SIAM J. SCI. COMPUT.
Vol. 35, No. 1, pp. C48–C71

HIDING GLOBAL COMMUNICATION LATENCY IN THE GMRES ALGORITHM ON MASSIVELY PARALLEL MACHINES*

P. GHYSELS†, T. J. ASHBY‡, K. MEERBERGEN§, AND W. VANROOSE†

Abstract. In the generalized minimal residual method (GMRES), the global all-to-all communication required in each iteration for orthogonalization and normalization of the Krylov base vectors is becoming a performance bottleneck on massively parallel machines. Long latencies, system noise, and load imbalance cause these global reductions to become very costly global *synchronizations*. In this work, we propose the use of nonblocking or asynchronous global reductions to hide these global communication latencies by overlapping them with other communications and calculations. A *pipelined* variation of GMRES is presented in which the result of a global reduction is used only one or more iterations after the communication phase has started. This way, global synchronization is relaxed and scalability is much improved at the expense of some extra computations. The numerical instabilities that inevitably arise due to the typical monomial basis by powering the matrix are reduced and often annihilated by using Newton or Chebyshev bases instead. Our parallel experiments on a medium-sized cluster show significant speedups of the pipelined solvers compared to standard GMRES. An analytical model is used to extrapolate the performance to future exascale systems.

Key words. GMRES, Gram–Schmidt, parallel computing, latency hiding, global communication

AMS subject classification. 65F10

DOI. 10.1137/12086563X

1. Introduction.

The use of partial differential equations (PDEs) to model the dynamics of a complex system is widespread in scientific practice. Often, discretization of such PDEs results in a very large, sparse linear system, typically solved by a preconditioned Krylov method or by multigrid, either as a standalone multigrid solver or as a preconditioner for a Krylov iteration. Nonstationary iterative Krylov subspace methods, such as GMRES and CG, rely on vector inner products and norms to construct an orthonormal set of Krylov basis vectors. As scalar products require global information, i.e., from every core on every node of the machine, the lower bound on the latency of such an operation is doomed to grow because the maximum parallelism is limited to a tree-like structure with the number of leaves determined by the number of cores. Thus, the minimum number of sequential steps grows with the height of the tree. This can be contrasted with the other operations required in a Krylov method, the most important ones being vector-vector addition (for instance, $\alpha x + y$, called AXPY operation) and the sparse matrix-vector product (SpMV). For PDE discretization matrices, which are typically very sparse, a parallel SpMV only requires communication in some small fixed neighborhood of each node, regardless of

*Submitted to the journal's Software and High-Performance Computing section February 10, 2012; accepted for publication (in revised form) November 14, 2012; published electronically January 8, 2013. This work was funded by Intel and by the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT) and 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 DE-AC02-05CH11231.

http://www.siam.org/journals/sisc/35-1/86563.html

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

‡Intel ExaScience Lab Imec vzw, Kapeldreef 75, B-3001 Leuven, Belgium (ashby@imec.be).

§KU Leuven, Department of Computer Science, Celestijnenlaan 200A, bus 2402, B-3001 Leuven, Belgium (karl.meerbergen@cs.kuleuven.be).

the problem size. (This assumes that neighboring simulation domains map to neighboring processors.) The AXPY is trivially data-parallel. Hence, the scalar products will eventually become the limiting factor for the available parallelism in the algorithm [5]. Furthermore, due to an extreme reduction in scale and voltages, future generations of processors will exhibit both an increase in variability of speed and an increased susceptibility to both transient and permanent failures. This hardware variability, together with system noise and load imbalances, will make global communication an even more costly global *synchronization* step. For a study on the effects of system noise on application scaling see [21] and [4].

