Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm

P. Ghysels
W. Vanroose

Report 12.2012.1, December 2012

This work is funded by Intel and by the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT).

Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm

P. Ghysels[*†] W. Vanroose[*]

December 5, 2012

Abstract

Scalability of Krylov subspace methods suffers from costly global synchronization steps that arise in dot-products and norm calculations on parallel machines. In this work, a modified Conjugate Gradient (CG) method is presented that removes the costly global synchronization steps from the standard CG algorithm by only performing a single non-blocking reduction per iteration. This global communication phase can be overlapped by the matrix-vector product, which typically only requires local communication. The resulting algorithm will be referred to as *pipelined CG*. An alternative pipelined method, mathematically equivalent to the Conjugate Residual method that makes different trade-offs with regard to scalability and serial runtime is also considered. These methods are compared to a recently proposed asynchronous CG algorithm proposed by B. Gropp. Extensive numerical experiments demonstrate the numerical stability of the methods. Moreover, it is shown that hiding the global synchronization step improves scalability on distributed memory machines using the message passing paradigm and leads to significant speedups compared to standard CG.

1 Introduction

Many high-performance computing applications rely on Krylov subspace methods for their linear algebra. These Krylov methods exploit the sparsity of the matrices that typically appear in scientific applications simulating a problem modeled by a partial differential equation (PDE). The linear systems to be solved are derived by a discretization technique such as finite-differences, finite-volumes or finite-elements in which neighboring variables are related through a stencil. This results in matrices with only a few non-zero elements per row.

The building blocks for Krylov subspace methods are the sparse matrix-vector product (SpMV), vector-vector additions and dot-products. Each building block has a different communication pattern on distributed memory machines, resulting in different scaling properties.

The SpMV often requires only local communication. The matrix derived from a PDE can, possibly after a permutation of rows and columns, be mapped to the underlying machine architecture in such a way that applying a matrix-vector product only requires communication between neighboring nodes, i.e. nodes that are separated by a small number of hops. In the state-of-the-art literature, many examples can be found of stencil based codes that scale nearly optimal to very large parallel machines. In this work, the focus is on sparse and well structured matrices or matrix-free linear operators such as stencil applications.

Vector operations such as an AXPY ($y \leftarrow \alpha x + y$) can be calculated locally and do not require communication between nodes. However, on a multicore shared memory system even a simple AXPY operation typically does not scale well over the cores due to memory bandwidth congestion. But this on-chip communication bottleneck is not the focus of the current article.

[*]University of Antwerp - Department of Mathematics and Computer Science, Middelheimlaan 1, B-2020 Antwerp - Belgium (pieter.ghysels@ua.ac.be)

[†]Intel ExaScience Lab - Kapeldreef 75, B-3001 Leuven - Belgium

In contrast, we focus on the global communication. A dot-product of two vectors v and w involves such global communication and requires participation of all processes. Since the dot-product is a single scalar value $\alpha = \sum_{i=1}^{n} v_i^T w_i$, this operation needs the result of each local dot-product and involves a synchronization of all the processes as well. Even if the global communication is extremely fast, for instance when a special reduction network is available, the explicit synchronization of the involved processes makes a dot-product a costly operation. For large systems the cost of the global reduction grows as $\mathcal{O}(\log(P_n))$, the height of the reduction tree, where P_n is the number of nodes. This cost model ignores variability due to OS jitter, core speed variability or load imbalance that start to play an important role for larger systems [22].

Research in reducing the number of global reductions in Krylov methods goes back to the first implementations of Krylov methods on parallel computers [3]. Variations of the Conjugate Gradient (CG) method with only a single global synchronization point have been presented by Saad and Meurant [29, 28], by D'Azevedo, Romine [12] and Eijkhout [11] as well as by Chronopoulos and Gear [9]. Also, a CG variation exists that is based on two three-term recurrences instead of three two-term recurrences and that only requires one reduction [31]. Yang et al [42, 40, 39, 41] propose the so called *improved* versions of the QMR, BiCGStab, BiCG and CGS algorithms respectively, which reduce the number of synchronizations to just one per iteration for these methods. The total number of global reductions is reduced even further by s-step Krylov methods [9, 24, 2, 26, 8, 10, 25, 34, 6], where s matrix-vector products are combined and the resulting Krylov basis is orthogonalized simultaneously with a single global reduction, leading to a communication reduction of a factor s. Apart from reducing global communication, these methods are also aimed at reducing communication with the slow memory. However, with increasing s, the stability of the s-step Krylov basis deteriorates. In [27], hierarchical or nested Krylov methods are used that reduce the number of global reductions on the whole machine, but freely allow global communication on a smaller subset of nodes, where communication is cheaper.

Instead of, or in addition to trying to reduce global synchronizations, several authors have come up with ways to overlap expensive communication phases with computations. In [15] it is suggested to overlap the dot-products in CG with the update of the solution vector or with application of a factored preconditioner. In [13], block Gram-Schmidt is used in the Generalized Minimal Residual (GMRES) method where communication for one block can be overlapped with computation on a different block. In the Arnoldi algorithm for eigenvalue computations, Hernandez et al [19] overlap global communication for reorthogonalization of the Krylov basis with the computations for orthogonalization of the next basis vector. Recently, we have proposed a pipelined GMRES solver [16] where the global reductions are overlapped with the calculation of multiple matrix-vector products. The results of the dot-products are only used with a delay of a few iterations in the algorithm. In [1] a model is developed for the performance of the pipelined GMRES algorithm on large clusters. An asynchronous version of CG has been proposed recently by Gropp [18] where one reduction can be overlapped with the matrix-vector product and the other with the preconditioner. Overlapping global communication with local work has recently become easier and more attractive due to the inclusion of non-blocking collectives in the MPI-3 standard [4, 23].

The *aim* of this work is to hide the latency of global communication due to dot-products in the preconditioned Conjugate Gradient method. The trade-off between improved scalability and extra floating point operations should result in speedups on medium to large parallel machines without significantly sacrificing numerical robustness.

Our main *contributions* are the development of a preconditioned pipelined CG method that only has a single non-blocking reduction per iteration. This non-blocking reduction can be overlapped with the matrix-vector product and with application of the preconditioner. Also, a preconditioned pipelined Conjugate Residual (CR) method is presented with one non-blocking reduction that can be overlapped with the matrix-vector product. We show that both methods have much improved scalability and runtime compared to standard CG.

The paper is outlined as follows. Section 2 reviews the standard preconditioned CG method. A modified CG method due to Chronopoulos and Gear that will be used to derive pipelined CG is discussed in Section 2.2. Section 3 presents algorithms that can overlap the global reduction with the matrix-vector product, with in Section 3.1 first the unpreconditioned pipelined CG method.

Next, in Section 3.2 the preconditioned pipelined CG method is derived. In Section 3.3 a preconditioned pipelined CR method is derived from pipelined CG. Section 3.4 gives the asynchronous CG algorithm due to Gropp. We report on extensive numerical tests that show the stability of the pipelined CG and CR methods in Section 4. Also in Section 4, we hint at the use of a residual replacement strategy to improve the maximum attainable accuracy of the pipelined methods. Finally, the results of benchmarks on a parallel distributed memory computer are presented in Section 5.

2 The Conjugate Gradient algorithm

First, the standard preconditioned conjugate gradient algorithm is reviewed together with the definition of the Krylov subspace. Then a variation to standard CG due to Chronopoulos and Gear is presented that only requires a single global reduction per iteration.

2.1 Preconditioned Conjugate Gradients

The mother of all Krylov methods is conjugate gradients, dating back to a paper from 1952 by Hestenes and Stiefel [20]. Algorithm 1 shows the preconditioned CG iteration [31], which iteratively solves $M^{-1}Ax = M^{-1}b$ where both A and M are symmetric and positive definite square matrices of size $N \times N$.

Algorithm 1 Preconditioned CG

```
1: $r_0 := b - Ax_0;\quad u_0 := M^{-1}r_0;\quad p_0 := u_0$
2: for $i = 0, \dots$ do
3:     $s := Ap_i$
4:     $\alpha := (r_i, u_i) / (s, p_i)$
5:     $x_{i+1} := x_i + \alpha p_i$
6:     $r_{i+1} := r_i - \alpha s$
7:     $u_{i+1} := M^{-1}r_{i+1}$
8:     $\beta := (r_{i+1}, u_{i+1}) / (r_i, u_i)$
9:     $p_{i+1} := u_{i+1} + \beta p_i$
10: end for
```

The residual vector of the original system is $r_i = b - Ax_i$, while $u_i = M^{-1}r_i$ is the residual of the preconditioned system and p_i is called the search direction. If the exact solution is $\hat{x} = A^{-1}b$, then the error is defined as $e_i = \hat{x} - x_i$. All subsequent approximations x_i lie in a so-called Krylov subspace $\mathcal{K}_i(M^{-1}A, M^{-1}r_0)$, which is defined as

$$\mathcal{K}_i(M^{-1}A, M^{-1}r_0) = \text{span}\{u_0, M^{-1}Au_0, \dots, (M^{-1}A)^{i-1}u_0\}\,. \tag{1}$$

It is well known that the CG iteration generates a sequence of iterates $x_i \in x_0 + \mathcal{K}_i(M^{-1}A, u_0)$, with the property that at step i, $\|e_i\|_A = \sqrt{e_i^T A e_i}$ is minimized.

