

where d is a positive integer. We remark that this method has been mentioned in several publications (see, e.g., [9, 14]). The alternate step gradient method [14] is indeed a special case with $d = 2$. Here we recall this method since it reduces $2(d-1)$ dot product operations in every d iterations, resulting as well in a reduction of communication when parallel computing is required. A relevant method called cyclic Barzilai-Borwein (CBB) can be derived as hinted by model (7). In this text, however, we shall not investigate this variant further since CBB and CSD share similar properties in both sequential and parallel views.

In practice, iterative methods are exploited when solving large sparse linear systems, for which the condition number may be large. Such system may be solved many orders of magnitude faster by CG than by a simple gradient method. On the other hand, for some moderate problems, BB and CSD are sometimes competitive and more straightforward to be implemented. SD is seldom used as an entire solver for sparse matrix, except that its spectral properties may contribute to the estimation of second order information, see [15] for more details.

2.3 s-step steepest descent

The approximate solution in the next iteration can be expressed as

$$x^{(n+1)} = x^{(n)} - P(A)g^{(n)}.$$

Choosing $P(A) = A^{-1}$ leads indeed to the exact solution x^*. According to the Cayley-Hamilton theorem, we know that A^{-1} could be presented as a matrix polynomial. Assume that A is a polynomial of the form

$$P(A) = \alpha_1^{(n)} I + \alpha_2^{(n)} A + \cdots + \alpha_s^{(n)} A^{s-1},$$

where I is an N-dimensional identity matrix and s is a positive integer. We consider the s-dimensional plane

$$L_s^{(n)} = \left\{ x^{(n)} - \sum_{j=1}^{s} \alpha_j^{(n)} A^{j-1} g^{(n)} : \alpha_j^{(n)} \in \mathbb{R} \right\}. \tag{9}$$

Here, as before $g^{(n)} = Ax^{(n)} - b$. Notice that Equation (9) depicts indeed the search spaces of gradient methods when choosing $s = 1$.

Now we try to search an optimal solution in the current iteration by choosing s steplengths in the plane. If minimizing the A-norm error, then the next solution vector $x^{(n+1)}$ is defined to be a unique point in $L_s^{(n)}$. The relevant algorithm called s-step steepest descent (s-SD) can be illustrated in Algorithm 1.

Algorithm 1 s-step SD method

1: set $x^{(0)}$, compute $g^{(0)} = Ax^{(0)} - b$
2: **for** $n = 0, 1, \ldots$ **do**
3: select $\alpha_1^{(n)}, \ldots, \alpha_s^{(n)}$ to minimize $\left\|x_* - x^{(n+1)}\right\|_A^2$ over $L_s^{(n)}$
4: $x^{(n+1)} = x^{(n)} - \alpha_1^{(n)} g^{(n)} - \cdots - \alpha_s^{(n)} A^{s-1} g^{(n)}$
5: $g^{(n+1)} = g^{(n)} - \alpha_1^{(n)} A g^{(n)} - \cdots - \alpha_s^{(n)} A^s g^{(n)}$
6: **end for**

In order to solve s scalars over $L_s^{(n)}$, one finds that $g^{(n+1)}$ must be orthogonal to $\{g^{(n)}, Ag^{(n)}, \ldots, A^{s-1}g^{(n)}\}$, and thus we get the following system of linear equations

$$\begin{array}{ccccccc}
\alpha_1^{(n)}(g^{(n)})^\intercal A g^{(n)} & + & \ldots & + & \alpha_s^{(n)}(g^{(n)})^\intercal A^s g^{(n)} & = & (g^{(n)})^\intercal g^{(n)}, \\
\alpha_1^{(n)}(Ag^{(n)})^\intercal A g^{(n)} & + & \ldots & + & \alpha_s^{(n)}(Ag^{(n)})^\intercal A^s g^{(n)} & = & (Ag^{(n)})^\intercal g^{(n)}, \\
\vdots & & & & \vdots & & \vdots \\
\alpha_1^{(n)}(A^{s-1}g^{(n)})^\intercal A g^{(n)} & + & \ldots & + & \alpha_s^{(n)}(A^{s-1}g^{(n)})^\intercal A^s g^{(n)} & = & (A^{s-1}g^{(n)})^\intercal g^{(n)}.
\end{array}$$