One way to avoid expensive global communication is to use simple stationary methods, like Jacobi, (red-black) Gauss–Seidel, or Chebyshev iteration, which do not require global communication, apart from the stopping criterion. However, convergence of these methods is typically quite slow and conditional. Recently, so-called s-step Krylov methods have regained a lot of interest [8, 24, 14, 3, 26, 10, 9, 25]. The main idea in these methods is to create several, s, new Krylov base vectors at once and orthogonalize them together. This reduces the number of global synchronization steps by a factor s. Apart from reducing global communication, this can also improve data locality and cache reuse. However, the maximal s is limited by numerical stability, although switching to a different basis, like Newton or Chebyshev [3, 24], can stretch the limits for s a bit further. Still, in s-step GMRES, computation and global communication phases are performed consecutively, potentially leaving many processors idling during the communication phase.

Instead of trying to reduce the number of global reductions, we present a GMRES variation that hides the reduction latency by overlapping the global communication phase with other useful computations and local communications. The reductions are performed in the background while new Krylov basis vectors are generated. The scalar product values, needed to orthogonalize previous basis vectors, are used only when they become available, which can be after one or possibly several new basis vectors have been created. We call the resulting algorithm p(ℓ)-GMRES, where p stands for "pipelined" and ℓ, for latency, is the number of new (not yet orthogonal) Krylov basis vectors that can be constructed before the result of an inner product becomes available. The total number of floating point operations in the p(ℓ)-GMRES algorithm is slightly larger than in standard GMRES. Therefore, there is a trade-off between improved scalability and computational overhead which highly depends on the hardware specifications. As in s-step GMRES, our p(ℓ)-GMRES algorithm can also use a different basis to improve numerical stability. Section 7 gives a detailed comparison of both s-step and pipelined GMRES. The idea of overlapping global communication with local computations has been applied to the CG method before [12, 17]. However, the idea of pipelining is more general because it allows overlap with both local computations and possibly several phases of local communication.

The major building block of the GMRES method is the Arnoldi process. Since the other steps are usually small in terms of computation time and require no or little communication, we primarily discuss the Arnoldi process. As a result, this work can be used for other applications of Arnoldi's method, such as eigenvalue problems, the matrix exponential (i.e., time stepping), Lyapunov solvers, and model reduction. For the same reason, we do not discuss restarting or any other tricks to reduce the memory cost of GMRES. Within the Arnoldi process, we used classical Gram–Schmidt, in contrast to the usually employed modified Gram–Schmidt in GMRES. We explain this choice in section 2. Reorthogonalization, as required by eigenvalue computations,

and often the solution of hard linear systems and model reduction can be added to classical Gram–Schmidt, as we explain in section 2. In [18], the reorthogonalization step of iterated Gram–Schmidt in an eigenvalue solver is performed asynchronously.

When a sufficiently accurate and locally operating load balancing strategy is used in combination with asynchronous global reductions, the effect of system noise and load imbalance on the parallel scaling will likely be reduced. By using nonblocking communication, the resulting algorithm is much less sensitive to system noise, since not only the link latencies but also the synchronization overhead can be hidden; see also [21]. Apart from hiding scalar product latencies, the proposed p(ℓ)-GMRES variation also relaxes data dependencies, which allows more freedom in scheduling the subtasks of a single GMRES iteration over the available cores; see, for instance, [1] for similar runtime scheduling approaches applied to dense matrix factorizations.

The main contribution of this paper is the formulation of a GMRES version that interleaves the calculation of dot-products with the matrix-vector product and the vector additions. The paper also formulates an easy way to introduce matrix shifts that increase the numerical stability. Various basis sets are validated on benchmark matrices. Also new is a performance model that predicts a more favorable scaling of the algorithms for large systems as a result of the latency hiding.

The paper continues as follows. Section 2 briefly reviews standard GMRES with a focus on the communication pattern. In section 3 an alternative to classical Gram–Schmidt is proposed where the reductions for orthogonalization and normalization are combined. This technique is used in section 4.1 to create a one-step pipelined algorithm in which dot-products are always interleaved with one matrix-vector product. In section 4.2, this is generalized to p(ℓ)-GMRES, which can hide the reduction latency with ℓ iterations. Numerical results for different matrices and different Krylov bases are presented in section 5. In section 6, the parallel performance of the presented algorithms is evaluated on a medium-sized parallel machine. Also, using an analytical model, parallel performance is extrapolated to a hypothetical exascale machine. Next, section 7 discusses preconditioning and gives a detailed comparison with s-step GMRES [24]. An outlook is given in section 8.