In terms of communication, the important steps in Algorithm 1 are: application of the sparse matrix-vector product (SpMV) Ap_i in line 3, application of the preconditioner $M^{-1}r_{i+1}$ in line 7, and the two dot-products (s, p_i) and (r_{i+1}, u_{i+1}) in lines 4 and 8. All other steps are vector updates that do not require communication between nodes. We assume that the matrix-vector product and application of the preconditioner only require communication among neighboring nodes, which can be implemented in a scalable way. The two dot-products, causing two global synchronization points per iteration become the bottleneck with increasing parallelism.

Note that even when A and M are both symmetric and positive definite, the left or right preconditioned systems $M^{-1}Ax = M^{-1}b$ and $AM^{-1}u = b$ with $x = M^{-1}u$ respectively are no longer symmetric in general. When the preconditioner is available in the form $M = LL^T$, then one way to preserve symmetry is to use split preconditioning $L^{-1}AL^{-T}u = L^{-1}b$. In [15], a split preconditioned CG method is presented where the global reductions can be overlapped with local work. The communication for one reduction can be overlapped with application of L^{-T} and, since the iteration does not depend on x_i, the update for x_i can be postponed by one iteration where it can be overlapped with the second reduction.

Another way to preserve symmetry is based on the observation that $M^{-1}A$ is self-adjoint with respect to the M inner-product $(x, y)_M = (x, My) = (Mx, y)$:

$$\left(M^{-1}Ax, y\right)_M = (Ax, y) = (x, Ay) = (x, M(M^{-1}A)y) = (x, M^{-1}Ay)_M \,. \tag{2}$$

Replacing the usual Euclidean inner-product in CG with this M inner-product leads to Algorithm 1. Similarly, for right preconditioning, the M^{-1} inner-product also leads to Algorithm 1. Furthermore, the iterates generated by split preconditioned CG are identical to those of Algorithm 1 [31].

2.2 Chronopoulos/Gear CG

Several authors have suggested alternatives to CG to reduce the number of global synchronization points to just one. One such method was presented by Saad [29], and was later improved for stability by Meurant [28]. This latter method performs an additional dot-product compared to the standard CG implementation. But, these three dot-products, compared to two for standard CG, can be combined in a single reduction. Similarly, Chronopoulos and Gear [9] presented a CG variation with a single global synchronization point that requires one additional AXPY compared to CG. A slight variation to this method was published by D'Azevedo, Romine [12] and Eijkhout [11]. Also, the three term recurrence version of CG [31] can be implemented with a single global reduction. All these CG variations have slightly different properties in terms of memory requirements, total number of flops and stability. The remainder of this work builds on the method by Chronopoulos and Gear [9] as shown in Algorithm 2.

Algorithm 2 Preconditioned Chronopoulos/Gear CG

```
1: $r_0 := b - Ax_0$;   $u_0 := M^{-1}r_0$;   $w_0 := Au_0$
2: $\alpha_0 := (r_0, u_0) \,/\, (w_0, u_0)$;   $\beta_0 := 0$;   $\gamma_0 := (r_0, u_0)$
3: for $i = 0, \dots$ do
4:     $p_i := u_i + \beta_i p_{i-1}$
5:     $s_i := w_i + \beta_i s_{i-1}$
6:     $x_{i+1} := x_i + \alpha_i p_i$
7:     $r_{i+1} := r_i - \alpha_i s_i$
8:     $u_{i+1} := M^{-1}r_{i+1}$
9:     $w_{i+1} := Au_{i+1}$
10:    $\gamma_{i+1} := (r_{i+1}, u_{i+1})$
11:    $\delta := (w_{i+1}, u_{i+1})$
12:    $\beta_{i+1} := \gamma_{i+1}/\gamma_i$
13:    $\alpha_{i+1} := \gamma_{i+1}/\,(\delta - \beta_{i+1}\gamma_{i+1}/\alpha_i)$
14: end for
```

Compared to standard CG, Algorithm 2 performs an additional vector update for the vector $s_i = Ap_i$, found by multiplying the recurrence for p_i by A

$$Ap_i = Au_i + \beta_i Ap_{i-1} \,, \qquad s_i = Au_i + \beta_i s_{i-1} \,, \tag{3}$$

or since in the Chronopoulos/Gear method $w_i = Au_i$, this becomes $s_i = w_i + \beta_i s_{i-1}$. Algorithm 2 was derived from its unpreconditioned version by replacing the Euclidean inner-product by the M inner-product and is mathematically equivalent to standard preconditioned CG, Algorithm 1.

The communication phase for both dot-products from Algorithm 2, lines 10 and 11, can be combined in a single global reduction. The update for x_i can be postponed and used to overlap the global reduction. However, even for small parallel machines the runtime of a single vector update will not be enough to fully cover the latency of the global communication.

An optimized implementation of Algorithm 2 could exploit the fact that only a single memory sweep is required, allowing for more efficient use of available memory bandwidth compared to standard CG, which needs at least two. This was already suggested by Chronopoulos and Gear in their original paper [9], where they also generalize Algorithm 2 to an s-step CG method that only needs one memory sweep per s steps. Algorithm 2 corresponds to the $s = 1$ case.

3 Hiding global communication

In this section, modified and reordered versions of Algorithm 2 are presented where the global synchronization can be overlapped by the sparse matrix-vector product. Section 3.1 starts with the unpreconditioned version of this pipelined CG algorithm for simplicity. Section 3.2 adds preconditioning to it and Section 3.3 uses a different inner-product to preserve symmetry for the preconditioned case, which leads to a pipelined Conjugate Residual (pipe-CR) method. Section 3.4 discusses an asynchronous CG method recently proposed by Gropp.

3.1 Pipelined Conjugate Gradients

Recall from Section 2 that $r_i = b - Ax_i$ and $s_i = Ap_i$ with p_i the search direction. Lets first consider the unpreconditioned iteration, i.e., $u_i = M^{-1}r_i = r_i$, then $w_i = Au_i = Ar_i$.

Instead of computing $w_{i+1} = Ar_{i+1}$ directly using the SpMV, it also satisfies the recurrence relation

$$Ar_{i+1} = Ar_i - \alpha_i As_i \,, \qquad w_{i+1} = w_i - \alpha_i As_i \,. \tag{4}$$

This however requires $z_i = As_i \equiv A^2 p_i$, which can be computed via the SpMV as As_i, or for which also a recurrence relation can be found

$$As_i = Aw_i + \beta_i As_{i-1} \,, \qquad z_i = Aw_i + \beta_i z_{i-1} \,. \tag{5}$$

This depends on $q_i = Aw_i \equiv A^2 r_i$, which can be computed from the SpMV. Starting from Algorithm 2, adding the recurrences for w_{i+1} (4) and z_i (5), replacing the SpMV by $q_i = Aw_i$ and reordering the steps leads to Algorithm 3.

Algorithm 3 Pipelined Chronopoulos/Gear CG

```
1:  $r_0 := b - Ax_0$;   $w_0 := Ar_0$
2:  for $i = 0, \ldots$ do
3:      $\gamma_i := (r_i, r_i)$
4:      $\delta := (w_i, r_i)$
5:      $q_i := Aw_i$
6:      if $i > 0$ then
7:          $\beta_i := \gamma_i/\gamma_{i-1}$;   $\alpha_i := \gamma_i/(\delta - \beta_i\gamma_i/\alpha_{i-1})$
8:      else
9:          $\beta_i := 0$;   $\alpha_i := \gamma_i/\delta$
10:     end if
11:     $z_i := q_i + \beta_i z_{i-1}$
12:     $s_i := w_i + \beta_i s_{i-1}$
13:     $p_i := r_i + \beta_i p_{i-1}$
14:     $x_{i+1} := x_i + \alpha_i p_i$
15:     $r_{i+1} := r_i - \alpha_i s_i$
16:     $w_{i+1} := w_i - \alpha_i z_i$
17: end for
```

Mathematically, Algorithm 3 is still equivalent with standard CG. However, as in the method by Chronopoulos/Gear, the two dot-products in lines 3 and 4 can be combined in a single reduction.

Furthermore, since the result of this reduction is only needed in line 7, this global reduction can be overlapped with the SpMV. We shall refer to Algorithm 3 as unpreconditioned pipelined CG.

In finite precision arithmetic, pipelined CG will behave differently than standard CG since rounding errors are propagated differently. Numerical stability of this algorithm, and its preconditioned variants presented in the next paragraphs, will be studied using a wide range of matrices in Section 4.