Let $\omega_j^{(n)} = (g^{(n)})^\intercal A^j g^{(n)}$, it follows that

$$\begin{pmatrix}
\omega_1^{(n)} & \omega_2^{(n)} & \cdots & \omega_s^{(n)} \\
\omega_2^{(n)} & \omega_3^{(n)} & \cdots & \omega_{s+1}^{(n)} \\
\vdots & \vdots & & \vdots \\
\omega_s^{(n)} & \omega_{s+1}^{(n)} & \cdots & \omega_{2s-1}^{(n)}
\end{pmatrix}
\begin{pmatrix}
\alpha_1^{(n)} \\ \alpha_2^{(n)} \\ \vdots \\ \alpha_s^{(n)}
\end{pmatrix} =
\begin{pmatrix}
\omega_0^{(n)} \\ \omega_1^{(n)} \\ \vdots \\ \omega_{s-1}^{(n)}
\end{pmatrix}, \tag{10}$$

which can be denoted by $\Omega^{(n)}\alpha^{(n)} = \varphi^{(n)}$. Similar representations have appeared in [4] and [8].

Equation (10) is a positive definite Hankel matrix. It is known that such kind of matrices is ill-conditioned for which the condition number has a lower bound $3 \cdot 2^{s-6}$ (see, e.g., [16]). In our case, besides the numerical stability concerns, s shall be smaller than 10 since large inner linear system requires also huge computation costs in solving the parameters. The computational work consists of $4sN$ multiplications, $4sN$ additions, s matrix-vector operations and the cost of solving the Hankel system. For example, the Cholesky factorization requires approximately $(1/3)s^3 + (1/2)s^2$

arithmetic operations, or we can say, in flops. For large N, such cost is acceptable in view of a relatively small s.

We can take the above idea of s-SD one step further and minimize 2-norm residual instead of A-norm error that leads to the s-step minimal gradient (s-MG) method. Such alternative method has similar form and property, which shall not be discussed in the following text.

2.4 Cyclic s-step formulation

Here we propose a new method that combines the s-SD process with nonmonotone strategy. In practice, s-SD performs similar to SD and thus shows potential convergence curve at the beginning, however, stagnates before arriving at the termination threshold. Both SD and s-SD tend to zigzag in two directions (see [4]). A convergence behavior called "decreasing together" has been observed in the nonmonotone gradient methods (see [17]), for which the steplength is indeed a delay term in view of the current iteration number n. Additionally, iterative methods with retards may contribute significantly to the saving of communication costs in a parallel environment.

Algorithm 2 Cyclic s-step SD method

set $x^{(0)}$, d, compute $g^{(0)} = Ax^{(0)} - b$
for $n = 0, 1, \ldots$ **do**
  if $n \bmod d = 0$ **then**
    select $\alpha_1^{(n)}, \ldots, \alpha_s^{(n)}$ to minimize $\left\|x_* - x^{(n+1)}\right\|_A^2$ over $L_s^{(n)}$
    compute $\hat{\alpha}^{(n)}$
    $x^{(n+1)} = x^{(n)} - \alpha_1^{(n)} g^{(n)} - \cdots - \alpha_s^{(n)} A^{s-1} g^{(n)}$
    $g^{(n+1)} = g^{(n)} - \alpha_1^{(n)} A g^{(n)} - \cdots - \alpha_s^{(n)} A^s g^{(n)}$
    $r = 0$
  else
    $x^{(n+1)} = x^{(n)} - \hat{\alpha}^{(n-r)} g^{(n)}$
    $g^{(n+1)} = g^{(n)} - \hat{\alpha}^{(n-r)} A g^{(n)}$
  end if
  $r \coloneqq r + 1$
end for