2. Standard GMRES. We start with a brief overview of the standard GMRES algorithm shown in Algorithm 1, as originally published by Saad and Schultz [29]. The key components of interest in this paper are the SpMV (line 3), Gram–Schmidt orthogonalization (lines 4 and 5), and normalization (lines 6 and 7). We now discuss each of these components.

Algorithm 1. GMRES.

1: $r_0 \leftarrow b - Ax_0\,; \quad v_0 \leftarrow r_0/||r_0||_2$
2: **for** $i = 0, \ldots, m-1$ **do**
3: $\quad z \leftarrow Av_i$
4: $\quad h_{j,i} \leftarrow \langle z, v_j \rangle\,,\ j = 0, \ldots, i$
5: $\quad \tilde{v}_{i+1} \leftarrow z - \sum_{j=1}^{i} h_{j,i} v_j$
6: $\quad h_{i+1,i} \leftarrow ||\tilde{v}_{i+1}||_2$
7: $\quad v_{i+1} \leftarrow \tilde{v}_{i+1}/h_{i+1,i}$
8: $\quad$ # apply Givens rotations to $H_{:,i}$
9: **end for**
10: $y_m \leftarrow \text{argmin}||(H_{m+1,m} y_m - ||r_0||_2 e_1)||_2$
11: $x \leftarrow x_0 + V_m y_m$

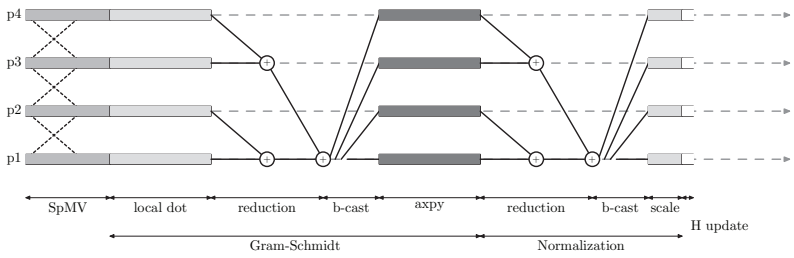


FIG. 2.1. *Schematic representation of a single iteration of the standard GMRES algorithm on four nodes; see Algorithm* 1. *Length of the different phases is not to scale. Communication for the SpMV (dashed lines) is assumed to be among neighbors only, and its latency is completely overlapped by computations for the SpMV.*

The SpMV on line 3 can be a black box function call and can include application of the preconditioner. In this work, the focus will be mostly on preconditioned matrix-vector multiplications which only require communication between nodes that are in a small fixed neighborhood of each other. As discussed in section 6.2, this class of matrices is most interesting from an applications point of view and is targeted by the latency hiding algorithms presented in section 4.

The dot-products in classical Gram–Schmidt on line 4 can be performed in a single global reduction operation, typically using a reduction tree. On a parallel machine this can introduce latency due to global communication. Classical Gram–Schmidt is often replaced by modified Gram–Schmidt for its improved stability. On the other hand, modified Gram–Schmidt requires i global reductions back-to-back when orthogonalizing against i vectors (as is the case in the ith iteration). Hence, in a distributed setting, modified Gram–Schmidt becomes too costly and classical Gram–Schmidt can be used in combination with reorthogonalization when orthogonality of the basis is lost. This is referred to as iterated classical Gram–Schmidt; see [6] and also [11]. In this paper, we restrict ourselves to classical Gram–Schmidt, which works well in practice for many applications, and keep in mind that iterated Gram–Schmidt can be used to improve stability.

The normalization, line 6, also requires a global reduction. In the algorithm, $\tilde{v}_i$ represents the nonnormalized version of v_i. For an actual implementation, the tildes can be dropped, meaning that $\tilde{v}_i$ will use the same memory location as v_i; likewise z can be stored in v_{i+1}.