3.2 Preconditioned pipelined CG

We shall consider the left preconditioned system $M^{-1}Ax = M^{-1}b$ with both M and A symmetric and positive definite. As for standard CG, pipelined CG, Algorithm 3 cannot be applied to the preconditioned system without modification since $M^{-1}A$ is in general not symmetric positive definite. For preconditioned pipelined CG the same strategy as for standard preconditioned CG will be followed: apply Algorithm 3 to the preconditioned system and replace the classic Euclidean inner-product by the M inner-product $(\cdot,\cdot)_M$, since $M^{-1}A$ is self-adjoint with respect to the M inner-product. The dot-products from pipelined CG (lines 3 and 4) now use the residual of the preconditioned system, $u_i = M^{-1}r_i$, and the M inner-product instead of the Euclidean inner-product

$$\gamma_i = (u_i, u_i)_M = (Mu_i, u_i) = (r_i, u_i) \tag{6}$$
$$\delta_i = \left(M^{-1}Au_i, u_i\right)_M = (Au_i, u_i)\ . \tag{7}$$

Now, let $w_i = Au_i$, $s_i = Ap_i$ and also introduce $q_i = M^{-1}s_i$. A recurrence relation for the preconditioned residual u_i is found by multiplying the recurrence for the original residual on the left by M^{-1}

$$M^{-1}r_{i+1} = M^{-1}r_i - \alpha_i M^{-1}s_i\,, \qquad u_{i+1} = u_i - \alpha_i q_i\,, \tag{8}$$

with a similar recurrence for $q_i = M^{-1}s_i$

$$M^{-1}s_i = M^{-1}w_i + \beta_i M^{-1}s_i\,, \qquad q_i = M^{-1}w_i + \beta_i q_{i-1}\,. \tag{9}$$

If defined that $m_i = M^{-1}w_i \equiv M^{-1}Au_i \equiv M^{-1}AM^{-1}r_i$, then since $w_{i+1} = Au_{i+1}$,

$$Au_{i+1} = Au_i - \alpha_i Aq_i\,, \qquad w_{i+1} = w_i - \alpha_i Aq_i\,. \tag{10}$$

Contrary to the unpreconditioned case, now define $z_i = Aq_i$ and the final recurrence relation for z_i becomes

$$Aq_i = AM^{-1}w_i + \beta_i Aq_i\,, \qquad z_i = Am_i + \beta_i z_{i-1}\,, \tag{11}$$

and let $n_i = Am_i \equiv AM^{-1}w_i$.

Combining the updates for r_{i+1}, x_{i+1}, s_i and p_i, and the equations (6)-(11) with the matrix-vector product $n_i = Am_i$ and application of the preconditioner $m_i = M^{-1}w_i$, directly leads to preconditioned pipelined CG, Algorithm 4. In this algorithm, the reduction for the dot-products (lines 3 and 4) can be overlapped with application of both the preconditioner (line 5) and the matrix-vector product (line 6). However, the number of AXPYs has increased from just three in standard CG or four in Chronopoulos/Gear CG to eight in Algorithm 4. Again, these operations can be fused together (also including the dot-products from the start of the next iteration) such that only a single memory sweep is necessary.

The preconditioned pipelined CG method, Algorithm 4, is based on the M inner-product just like standard preconditioned CG, Algorithm 1, and hence we know that the two methods are mathematically equivalent. They both minimize the A norm of the error, $\|e_k\|_A$ over the Krylov space $\mathcal{K}_k(M^{-1}A, M^{-1}r_0)$.

Algorithm 4 Preconditioned pipelined CG

```
1: $r_0 := b - Ax_0$;   $u_0 := M^{-1}r_0$;   $w_0 := Au_0$
2: for $i = 0, \ldots$ do
3:     $\gamma_i := (r_i, u_i)$
4:     $\delta := (w_i, u_i)$
5:     $m_i := M^{-1}w_i$
6:     $n_i := Am_i$
7:     if $i > 0$ then
8:         $\beta_i := \gamma_i/\gamma_{i-1}$;   $\alpha_i := \gamma_i/\left(\delta - \beta_i\gamma_i/\alpha_{i-1}\right)$
9:     else
10:        $\beta_i := 0$;   $\alpha_i := \gamma_i/\delta$
11:    end if
12:    $z_i := n_i + \beta_i z_{i-1}$
13:    $q_i := m_i + \beta_i q_{i-1}$
14:    $s_i := w_i + \beta_i s_{i-1}$
15:    $p_i := u_i + \beta_i p_{i-1}$
16:    $x_{i+1} := x_i + \alpha_i p_i$
17:    $r_{i+1} := r_i - \alpha_i s_i$
18:    $u_{i+1} := u_i - \alpha_i q_i$
19:    $w_{i+1} := w_i - \alpha_i z_i$
20: end for
```

3.3 Preconditioned pipelined Conjugate Residuals

For Algorithm 4, the Euclidean inner-product in Algorithm 3 was replaced by the M inner-product. However, observe that $M^{-1}A$ is also self-adjoint with respect to the A inner-product

$$\left(M^{-1}Ax, y\right)_A = (AM^{-1}Ax, y) = (M^{-1}Ax, Ay) = (M^{-1}Ax, y)_A \, . \tag{12}$$

Using this inner-product instead leads to

$$\gamma_i = \left(u_i, u_i\right)_A = \left(Au_i, u_i\right) = \left(w_i, u_i\right) \tag{13}$$

$$\delta_i = \left(M^{-1}Au_i, u_i\right)_A = \left(M^{-1}Au_i, Au_i\right) = \left(m_i, w_i\right) \, . \tag{14}$$

Using equations (13) and (14) instead of (6) and (7) leads to Algorithm 5. However, since Algorithm 5 is based on a different inner-product, it is no longer equivalent with standard CG but as will become clear it can be seen as a pipelined version of the Conjugate Residual (CR) method [31].

Since (14) depends on $m_i = M^{-1}w_i$, the preconditioner now has to be applied before (line 3) the dot-products can be started. This means that the global reduction can only be overlapped with the matrix-vector product (line 6). However, the iteration does not depend on the original unpreconditioned residual r_i and on $s_i = Ap_i$ anymore. These two recurrences can safely be dropped, saving some floating point operations and memory. When the recurrence for r_i is dropped, a stopping criterion can still be based on the preconditioned residual u_i.

Another optimization is possible: note that, apart from lines 14 and 15, the iterates in Algorithm 5 do not depend on x or p. Hence the recurrences for x and p can be postponed one iteration and used in the overlap with the reduction. However, for the p recurrence, this requires storage for one extra vector. For long reductions, this would shorten the critical path of the method by two vector updates.

Theorem 3.1. *Algorithm 5, when applied to a linear system $M^{-1}Ax = M^{-1}b$, with exact solution $\hat{x}$ and with both M and A symmetric positive definite, minimizes $\|e_k\|_{AM^{-1}A}$, where $e_k = \hat{x} - x_k$ is the error and $x_k \in x_0 + \mathcal{K}_k(M^{-1}A, u_0)$. Without preconditioner, i.e., $M^{-1} = I$, the iteration minimizes the 2-norm of the residual $\|r_k\|_2 = \|e_k\|_{A^2}$.*

Algorithm 5 Preconditioned pipelined Conjugate Residuals

```
1: $r_0 := b - Ax_0$;   $u_0 := M^{-1}r_0$;   $w_0 := Au_0$
2: for $i = 0, \ldots$ do
3:     $m_i := M^{-1}w_i$
4:     $\gamma_i := (w_i, u_i)$
5:     $\delta := (m_i, w_i)$
6:     $n_i := Am_i$
7:     if $i > 0$ then
8:         $\beta_i := \gamma_i/\gamma_{i-1}$;   $\alpha_i := \gamma_i/\left(\delta - \beta_i\gamma_i/\alpha_{i-1}\right)$
9:     else
10:        $\beta_i := 0$;   $\alpha_i := \gamma_i/\delta$
11:    end if
12:    $z_i := n_i + \beta_i z_{i-1}$
13:    $q_i := m_i + \beta_i q_{i-1}$
14:    $p_i := u_i + \beta_i p_{i-1}$
15:    $x_{i+1} := x_i + \alpha_i p_i$
16:    $u_{i+1} := u_i - \alpha_i q_i$
17:    $w_{i+1} := w_i - \alpha_i z_i$
18: end for
```

Proof. We first show the following orthogonality properties

$$(u_k, u_j)_A = u_k^T A u_j = u_k^T w_j = 0\,, \quad j < k\,, \tag{15}$$

$$(p_k, M^{-1}Ap_j)_A = p_k^T A M^{-1} A p_j = p_k^T A q_j = 0\,, \quad j < k\,, \tag{16}$$

by induction on k. First check that indeed for $k = 1$

$$u_1^T A u_0 = \left(u_0 - \alpha_0 q_0\right)^T w_0 = u_0^T w_0 - \alpha_0 \left(m_0 + \beta_0\right)^T w_0 \tag{17}$$

is zero if $\beta_0 = 0$ and

$$\alpha_0 = \frac{(u_0, w_0)}{(m_0, w_0)} = \frac{\gamma_0}{\delta_0}\,, \tag{18}$$

which corresponds to the definitions of β_0 and α_0 in Algorithm 5. Likewise, for (16),

$$p_1^T A q_0 = p_1^T A m_0 = p_1^T n_0 = p_1^T z_0 = \left(u_1 + \beta_1 p_0\right)^T \left(\frac{w_0 - w_1}{\alpha_0}\right) \tag{19}$$

$$= \frac{1}{\alpha_0}\left(u_1^T w_0 - u_1^T w_1 + \beta_1 p_0^T w_0 - \beta_1 p_0^T w_1\right) \tag{20}$$

$$= \frac{1}{\alpha_0}\left(-\gamma_1 + \beta_1 u_0^T w_0 - \beta_1 u_0^T w_1\right) = \frac{1}{\alpha_0}\left(-\gamma_1 + \beta_1 \gamma_0\right) \tag{21}$$