The algorithm called cyclic s-step steepest descent (Cs-SD) is illustrated in Algorithm 2, from which we observe that there exist a lagged parameter d and a new variable $\hat{\alpha}^{(n)}$ Thus, the s-SD process shall be executed once per cycle, where a cycle can be defined as d iterations. On the other hand, $\hat{\alpha}^{(n)}$ could be selected within the framework (7) such that both numerator and denominator have been obtained in the past. By Equation (10), we could deduce that there exist $2s$ parameters $\omega_j^{(n)}$ being

computed when $n \bmod d = 0$, and then every

$$\hat{\alpha}^{(n+i)} = \frac{\omega_j^{(n)}}{\omega_{j+1}^{(n)}}$$

satisfies the GMR framework with $1 \leq i < d$ and $0 \leq j < 2s-1$. En average, this method requires in each iteration $2(d+1)sN/d$ multiplications, $2(d+1)sN/d$ additions, s matrix-vector products and $\mathcal{O}(s^3/d)$ operations to solve the positive definite Hankel system.

3 Parallel implementation

The paralelization of a sequential algorithm is sometimes a trade-off between the computation costs and communication costs, and we need to further consider the storage management. For example, when executing the dot product operation, since the matrices are often stored equally over a distributed memory architecture, one needs to wait for two synchronization points, which constitutes a bottleneck in the algorithm.

Let $x^{(0)} = 0$ and let x_i denote the vector data saved in ith processor where $i = 1, \ldots, p$. Assume that $p \geq s$ and A is partitioned into the banded submatrices corresponding to p. Our implementation of parallel s-SD is illustrated in Algorithm 3.

Algorithm 3 Parallel s-step SD

given s, set $g_i^{(0)} = -b_i$
`Allgather` $(g_i^{(0)}, g^{(0)})$
for $n = 0, 1, \ldots$ **do**
    $\omega_0^{(n)} = (g_i^{(n)})^\intercal g_i^{(n)}$
    set $u = g^{(n)}$
    for $j = 1, \ldots, s$ **do**
        $u_i = A_i u$
        set $q_j = u_i$
        `Allgather` (u_i, u)
        $\omega_j^{(n)} = (g_i^{(n)})^\intercal u_i$
    end for
    for $j = s+1, \ldots, 2s-1$ **do**
        $\omega_j^{(n)} = q_s^\intercal q_{j-s}$
    end for
    `Allreduce` $(\omega^{(n)}, 2s,$ `SUM`$)$
    solve $\Omega^{(n)}\alpha^{(n)} = \varphi^{(n)}$
    $x_i^{(n+1)} = x_i^{(n)} - \alpha_1^{(n)} g_i^{(n)} - \alpha_2^{(n)} q_1 - \cdots - \alpha_s^{(n)} q_{s-1}$
    $g_i^{(n+1)} = g_i^{(n)} - \alpha_1^{(n)} q_1 - \alpha_2^{(n)} q_2 - \cdots - \alpha_s^{(n)} q_s$
    `Allgather` $(g_i^{(n+1)}, g^{(n+1)})$
end for

Recall that `Allgather` and `Allreduce` are collective communication routines defined in the MPI specification, which involves participation of all processors. We note that the dimension of matrix may not be a power of 2, and thus the function `Allgatherv`, a variant for gathering data with displacement of indices, may be more appropriate in terms of the implementation. In Algorithm 3 the parameters are simplified. For `Allgather`, the first parameter is the sending buffer, while the second one is the receiving buffer. We collect the result of matrix-vector product whenever the data is ready. On the other hand, we compute the summation of p distributed $\omega_j^{(n)}$ vector with a length of $2s$ only once per iteration. The solution of Hankel system may be obtained by some parallel algorithms. Since we suppose that s is a small integer, in our case, the vector $\alpha^{(n)}$ is solve independently by p processors using Cholesky factorization.