The least-squares problem in line 10 is typically solved by transforming the upper Hessenberg matrix H to upper triangular form using Givens rotations. These Givens rotations are applied incrementally, one column per iteration. Solving the least-squares problem then only requires a backward substitution, which can be performed after the GMRES iteration has converged. The manipulations of H will not be discussed in detail since the key part of the algorithm is the Arnoldi process.

The different steps of the algorithm in a parallel implementation are discussed in Figure 2.1. This example discusses an implementation with four nodes. A parallel SpMV only requires local communication, which should scale well with the number of nodes and can often be overlapped with local computations. The AXPY operations in line 5 and the scalar-vector multiplication in line 7 do not require communication. The dot-products in the Gram–Schmidt orthogonalization, however, do require global communication, which in this example is done with a binomial reduction tree.

As the number of nodes increases, the global reductions required in lines 4 and 6 may well become the bottleneck [2, 14, 5]. This global communication cannot be

overlapped by computations due to data dependencies. Each GMRES iteration takes at least twice the total minimum latency of a global all-to-all reduce.

3. Avoiding explicit normalization of $\tilde{v}_i$. Instead of first constructing $\tilde{v}_i$, orthogonal to all previous base vectors $v_0, \ldots, v_{i-1}$, and then calculating its norm using another global reduction, this norm can be computed immediately without extra global communication. To improve numerical stability, a shift σ_i will be introduced in the matrix-vector product.

PROPOSITION 3.1. *For $i \geq 0$, let $V_{i+1} := [v_0, v_1, \ldots, v_{i-1}, v_i]$ be an orthonormal basis for the Krylov subspace $\mathcal{K}_{i+1}(A, v_0) = \text{span}\{v_0, Av_0, \ldots, A^i v_0\}$ and let $\sigma_i \in \mathbb{C}$. The following steps expand the basis of the Krylov subspace to V_{i+2} and calculate an additional column of the Hessenberg matrix, $H_{:,i}$, using a single reduction (compared to two reductions for classical Gram–Schmidt):*

1. $z_{i+1} = (A - \sigma_i I)\, v_i$.
2. $\langle z_{i+1}, v_j \rangle$ for $j = 0, \ldots, i$ and $\|z_{i+1}\|$.
3. $h_{j,i} = \langle z_{i+1}, v_j \rangle$ for $j = 0, \ldots, i-1$.
4. $h_{i,i} = \langle z_{i+1}, v_i \rangle + \sigma_i$.
5. $h_{i+1,i} = \sqrt{\|z_{i+1}\|^2 - \sum_{j=0}^{i} \langle z_{i+1}, v_j \rangle^2}$.
6. $v_{i+1} = \left(z_{i+1} - \sum_{j=0}^{i} \langle z_{i+1}, v_j \rangle v_j \right) / h_{i+1,i}$.

Proof. The straightforward way would be to first compute $\tilde{v}_{i+1} = Av_i - \sum_{j=0}^{i} h_{j,i} v_j$, which is orthogonal to all other vectors v_j for $j \leq i$, and then normalize it to obtain $v_{i+1} = \tilde{v}_{i+1}/h_{i+1,i}$. The elements of the Hessenberg matrix are defined as $h_{j,i} = \langle Av_i, v_j \rangle$ for $j \leq i$ and $h_{i+1,i} = \|\tilde{v}_{i+1}\|$. However, this requires an additional latency cost due to the normalization.

Using $z_{i+1} = (A - \sigma_i I)\, v_i$, i.e., line 1 from the proposition,

$$\tilde{v}_{i+1} = z_{i+1} + \sigma_i v_i - \sum_{j=0}^{i} \langle Av_i, v_j \rangle\, v_j = z_{i+1} - \sum_{j=0}^{i-1} \langle Av_i, v_j \rangle\, v_j - (\langle Av_i, v_i \rangle - \sigma_i)\, v_i \tag{3.1}$$

and since for $j < i$, $\langle z_{i+1}, v_j \rangle = \langle Av_i, v_j \rangle = h_{j,i}$ and $\langle z_{i+1}, v_i \rangle = \langle Av_i, v_i \rangle - \sigma_i = h_{i,i}$, this becomes

$$\tilde{v}_{i+1} = z_{i+1} - \sum_{j=0}^{i} \langle z_{i+1}, v_j \rangle v_j \,. \tag{3.2}$$