is zero when $\beta_1 = \gamma_1/\gamma_0$. For $k > 1$, multiply the recurrence relation for u_k^T on the right with Au_j to get

$$u_k^T A u_j = u_{k-1}^T A u_j - \alpha_{k-1} q_{k-1}^T A u_j\,. \tag{22}$$

For $j < k - 1$, both terms in the right-hand side are zero by induction and because the u and p

vectors span the same Krylov space. For $j = k-1$ the right-hand side of (22) is zero if

$$\begin{aligned}\alpha_{k-1} &= \frac{u_{k-1}^T A u_{k-1}}{q_{k-1}^T A u_{k-1}} = \frac{(u_{k-1}, w_{k-1})}{(q_{k-1}, w_{k-1})} = \frac{\gamma_{k-1}}{(m_{k-1} + \beta_{k-1} q_{k-2}, w_{k-1})} \\ &= \frac{\gamma_{k-1}}{(m_{k-1}, w_{k-1}) + \beta_{k-1}(q_{k-2}, w_{k-1})} = \frac{\gamma_{k-1}}{\delta_{k-1} + \beta_{k-1}\left(\frac{u_{k-2}-u_{k-1}}{\alpha_{k-2}}, w_{k-1}\right)} \\ &= \frac{\gamma_{k-1}}{\delta_{k-1} + \frac{\beta_{k-1}}{\alpha_{k-2}}\left[(u_{k-2}, w_{k-1}) - (u_{k-1}, w_{k-1})\right]} = \frac{\gamma_{k-1}}{\delta_{k-1} - \frac{\beta_{k-1}}{\alpha_{k-2}}(u_{k-1}, w_{k-1})} \\ &= \frac{\gamma_{k-1}}{\delta_{k-1} - \frac{\beta_{k-1}\gamma_{k-1}}{\alpha_{k-2}}}, \end{aligned} \tag{23}$$

this corresponds with the choice for α in Algorithm 5, line 8. Likewise for (16), the right-hand side of

$$p_k^T A q_j = u_k^T A q_j + \beta_k p_{k-1}^T A q_j , \tag{24}$$

is zero for $j < k-1$ by induction and is zero for $j = k-1$ if

$$\begin{aligned}\beta_k &= -\frac{u_k^T A q_{k-1}}{p_{k-1}^T A q_{k-1}} = -\frac{(u_k, q_{k-1})_A}{(p_{k-1}, q_{k-1})_A} = -\frac{\left(u_k, \frac{u_k - u_{k-1}}{-\alpha_{k-1}}\right)_A}{\left(u_{k-1} + \beta_{k-1} p_{k-2}, \frac{u_k - u_{k-1}}{-\alpha_{k-1}}\right)_A} \\ &= -\frac{(u_k, u_k)_A - (u_k, u_{k-1})_A}{(u_{k-1}, u_k)_A - (u_{k-1}, u_{k-1}) + \beta_{k-1}\left[(p_{k-2}, u_k)_A - (p_{k-2}, u_{k-1})_A\right]} \\ &= \frac{(u_k, u_k)_A}{(u_{k-1}, u_{k-1})} = \frac{\gamma_k}{\gamma_{k-1}}, \end{aligned} \tag{25}$$

which corresponds to the definition of β in Algorithm 5, line 8. This concludes the proof of the orthogonality conditions (15) and (16).

To show the minimization property, first note that $AM^{-1}A$ is symmetric and positive definite and hence $\|\cdot\|_{AM^{-1}A}$ defines a valid norm. To show that $x_k \in x_0 + \mathcal{K}_k(M^{-1}A, u_0)$ minimizes $\|e\|_{AM^{-1}A}$, we consider an arbitrary point $x = x_k - \Delta x \in x_0 + \mathcal{K}_k(M^{-1}A, M^{-1}b)$ with error $e = \hat{x} - x = e_k + \Delta x$. Then

$$\begin{aligned}\|e\|_{AM^{-1}A}^2 = \|e_k + \Delta x\|_{AM^{-1}A}^2 &= (e_k + \Delta x)^T AM^{-1}A(e_k + \Delta x) \\ &= e_k^T AM^{-1}Ae_k + \Delta x^T AM^{-1}A\Delta x + 2e_k^T AM^{-1}A\Delta x \\ &= \|e_k\|_{AM^{-1}A}^2 + \|\Delta x\|_{AM^{-1}A}^2 + 2u_k^T A\Delta x . \end{aligned} \tag{26}$$

The last term is zero by orthogonality property (15), and the second term is always positive and only zero for $\Delta x = 0$. Hence, $x_k = x$ is the unique point that minimizes $\|e\|_{AM^{-1}A}$.

For $M = I$, $\|e_k\|_{AM^{-1}A} = \|e_k\|_{A^2} = \|Ae_k\|_2 = \|r_k\|_2$ is minimized. □

From Theorem 3.1 and the A-orthogonality of the residual vectors, relation (15), it is clear that Algorithm 5 is indeed a minimal residual method and is equivalent to Conjugate Residuals.

3.4 Method due to Gropp

Recently, an asynchronous CG method has been presented by Gropp [18]. This method, shown as Algorithm 6, still has two global synchronization points per iteration. One reduction (line 9) can be overlapped with the matrix-vector product (line 10) and the other (line 3) with application of the preconditioner (line 4). Compared to standard CG, only two additional AXPYs are required. Algorithm 6 uses the same notation as above: $r_i = b - Ax_i$, $u_i = M^{-1}r_i$, $s_i = Ap_i$, $q_i = M^{-1}s_i$ and $w_i = Au_i$. However, the scalar δ is defined differently.

Algorithm 6 Gropp's asynchronous CG

```
1: $r_0 := b - Ax_0$;   $u_0 := M^{-1}r_0$;   $p_0 := u_0$;   $s_0 := Ap_0$;   $\gamma_0 := (r_0, u_0)$
2: for $i = 0, \dots$ do
3:     $\delta := (p_i, s_i)$
4:     $q_i := M^{-1}s_i$
5:     $\alpha_i := \gamma_i / \delta$
6:     $x_{i+1} := x_i + \alpha_i p_i$
7:     $r_{i+1} := r_i - \alpha_i s_i$
8:     $u_{i+1} := u_i - \alpha_i q_i$
9:     $\gamma_{i+1} := (r_{i+1}, u_{i+1})$
10:    $w_{i+1} := Au_{i+1}$
11:    $\beta_{i+1} := \gamma_{i+1} / \gamma_i$
12:    $p_{i+1} := u_{i+1} + \beta_{i+1} p_i$
13:    $s_{i+1} := w_{i+1} + \beta_{i+1} s_i$
14: end for
```

	flops	time (excl, AXPYs, DOTs)	#glob syncs	memory
CG	10	2G + SpMV + PC	2	4
Chron/Gear-CG	12	G + SpMV + PC	1	5
pipe-CG	20	max(G, SpMV + PC)	1	9
pipe-CR	16	max(G, SpMV) + PC	1	7
Gropp-CG	14	max(G, SpMV) + max(G, PC)	2	7

Table 1: Overview of the different CG variations. Column *flops* lists the number of flops ($\times N$) for AXPYs and dot-products. The *time* column has the time spent in global all-reduce communication (G), in the matrix-vector product (SpMV) and the preconditioner (PC). Column *#global synchronizations* has the number of global communication phases per iteration. The *memory* column counts the number of vectors that need to be kept in memory (excluding x and b).

Table 1 gives a brief overview of the methods presented in this section with their parallel properties. The five methods listed are standard CG, the Chronopoulos/Gear variant of CG, preconditioned pipelined CG (pipe-CG), pipelined CR (pipe-CR) and the method by Gropp. In Table 1, the column *flops* lists the number of floating point operations for AXPYs and dot-products per iteration ($\times$ the vector length for the total). *Memory* lists how many vectors need to be stored, not counting x and b. The last column counts the number of global reductions. The time spent in global communication and application of the sparse matrix-vector product (SpMV) and the preconditioner (PC) is given in the *time* column. Here, G is the time for a global all-reduce (a reduction followed by a broadcast), which is mostly latency bound and can be overlapped with the SpMV or with the preconditioner or with both. The pipelined CG method offers the most potential overlap with the reduction.

4 Numerical Results

Numerical results are presented with the different CG methods, using different preconditioners, for a wide range of matrices from applications. A possible strategy to improve the maximum attainable accuracy of the methods is given in Section 4.2.

4.1 Problems from Matrix Market

The CG methods presented above have been implemented using PETSc (the Portable, Extensible Toolkit for Scientific Computation[1]). However, PETSc provides a CG method with a single global reduction due to Eijkhout [11, 12] that is based on the method by Chronopoulos and Gear, Algorithm 2, but differs slightly in the way the scalar α is computed. This single reduction method in PETSc can be used with the command line options `-ksp_type cg` and `-ksp_cg_single_reduction`. We shall compare with this implementation instead of Algorithm 2.