Similarly, the parallel Cs-SD method is sketched in Algorithm 4. The lagged parameter d is introduced for the sake of avoiding communication and exploiting the nonmonotone properties. As a result, the convergence curve is oscillating but faster and requires less synchronization points.

Algorithm 4 Parallel cyclic s-step SD

given s and d, set $g_i^{(0)} = -b_i$
`Allgather` ($g_i^{(0)}$, $g^{(0)}$)
for $n = 0, 1, \ldots$ **do**
    if $n \bmod d = 0$ **then**
        same as Algorithm 3
        compute $\hat{\alpha}^{(n)}$
        $r = 0$
    else
        $q = A_i g^{(n)}$
        $x_i^{(n+1)} = x_i^{(n)} - \hat{\alpha}^{(n-r)} g_i^{(n)}$
        $g_i^{(n+1)} = g_i^{(n)} - \hat{\alpha}^{(n-r)} q$
    end if
    `Allgather` ($g_i^{(n+1)}$, $g^{(n+1)}$)
    $r = r + 1$
end for

The final collection of gradient vector is necessary for the detection of convergence, however, the dot product operations for the next $\omega_0^{(n)}$ could be conducted in the meantime by using the `Allreduce` routine. Additionally, the computation of $\hat{\alpha}$ requires normally the coefficients of the Hankel matrix, leading to a saving of computational efforts. Another solution consists in choosing the $\hat{\alpha}$ dynamically during the postponing period. Thus, the algorithm might be further improved in practice.

We notice that the MG method, as mentioned in Section 2.3, could be formalized with s-step (see, e.g., [8]) and lagged term d in a similar manner.

4 Numerical experiments

In this section we present some experimental results by using the matrices from the University of Florida Sparse Matrix Collection [18]. All right-hand vectors are generated by $b = Ax^*$ where $x^* = (1, \ldots, 1)^\intercal$. Assume that $x^{(0)} = 0$ and $\varepsilon = 10^{-6}$. The stopping criterion is fixed with $\left\|g^{(n)}\right\| < 10^{-6} \left\|g^{(0)}\right\|$. All tests are conducted in the C++ environment with some mathematical [19] and MPI-based parallel [20, 21] programming libraries.

We first illustrated in Figure 1 the numerical behavior of Cs-SD compared with SD and s-SD. We choose here $s = 3$ and $d = 2$. The coefficient matrix is obtained by a

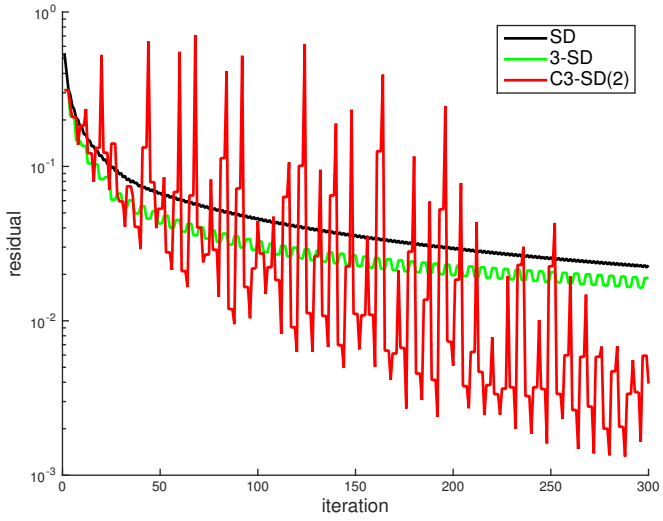

Figure 1: Comparison of SD, s-SD and Cs-SD.

structural problem with $N = 7102$ and the condition number $\kappa = 1.576 \times 10^4$. It is clear that the C$s$-SD shows nonmonotone curve oscillating throughout the iterations, while SD and s-SD, which minimizes the A-norm error, present relatively smooth curves. Notice that we have adjusted the number of iterations such that a s-step results in s iterations, which favors the comparison of different types of methods. In view of the convergence rate, however, iterative methods with retards are generally much more efficient than the locally optimal methods, although we could not yet give a Q-linear bound due to the nonstandard choice of step.