The methods presented above have been tested on a wide range of linear systems. Table 2 lists all square, real and symmetric positive definite matrices from Matrix Market[2], which covers a wide range of condition numbers, listed in column two of Table 2. Columns 3 and 4 give the total number of rows and number of nonzeros for each of the matrices. A linear system for each of these matrices with exact solution $\hat{x}_i = 1/\sqrt{N}$, with N the number of rows, such that $\|\hat{x}\|_2 = 1$ and right-hand side $b = A\hat{x}$ is solved with all of the presented methods. The initial guess was always $x_0 = 0$ and the default PETSc stopping criterion

$$\|u_i\|_2 < \max(10^{-5}\|b\|_2, 10^{-50}) \tag{27}$$

was used with $u_i = M^{-1}(Ax_i - b)$ the preconditioned residual. The rest of the table lists the required number of iterations for the 5 different CG methods, either without preconditioner, with Jacobi preconditioner or with incomplete Cholesky factorization, ICC(0), preconditioner.

A '-' entry in the table denotes failure to meet the stopping criterion within 10.000 iterations. In many cases, the pipe-CR method clearly reaches the specified tolerance in less iterations than the other methods, except for the nos7 matrix. In all cases, the single reduction CG method behaves similar to standard CG. In most cases, pipe-CG and Gropp-CG need about as many iterations as standard CG, with a few glitches for pipe-CG (mostly when used without preconditioner). The bcsstk and bcsstm matrices are the K and M matrices respectively from a generalized eigenvalue problem $Kx = \lambda Mx$. The bcsstm matrices are all diagonal matrices, hence the convergence in one step with even the trivial Jacobi preconditioner. These matrices are still included for illustration.

[1] http://www.mcs.anl.gov/petsc/
[2] http://math.nist.gov/MatrixMarket/

				no preconditioner					Jacobi preconditioner					ICC preconditioner				
matrix	cond(A)	N	#nnz	CG	CG1	Gropp	p-CR	p-CG	CG	CG1	Gropp	p-CR	p-CG	CG	CG1	Gropp	p-CR	p-CG
bcsstk14	1.3e+10	1806	63454	1268	1369	1268	270	1374	203	203	203	126	203	709	709	710	300	714
bcsstk15	8e+09	3948	117816	2325	2419	2261	478	2474	448	449	450	228	451	1300	1306	1311	329	1313
bcsstk16	65	4884	290378	189	190	190	176	190	123	123	123	116	123	26	26	26	26	26
bcsstk17	65	10974	428650	4280	4378	4185	1965	4670	1961	1960	1958	368	1970	2905	2984	2944	1376	3047
bcsstk18	65	11948	149090	2858	2718	2850	569	2991	594	587	596	182	595	1391	1391	1392	363	1393
bcsstk27	7.7e+04	1224	56126	428	432	428	345	438	170	171	171	139	171	15	15	15	15	15
bcsstm19	2.3e+05	817	817	166	175	165	171	187	1	1	1	1	1	1	1	1	1	1
bcsstm20	2.6e+05	485	485	127	131	129	127	136	1	1	1	1	1	1	1	1	1	1
bcsstm21	24	3600	3600	3	3	3	3	3	1	1	1	1	1	1	1	1	1	1
bcsstm22	9.4e+02	138	138	42	43	42	42	42	1	1	1	1	1	1	1	1	1	1
bcsstm23	9.5e+08	3134	3134	596	603	596	336	616	1	1	1	1	1	1	1	1	1	1
bcsstm24	1.8e+13	3562	3562	366	375	365	317	381	1	1	1	1	1	1	1	1	1	1
bcsstm25	6.1e+09	15439	15439	420	369	419	268	453	1	1	1	1	1	1	1	1	1	1
bcsstm26	2.6e+05	1922	1922	476	436	451	301	462	1	1	1	1	1	1	1	1	1	1
gr_30_30	3.8e+02	900	7744	33	33	33	33	33	33	33	33	33	33	16	16	16	16	16
nos1	2.5e+07	237	1017	953	1005	950	453	1196	303	323	308	188	345	304	316	321	143	331
nos2	6.3e+09	957	4137	643	642	647	369	963	1746	1809	1771	474	1937	217	226	247	86	219
nos3	7.3e+04	960	15844	219	219	219	215	220	183	183	183	180	183	41	41	41	41	41
nos4	2.7e+03	100	594	72	72	71	71	72	67	67	67	66	67	19	19	19	19	19
nos5	2.9e+04	468	5172	227	228	227	188	228	130	130	130	119	130	33	33	33	33	33
nos6	8e+06	675	3255	198	165	164	121	187	78	78	78	19	78	23	23	23	11	23
nos7	4.1e+09	729	4617	3300	3286	3025	-	664	67	67	67	27	67	21	21	21	8	21
s1rmq4m1	1.8e+06	5489	262411	2352	2398	2419	1253	2362	622	623	622	535	624	82	82	82	80	82
s1rmt3m1	2.5e+06	5489	217651	2801	2849	2830	1512	3017	700	702	703	605	703	158	158	158	149	158
s2rmq4m1	1.8e+08	5489	263351	2189	2124	2188	847	2301	1272	1475	1272	474	1274	102	102	102	51	102
s2rmt3m1	2.5e+08	5489	217681	3281	3147	3286	1166	3161	2093	2093	2096	693	2104	222	222	222	114	222
s3dkq4m2	1.9e+11	90449	2455670	3712	3712	3712	1053	3712	-	-	-	1309	-	1798	1798	1798	960	1798
s3dkt3m2	3.6e+11	90449	1921955	4939	4939	4939	1455	4940	-	-	-	1776	-	1370	1369	1369	338	1369
s3rmq4m1	1.8e+10	5489	262943	1539	1565	1569	478	1688	2952	2958	2963	499	2993	129	129	129	105	129
s3rmt3m1	2.5e+10	5489	217669	2104	2165	2139	604	2328	4445	4446	4493	758	4479	236	236	236	125	236
s3rmt3m3	2.4e+10	5357	207123	2976	3096	3006	857	3204	4848	4869	4865	576	4916	340	339	340	215	340

Table 2: All real, square and symmetric positive definite matrices from Matrix Market, listed with their condition number, number of columns/rows and the total number of nonzeros. A linear system is solved with each of these matrices with the 5 different CG variations presented, using either no preconditioner, a Jacobi preconditioner or an incomplete Cholesky factorization preconditioner.

Figure 1 illustrates the convergence history for a few randomly selected matrices from Table 2. Figure 1 (top-left) shows the relative residual $\|b - Ax_i\|_2/\|b\|_2$ for the bcsstk15 matrix with Jacobi preconditioning. Figure 1 (top-right) shows the s1rmt3m1 matrix with ICC preconditioner while in (bottom-left) and (bottom-right) the gr-30-30 and the nos2 matrices are shown respectively without preconditioner. For these experiments, the relative tolerance was set to rtol $= 10^{-20}$. Observe from Figure 1 that the convergence for CG, single reduction CG and Gropp-CG is nearly indistinguishable. The pipe-CG method converges as standard CG but levels off sooner, leading to a less accurate solution. The pipe-CR method behaves differently, typically with faster initial convergence, but the same asymptotic behavior. Also, the pipe-CR method has worse final attainable accuracy.

These results are as expected, since all methods, except pipe-CR are mathematically equivalent to standard CG, i.e., they minimize the A norm of the error, $\|\hat{x} - x_i\|_A$, at iteration i. The pipe-CR method minimizes the $AM^{-1}A$ norm of the error or, without preconditioner, pipe-CR minimizes $\|e_i\|_{A^2} = \|r_i\|$ in every iteration, which is confirmed by the monotonic convergence of the residual in Figure 1 (bottom).

4.2 Improving accuracy

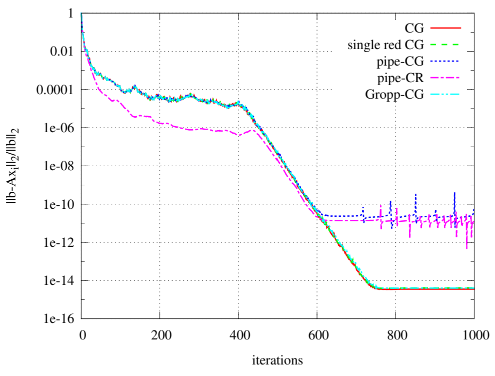
In order to improve the maximum attainable accuracy for the pipelined CG methods, we suggest to use a so-called residual replacement strategy, see for example [36, 5, 32]. In many Krylov iterations, the solution x_i and residual vector r_i are updated as

$$x_{i+1} = x_i + \alpha p_i \,, \qquad r_{i+1} = r_i - \alpha A p_i \,. \tag{28}$$

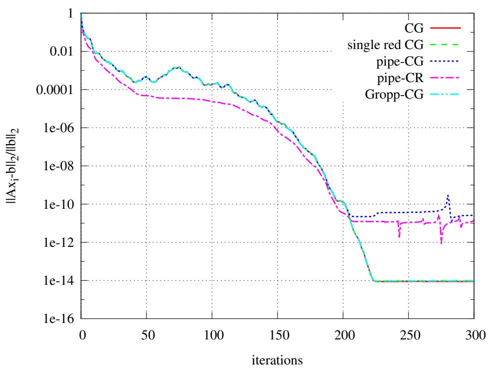
Both solution and residual will be affected differently by rounding errors. Any error made in the update for x is not reflected in r, since x is typically not used in the rest of the iteration. This leads to the well known problem that the *updated* residual r_i and the *true* residual $b - Ax_i$ start to deviate. A simple remedy is to periodically replace the updated r_i by $r_i = b - Ax_i$. However, if this is done too frequently, the superlinear convergence often observed in CG can be lost. On the other hand, when the difference between updated and true residual gets too big, convergence stagnates too soon, resulting in a less accurate final solution.