The next examples concern the parallel performance with two matrices constructed from some structural problems. The first one is collected from an ill-conditioned problem with $N = 10848$ and $\kappa = 9.967 \times 10^9$, while the second one is much moderate in view of the condition number but has relatively large dimension $N = 141347$. We list in Table 1 the average results among 10 tests for the first matrix. We choose here $s = 5$ and $d = 2$. From the results we can see that our implementation of s-SD is far lest efficient than the others. On the other hand, CSD seems very promising in

method	# iterations	time (s)	residual
SD	1000	0.770	5.528×10^{-5}
BB	1000	0.646	2.626×10^{-6}
CSD(2)	1000	0.609	1.470×10^{-4}
5-SD	1000	3.010	5.330×10^{-5}
C5-SD(2)	1000	0.616	2.256×10^{-5}

Table 1: Average results among 10 tests for different methods with the dimension of matrix $N = 10848$, which is a very ill-conditioned problem. Here, we use 16 processors and let the number of iterations equal 1000.

view of the execution times. Note that residual is not a fair indicator for performance since nonmonotone methods usually converge in a nonstandard way. Additionally, our new method Cs-SD is also competitive with other iterative methods with retards and much more efficient than s-SD. For the second matrix, results are illustrated in Table 2. Similarly, we notice that the convergence speed of s-SD in unacceptable.

method	# iterations	time (s)	residual
SD	1000	5.732	4.459×10^{-4}
BB	567	2.354	8.331×10^{-7}
CSD(3)	385	1.358	3.427×10^{-7}
5-SD	1000	18.316	3.260×10^{-6}
C5-SD(3)	649	2.575	7.804×10^{-7}

Table 2: Average results among 10 tests for different methods with the dimension of matrix $N = 141347$. We use 16 processors and fix the stopping criterion with $\left\|g^{(n)}\right\| < 10^{-6} \left\|g^{(0)}\right\|$.

Among all these methods, CSD is the most efficient, while BB and Cs-SD have practically the same performance. SD and s-SD do not converge within 1000 iterations. These observations are contrary to our expectations, as we speculated from theoretical analysis that s-step methods are communication-efficient. Thus, we conclude that our implementation, i.e., the sequential Cholesky factorization process for Hankel matrix deteriorates our formulations, which acts as a bottleneck in the algorithm. Recall that the implementations of inner system solver are the same for both s-SD and Cs-SD. The bottleneck here seems to be remedied by the cyclic strategy. On the other hand, the postponing strategy is successfully applied to the gradient and s-step gradient methods according to the test results of CSD and Cs-SD. We note that the number of iterations is less significant for the nonmonotone methods due to their chaotic convergence character.

5 Conclusion

In this paper we present the parallel formulations of gradient methods, as well as a new s-step method with retards. There exist several other issues that should be further tackled. For example, it is known that our implementations of s-step methods are unstable due to the choice of basis (see, e.g., [3]), which is reflected in the ill-conditioning of the Hankel matrix. Another problem consists in the nonmonotone behavior of lagged methods that may result in large error accumulations. In a word, we have not addressed the stability problem throughout the text. From the numerical results, we notice that the cyclic exploitation of lagged parameters leads to promising algorithms in parallel environment. On the other hand, the basic s-SD method is far less efficient than others, which is contrary to our expectations. We draw the conclusion that our implementation for solving the Hankel system based on the sequential Cholesky solver is a bottleneck in the s-SD method. The gain from communication could not compensate the cost of factorization. Thus, apart from the stability problem, the code segment of inner system solver should also be modified.

Acknowledgements

This work was supported by the French national programme LEFE/INSU and the project ADOM (Méthodes de décomposition de domaine asynchrones) of the French National Research Agency (ANR).

References

[1] A. Frommer, D.B. Szyld, "On Asynchronous Iterations", *J. Comput. Appl. Math.*, 123(1-2): 201–216, 2000.

[2] J.W. Demmel, L. Grigori, M. Hoemmen, J. Langou, "Communication-optimal Parallel and Sequential QR and LU Factorizations", *SIAM J. Sci. Comput.*, 34 (1): A206–A239, 2012.

[3] E.C. Carson, N. Knight, J.W. Demmel, "Avoiding Communication in Nonsymmetric Lanczos-based Krylov Subspace Methods", *SIAM J. Sci. Comput.*, 35(5): S42–S61, 2013.

[4] G.E. Forsythe, "On the Asymptotic Directions of the s-Dimensional Optimum Gradient Method", *Numer. Math.*, 11(1): 57–76, 1968.

[5] M. Hoemmen, *Communication-Avoiding Krylov Subspace Methods*, PhD thesis, UC Berkeley, 2010.

[6] Y. Saad, *Iterative Methods for Sparse Linear Systems*, SIAM, Philadelphia, 2nd edition, 2003.

[7] M.R. Hestenes, E. Stiefel, "Methods of Conjugate Gradients for Solving Linear Systems", *J. Res. Natl. Bur. Stand.*, 49(6): 409–436, 1952.

[8] A.T. Chronopoulos, C.W. Gear, "s-Step Iterative Methods for Symmetric Linear Systems", *J. Comput. Appl. Math.*, 25(2): 153–168, 1989.

[9] A. Friedlander, J.M. Martínez, B. Molina, M. Raydan, "Gradient Method with Retards and Generalizations", *SIAM J. Numer. Anal.*, 36(1): 275–289, 1999.

[10] Y.H. Dai, R. Fletcher, "On the Asymptotic Behaviour of Some New Gradient Methods", *Math. Program.*, 103(3): 541–559, 2005.

[11] R. Fletcher, "On the Barzilai-Borwein Method", in L. Qi, K. Teo, X. Yang (Editors), *Optimization and Control with Applications*, pages 235–256. Springer US, Boston, MA, 2005.

[12] J. Barzilai, J.M. Borwein, "Two-Point Step Size Gradient Methods", *IMA J. Numer. Anal.*, 8(1): 141–148, 1988.

[13] M. Raydan, "On the Barzilai and Borwein Choice of Steplength for the Gradient Method", *IMA J. Numer. Anal.*, 13(3): 321–326, 1993.

[14] Y.H. Dai, "Alternate Step Gradient Method", *Optimization*, 52(4-5): 395–415, 2003.

[15] R. De Asmundis, D. di Serafino, W.W. Hager, G. Toraldo, H. Zhang, "An Efficient Gradient Method Using the Yuan Steplength", *Comput. Optim. Appl.*, 59 (3): 541–563, 2014.

[16] E.E. Tyrtyshnikov, "How Bad are Hankel Matrices?", *Numer. Math.*, 67(2): 261–269, 1994.

[17] Y.H. Dai, Y.X. Yuan, "Analysis of Monotone Gradient Methods", *J. Ind. Manag. Optim.*, 1(2): 181–192, 2005.

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

[19] F. Magoulès, A.K. Cheik Ahamed, "Alinea: An Advanced Linear Algebra Library for Massively Parallel Computations on Graphics Processing Units", *Int. J. High Perform. Comput. Appl.*, 29(3): 284–310, 2015.

[20] F. Magoulès, G. Gbikpi-Benissan, "JACK: An Asynchronous Communication Kernel Library for Iterative Algorithms", *J. Supercomput.*, 73(8): 3468–3487, 2017.

[21] F. Magoulès, G. Gbikpi-Benissan, "JACK2: An MPI-based Communication Library with Non-blocking Synchronization for Asynchronous Iterations", *Adv. Eng. Softw.*, 119: 116–133, 2018.