We illustrate the potential of such a residual replacement strategy. In the pipe-CG and pipe-CR methods, the updated residual r_i, and the updated preconditioned residual u_i, will be replaced every 50-th iteration by the true residual $r_i = b - Ax_i$ and by the true preconditioned residual $u_i = M^{-1}r_i$ respectively. Furthermore, the vector $w_i = Au_i$ is also recomputed. For pipe-CR, the original residual r_i does not need to be stored, but needs to be computed to evaluate u_i anyway. Figure 2 shows the convergence for two matrices also shown in Figure 1, now repeated with the residual replacement strategy. For pipe-CG applied to bcsstk15 with Jacobi preconditioner the error after 1.000 iterations drops from $\|e_{1.000}\|_2 = 8.01\text{e-}8$ without residual replacement to 4.78e-11 with residual replacement. Likewise, for s1rmt3m1 with ICC preconditioner, the error drops from 1.01e-8 to 3.82e-12. For pipe-CR, bcsstk15, Jacobi preconditioner, the error goes from 2.58e-8 to 1.03e-9 and for s1rmt3m1 with ICC preconditioner, from 2.59e-9 to 1.65e-11. This is always an improvement of at least one order of magnitude.

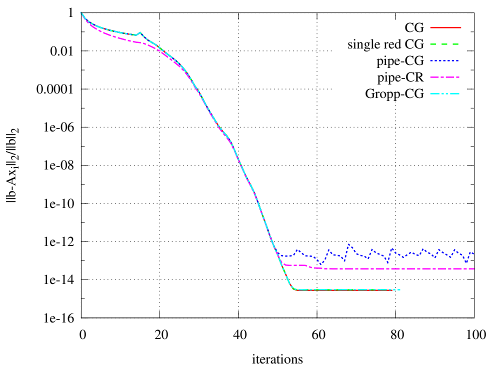
Of course, the value 50, also used in [32], is arbitrary and does not yield best possible results for all linear systems. Complicated strategies have been presented in the literature to determine good restart point. A strategy proposed by Neumaier [33] was to replace the updated residual by the true residual whenever the residual norm becomes smaller than any previously attained value. At that point, also the solution vector x is updated with the combined contributions since the last restart point, a so-called group update. However, this strategy was proposed with the irregular convergence behavior of methods like BiCG and CGS in mind, where, as is well known, iterations with large residual can lead to early stagnation of the final residual. Van der Vorst and Ye [36] derive an estimate for the deviation between updated and true residual and only do the residual replacement and group update when this bound exceeds some tolerance ϵ, typically the square root of the machine precision. Recently, Carson and Demmel [5] extended the bound from [36]



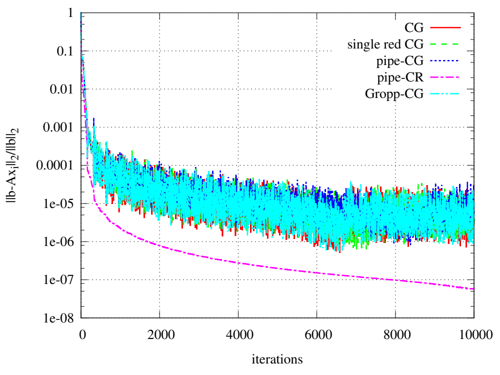
(a) bcsstk15 with Jacobi preconditioner



(b) s1rmt3m1 with ICC preconditioner

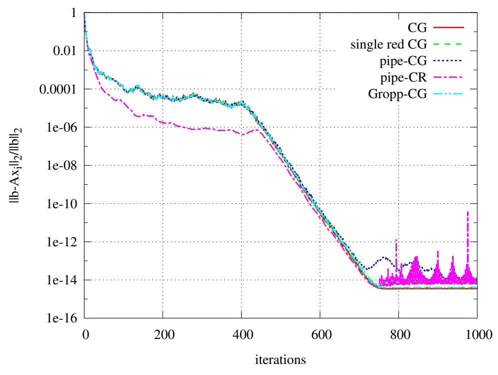


(c) gr-30-30 without preconditioner

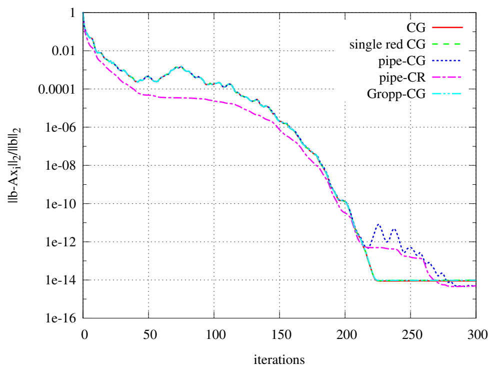


(d) nos2 without preconditioner

Figure 1: Convergence history for the different CG methods applied to four different symmetric positive definite test matrices from applications (see also Table 2). Convergence of CG, single reduction CG and Gropp CG is nearly indistinguishable. The pipelined methods level off somewhat sooner.



(a) bcsstk15 with Jacobi preconditioner



(b) s1rmt3m1 with ICC preconditioner

Figure 2: Convergence history for the different CG methods applied to two of the matrices also shown in Figure 1. However, for these tests the updated vectors r_i, u_i and w_i are replaced by $r_i = b - Ax_i$, $u_i = M^{-1}r_i$ and $w_i = Au_i$ every 50-th iteration. This yields an improvement in maximum attainable accuracy of several orders of magnitude.

to communication-avoiding (CA) or s-step Krylov methods and present an implementation for CA-CG and CA-BiCG. The residual replacement strategy has proven to be an effective strategy to improve the final accuracy of short term Krylov methods. Each replacement incurs additional work, but since replacements occur infrequently the performance impact is limited. However, a general replacement strategy applicable to the pipelined methods presented here remains future work.

Stability is also negatively impacted in the pipelined methods by the extra multiplication with the matrix A. A possible improvement might be to add a shift in the matrix-vector product, similar to what is also done in s-step Krylov methods [2, 24] and in [16] for pipelined GMRES. The CG iteration can provide information on the spectrum of A, which can be used to determine good shifts.

5 Parallel performance

Different variations to standard CG have been presented in order to overcome the bottleneck of global communication on large parallel machines. In Section 5.1, the parallel performance of the different methods is studied using a test problem on a medium sized parallel machine. Section 5.2 lists several candidate scenarios which could benefit from the pipelined CG or CR solvers.

5.1 Benchmark application

The parallel experiments are performed on an IBM iDataPlex machine from NERSC[3] called Carver. The full system has 1,202 compute nodes (9,984 processor cores) with a theoretical peak performance of 106Tflops/sec. All nodes are interconnected by 4×QDR InfiniBand technology, providing 32Gb/s of point-to-point bandwidth for high-performance message passing and I/O. Most nodes (1,120) have two quad-core Intel Xeon X5550 Nehalem 2.67GHz processors (eight cores/node) and since parallel jobs on Carver are limited to 64 nodes, only these Nehalem nodes were used. The pipe-CG, pipe-CR and Gropp-CG algorithms have been implemented using PETSc. PETSc provides a construct for asynchronous dot-products:

[3]http://www.nersc.gov/users/computational-systems/carver/

- VecDotBegin(...,&dot);
- PetscCommSplitReductionBegin(comm);
- *// ... other work, like an* SpMV *or application of the preconditioner*
- VecDotEnd(...,&dot);

where PetscCommSplitReductionBegin starts a non-blocking reduction by a call to `MPI_Iallreduce`. This returns an `MPI_Request` which is passed as an argument to `MPI_Wait` in VecDotEnd. Note that non-blocking collectives, including `MPI_Iallreduce`, only became available in MPI with the MPI-3 standard. The MPI library used for the experiments is MPICH-3.0rc1[4].

To asses the parallel performance of pipelined CG, the two-dimensional Bratu partial differential equation

$$-\Delta u - \lambda \exp(u) = -\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} - \lambda \exp(u) = 0\,, \qquad 0 < x, y < 1\,, \tag{29}$$

with boundary conditions

$$u = 0, \qquad \text{for } x = 0, x = 1, y = 0, y = 1\,, \tag{30}$$

will be used. The Bratu equation has applications in solid fuel ignition, chemical reaction theory, radiative heat transfer, nanotechnology and it even appears in Chandrasekhar's model for the expansion of the universe. This nonlinear equation is discretized using the standard 5-point stencil and solved with Newton iteration, which repeatedly calls a linear solver. The problem is solved on a 2049×2049 PETSc distributed array (DMDA) with λ, the parameter controlling the nonlinearity, set to $\lambda = 6$. This leads to a linear equation with about 4.2 million unknowns. The sparse matrix is constructed explicitly and stored distributed over the nodes in compressed sparse row format. The problem is not solved to a specified tolerance, but only a single Newton iteration is performed. For the linear solve, the five presented CG variations where used, with a maximum of 10 iterations. This is repeated 30 times, and from this the minimum time is taken to exclude effects of any system interference on the timings. For these tests, we make sure all methods do 10 iterations, i.e., they minimize the error over the same Krylov space. For the pipelined methods and for Gropp-CG, this requires one additional matrix-vector multiplication and one extra application of the preconditioner compared to standard CG. These extra operations are to start up the pipeline.

Figure 3 (left) shows the times recorded for 10 iterations on the 2049^2 problem without preconditioner, performed on 64 nodes with 8 MPI processes per node. This is a strong scaling experiment, with 8200 grid points per core when using all 512 cores. Figure 3 (right) shows the speedup of the different methods over standard CG on 1 node (using 8 MPI processes). Figure 4 shows timings for a similar experiment, but with block-Jacobi preconditioning. For the problem considered here, the ICC(0) preconditioner did not lead to convergence and hence, an incomplete LU factorization, ILU(0), was used inside each Jacobi block, with one block per core.

Figure 5 shows the same experiments as in Figures 3 and 4 but shows speedups over the time on one node for each method individually. This shows clearly that the pipelined CG and CR methods scale significantly better than the alternatives with blocking global communication. The best speedup is attained by pipe-CG that also offers the most potential for overlap. The superlinear convergence is likely due to cache effects. Note that in the implementation of the pipelined methods, the different operations where not fused, as was suggested at the end of Section 2.2. However, we believe this fusion of operations is important to improve the communication with slow memory and could yield an extra on-node performance gain compared to standard CG.

5.2 Possible use cases and further optimizations

The aim of this work is to develop a generally applicable highly scalable linear solver. However, the resulting pipelined CG algorithm is a trade-off between more floating point operations, more

[4]http://www.mpich.org/

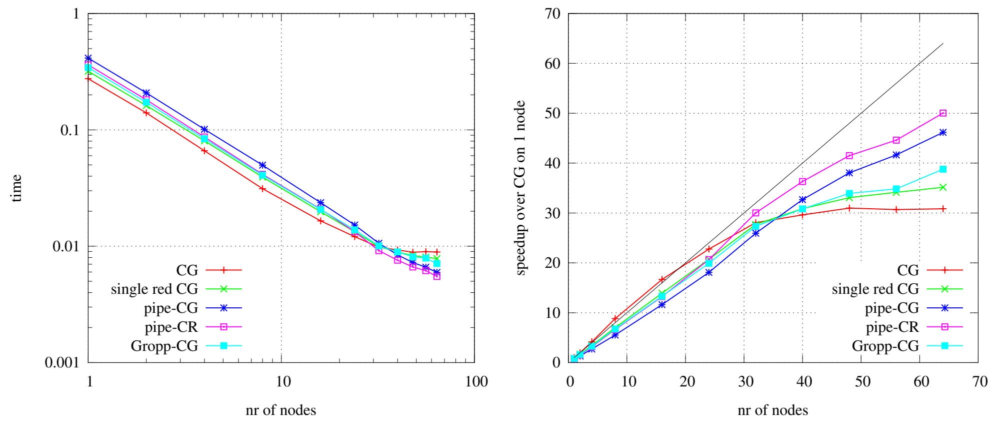


Figure 3: Left: Timings for 10 linear iterations with the different CG methods on up to 64 nodes, using 8 MPI processes per node, for a linear problem with 4.2 million unknowns. No preconditioner is used. Right: Speedup over standard CG on 1 node with 8 MPI processes per node.

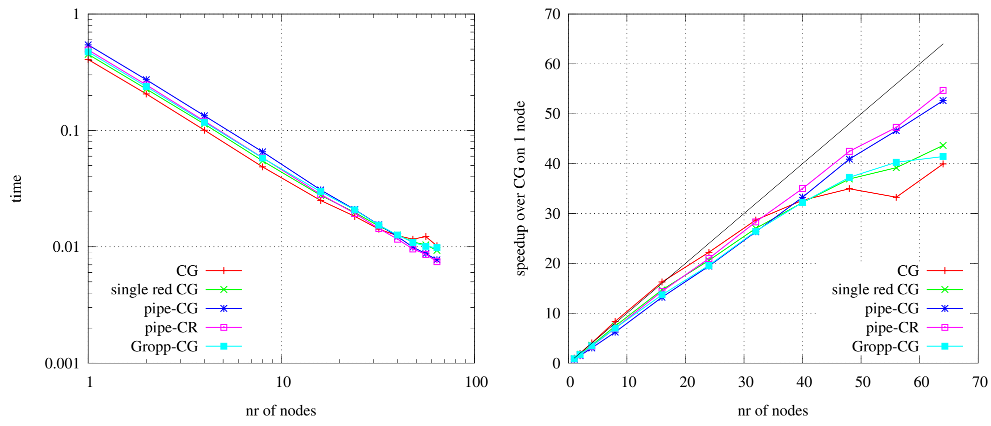


Figure 4: As in Figure 3, but with block Jacobi preconditioner with one block per core and ILU(0) inside each Jacobi block. Left: Timings for 10 linear iterations with the different CG methods on up to 64 nodes, using 8 MPI processes per node, for a linear problem with 4.2 million unknowns. Right: Speedup over standard CG on 1 node with 8 MPI processes per node.

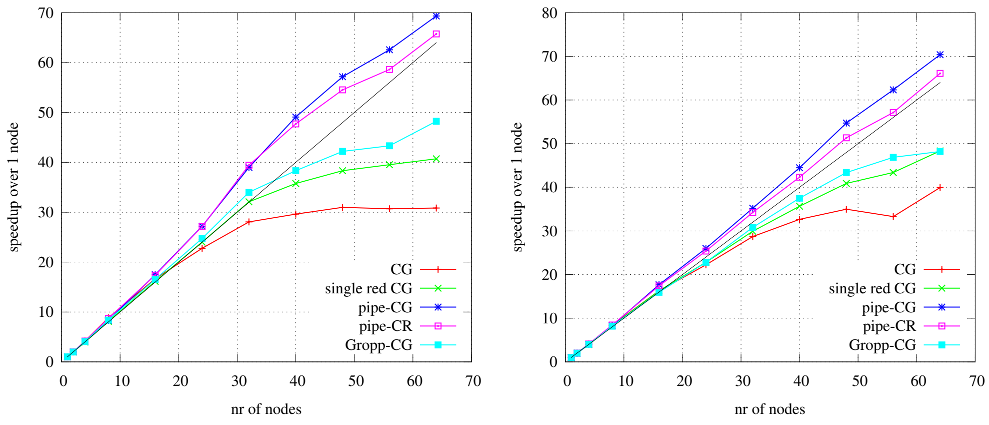


Figure 5: This shows the same experiments as in Figures 3 and 4. Left: No preconditioner. Right: Block Jacobi with ILU(0) preconditioner. However, now the speedup is plotted for the individual methods.

memory usage, and improved scalability. Therefore, not every application will benefit from this trade-off. We identify a set of scenarios where the improved scalability will likely pay off.

As was already mentioned, to get fast convergence, preconditioning is essential. In a solver with a near optimal preconditioner, most of the time is probably spent in application of the preconditioner and a pipelined outer CG solver will not pay off. However, problems where no good preconditioner is available would still require many CG iterations and could benefit. For instance augmented Lagrangian systems from contact and interior points are examples of nearly-singular symmetric systems for which a multigrid preconditioner is hard to construct.

Also when repeatedly solving linear system, e.g., in a nonlinear solver, in optimization or in implicit time integration, many linear solver iterations are required which could benefit from the pipelined CG solver. In some special cases, a relatively small linear system is solved on a parallel machine, which makes the linear solver mostly latency bound. Hiding the latency as in pipelined CG directly pays off in such cases. Examples are: the solver for the coarsest grid in a geometric multigrid U-cycle [38], the field solver in coupled physics methods such as the Particle-In-Cell [21] method where the size of the field is typically small compared to the number of particles used in the simulation, and domain decomposition and Schur complement solvers.

When the reductions are completely overlapped, the pipelined CG methods will scale as the SpMV, and the SpMV communication might become the bottleneck. In pipe-CG, Algorithm 4, application of the preconditioner (line 5) is followed immediately by the SpMV (line 6). A possible optimization would be to fuse the two operations. Consider for instance a polynomial preconditioned linear system $p^s(A)Ax = p^s(A)b$ with $p^s(A)$ a degree s polynomial in A. Applying both the preconditioner and the matrix-vector product could be done in a single latency by the matrix-powers kernel [14, 17]. For structured grid problems, this is relatively easy to implement by communicating a wider ghostregion, which is also called communication aggregation, since the same amount of data is transferred, only in less messages [37]. The polynomial preconditioner can then also be implemented efficiently using techniques as cache-oblivious, wavefront or time-skewed stencil loops [7]. Information about the spectrum of A can be computed as a side product of the CG iteration and can help to determine an optimal polynomial [30]. However, in a similar way, steps in a multigrid fine-grid smoother can be fused with the matrix-vector product from CG.

6 Conclusions

We presented pipelined variations of CG where the global communication can be overlapped with local work, such as the matrix-vector product and the preconditioner. Two different ways do include preconditioning are discussed, leading to two algorithms with different trade-offs regarding scalability and total number of operations. For a strong scaling experiment on a medium sized cluster, we show improved scalability and also faster runtime compared to standard CG. Numerical tests on matrices from Matrix Market with a wide range of condition numbers show that the convergence of the new methods is in line with standard CG. We also compare with a method recently presented by B. Gropp [18] that has somewhat better numerical properties but offers less overlap with the global communication and thus leads to a less scalable algorithm.

For large machines where a global reduction is expensive compared to a matrix-vector product, it can happen that the latency of the global reduction is not completely overlapped in the pipelined methods. If this turns out to become a bottleneck (perhaps on extremely large or on future systems), the pipelined strategy can be generalized to longer pipelining depths. In this case, the global reduction can be overlapped with multiple CG iterations (matrix-vector products), as was already presented for GMRES in [16]. However, longer pipelines also increase the recurrence length and the required number of operations and memory.

The presented pipelined methods can be seen as an alternative to s-step CG [5, 34]. Historically, s-step CG has had a “bad reputation” [35, 30] due to the numerical problems for increasing s. By recent advances such as different Krylov bases [2, 24] (like Newton or Chebyshev) and a residual replacement strategy [5], these methods have been applied successfully with quite large values of s. However, a problem with s-step methods remains the combination with preconditioning. Although some progress has been made in this regard recently, finding communication avoiding preconditioners that combine well with s-step methods remains a challenge. The pipelined methods presented in this work can be used with any preconditioner.

Acknowledgments

This work is funded by Intel and by the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT). This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. We also thank Jed Brown from Argonne National Laboratory for interesting comments.

References

[1] T. Ashby, P. Ghysels, W. Heirman, and W. Vanroose. The impact of global communication latency at extreme scales on krylov methods. *Algorithms and Architectures for Parallel Processing*, pages 428–442, 2012.

[2] Z. Bai, D. Hu, and L. Reichel. A Newton basis GMRES implementation. *IMA Journal of Numerical Analysis*, 14(4):563–581, 1994.

[3] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition.* SIAM, Philadelphia, PA, 1994.

[4] J. Brown. User-defined Nonblocking Collectives Must Make Progress. IEEE Technical Committee on Scalable Computing (TCSC), 2012.

[5] E. Carson and J. Demmel. A Residual Replacement Strategy for Improving the Maximum Attainable Accuracy of s-step Krylov Subspace Methods. Technical Report UCB/EECS-2012-44, University of California at Berkeley, 2012.

[6] E. Carson, N. Knight, and J. Demmel. Avoiding Communication in Two-Sided Krylov Subspace Methods. Technical report, University of California at Berkeley, Berkeley, CA, USA, 2011.

[7] M. Christen, O. Schenk, and H. Burkhart. Patus: A code generation and autotuning framework for parallel iterative stencil computations on modern microarchitectures. In *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 676–687. IEEE, 2011.

[8] A. T. Chronopoulos. s-Step iterative methods for (non) symmetric (in) definite linear systems. *SIAM journal on numerical analysis*, 28(6):1776–1789, 1991.

[9] A. T. Chronopoulos and C. W. Gear. s-step iterative methods for symmetric linear systems. *Journal of Computational and Applied Mathematics*, 25(2):153–168, 1989.

[10] A. T. Chronopoulos and C. D. Swanson. Parallel iterative S-step methods for unsymmetric linear systems. *Parallel Computing*, 22(5):623–641, 1996.

[11] E. F. D'Azevedo, V. L. Eijkhout, and C. H. Romine. Lapack Working Note 56 Conjugate Gradient Algorithms with Reduced Synchronization Overhead on Distributed Memory Multiprocessors, 1999.

[12] E. F. D'Azevedo and C. H. Romine. Reducing communication costs in the conjugate gradient algorithm on distributed memory multiprocessors. Technical report, Oak Ridge National Lab, TN, 1992.

[13] E. De Sturler and H. A. Van der Vorst. Reducing the effect of global communication in GMRES(m) and CG on parallel distributed memory computers. *Applied Numerical Mathematics*, 18(4):441–459, 1995.

[14] J. Demmel, M. Hoemmen, M. Mohiyuddin, and K. Yelick. Avoiding communication in sparse matrix computations. *2008 IEEE International Symposium on Parallel and Distributed Processing*, pages 1–12, April 2008.

[15] J. W. Demmel, M. T. Heath, and H. A. Van Der Vorst. Parallel numerical linear algebra. *Acta Numerica*, 2(-1):111–197, 1993.

[16] P. Ghysels, T. J. Ashby, K. Meerbergen, and W. Vanroose. Hiding global communication latency in the GMRES algorithm on massively parallel machines. *SIAM Journal on Scientific Computing*, 2012.

[17] P. Ghysels, P. Kłosiewicz, and W. Vanroose. Improving the arithmetic intensity of multigrid with the help of polynomial smoothers. *Numerical linear algebra with applications, Special issue Copper Mountain 2011*, 19(2):253–267, 2012.

[18] W. Gropp. Update on Libraries for Blue Waters. http://jointlab.ncsa.illinois.edu/events/workshop3/pdf/presentations/Gropp-Update-on-Libraries.pdf.

[19] V. Hernandez, J. E. Roman, and A. Tomas. Parallel Arnoldi eigensolvers with enhanced scalability via global communications rearrangement. *Parallel Computing*, 33(7-8):521–540, 2007.

[20] M. R. Hestenes and E. Stiefel. Methods of Conjugate Gradients for Solving Linear Systems. *Journal of Research of the National Bureau of Standards*, 49(6), 1952.

[21] R. W. Hockney and J. W. Eastwood. *Computer Simulation Using Particles.* Adam Hilger, 1988.

[22] T. Hoefler, T. Schneider, and A. Lumsdaine. LogGOPSim - Simulating Large-Scale Applications in the LogGOPS Model. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 597–604. ACM, Jun. 2010.

[23] T. Hoefler, J. Squyres, G. Bosilca, G. Fagg, A. Lumsdaine, and W. Rehm. Non-Blocking Collective Operations for MPI-2. *Open Systems Lab, Indiana University, Tech. Rep*, 8, 2006.

[24] M. Hoemmen. *Communication-avoiding Krylov subspace methods.* PhD thesis, University of California, 2010.

[25] W. D. Joubert and G. F. Carey. Parallelizable restarted iterative methods for nonsymmetric linear systems. part I: Theory. *International journal of computer mathematics*, 44(1-4):243–267, 1992.

[26] S. K. Kim and A. T. Chronopoulos. An efficient parallel algorithm for extreme eigenvalues of sparse nonsymmetric matrices. *International Journal of High Performance Computing Applications*, 6(4):407–420, 1992.

[27] L. C. McInnes, B. Smith, H. Zhang, and R. Tran Mills. Hierarchical and nested krylov methods for extreme-scale computing. Technical Report ANL/MCS-P2097-0612, Argonne National Laboratory, 2012.

[28] G. Meurant. Multitasking the conjugate gradient method on the CRAY X-MP/48. *Parallel Computing*, 5(3):267–280, 1987.

[29] Y. Saad. Practical use of some Krylov subspace methods for solving indefinite and nonsymmetric linear systems. *SIAM journal on scientific and statistical computing*, 5(1):203–228, 1984.

[30] Y. Saad. Krylov subspace methods on supercomputers. *SIAM Journal on Scientific and Statistical Computing*, 10:1200, 1989.

[31] Y. Saad. *Iterative methods for sparse linear systems.* Society for Industrial Mathematics, 2003.

[32] J. R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain, 1994.

[33] G. L. G. Sleijpen and H. A. van der Vorst. Reliable updated residuals in hybrid bi-cg methods. *Computing*, 56(2):141–163, 1996.

[34] S. A. Toledo. *Quantitative performance modeling of scientific computations and creating locality in numerical algorithms.* PhD thesis, Massachusetts Institute of Technology, 1995.

[35] H. A. Van der Vorst. *Iterative Krylov methods for large linear systems*, volume 13. Cambridge University Press, 2003.

[36] Van Der Vorst, H. A. and Ye, Q. Residual replacement strategies for Krylov subspace iterative methods for the convergence of true residuals. *SIAM Journal on Scientific Computing*, 22(3):835–852, 1999.

[37] S. Williams, D. D. Kalamkar, A. Singh, A. M. Deshpande, B. Van Straalen, M. Smelyanskiy, A. Almgren, P. Dubey, J. Shalf, and L. Oliker. Optimization of geometric multigrid for emerging multi-and manycore processors. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 96. IEEE Computer Society Press, 2012.

[38] D. Xie and L. R. Scott. An analysis of parallel u-cycle multigrid method.

[39] L. T. Yang. The improved CGS method for large and sparse linear systems on bulk synchronous parallel architectures. In *Algorithms and Architectures for Parallel Processing, 2002. Proceedings. Fifth International Conference on*, pages 232–237. IEEE, 2002.

[40] L. T. Yang and R. P. Brent. The improved BiCGStab method for large and sparse unsymmetric linear systems on parallel distributed memory architectures. In *Algorithms and Architectures for Parallel Processing, 2002. Proceedings. Fifth International Conference on*, pages 324–328. IEEE, 2002.

[41] L. T. Yang and R. P. Brent. The Improved Parallel BiCG Method for Large and Sparse Unsymmetric Linear Systems on Distributed Memory Architectures. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium, IPDPS 2002.* IEEE, 2003.

[42] T. R. Yang and H. X. Lin. The improved quasi-minimal residual method on massively distributed memory computers. In *High-Performance Computing and Networking*, pages 389–399. Springer, 1997.