Now, instead of calculating the norm of $\tilde{v}_{i+1}$ explicitly, it can be found with the help of $\|z_{i+1}\|$, which is also calculated in line 2,

$$h_{i+1,i} = \|\tilde{v}_{i+1}\| = \|z_{i+1} - \sum_{j=0}^{i} \langle z_{i+1}, v_j \rangle v_j\| = \sqrt{\|z_{i+1}\|^2 - \sum_{j=0}^{i} \langle z_{i+1}, v_j \rangle^2} \,. \tag{3.3}$$

This strategy allows us to construct the orthonormal vector v_{i+1} immediately in line 6. Calculating the norm $\|z_{i+1}\|$ (in the same global reduction as the other dot-products) allows us to calculate element $h_{i+1,i}$ of the Hessenberg matrix in line 5. ∎

Remark. When orthogonality of the V_i basis is lost due to rounding errors, a breakdown may occur in line 5 of Proposition 3.1 since the argument of the square root can become negative. If such a *square root breakdown* happens, the GMRES iteration can simply be restarted or a reorthogonalization can be applied. Restarting

GMRES will slow down the convergence but it will also make the Gram–Schmidt procedure cheaper since the Krylov basis is built up from scratch. Also, when $h_{i+1,i}$ evaluates to zero, the algorithm breaks down in line 6. In GMRES, this is typically called a *happy* or *lucky* breakdown since in exact arithmetic $h_{i+1,i} = 0$ if and only if the approximate solution is exact. However, in the GMRES algorithm based on Proposition 3.1, $h_{i+1,i}$ can also become zero due to rounding errors.

Remark. Alternatively, one can use the formula [33]

$$h_{i+1,i} = \|z_{i+1}\| \sin\left(\cos^{-1}\left(\left(\sum_{j=0}^{i} h_{j,i}^2\right)/\|z_{i+1}\|\right)\right), \tag{3.4}$$

which is slightly more accurate than (3.3) but can of course also breakdown. Others have proposed iterative schemes based on Halley's method for computing such Pythagorean sums; see, for instance, [27].

The introduction of the shift σ in the matrix-vector product (line 1) reduces the growth of the vector norms in successive iterations, which makes the algorithm less sensitive to rounding errors in the orthogonalization. This will be further discussed in section 4.3. It is well known that the Krylov space for the shifted matrix is the same as that spanned by the original matrix A. This is called the shift-invariance property.

By replacing lines 3 to 7 in the original GMRES algorithm with the steps from Proposition 3.1, one gets a GMRES variation that requires only a single global reduction per iteration. We shall refer to this algorithm as ℓ^1-GMRES; see Algorithm 2.

ALGORITHM 2. ℓ^1-GMRES.

1: $r_0 \leftarrow b - Ax_0\,; \quad v_0 \leftarrow r_0/||r_0||_2$
2: **for** $i = 0, \ldots, m-1$ **do**
3: $\quad z \leftarrow (A - \sigma_i I)v_i$
4: $\quad h_{j,i} \leftarrow \langle z, v_j \rangle\,,\ j = 1, \ldots, i$
5: $\quad h_{i+1,i} \leftarrow \sqrt{\|z\|^2 - \sum_{j=1}^{i} h_{j,i}^2}$
6: $\quad$ # Check for breakdown and restart or reorthogonalize if necessary
7: $\quad v_{i+1} \leftarrow \left(z - \sum_{j=1}^{i} h_{j,i} v_j\right)/h_{i+1,i}$
8: $\quad h_{i,i} \leftarrow h_{i,i} + \sigma_i$
9: **end for**
10: $y_m \leftarrow \text{argmin}||(H_{m+1,m}y_m - ||r_0||_2 e_1)||_2$
11: $x \leftarrow x_0 + V_m y_m$

4. Pipelined GMRES. In this section a class of pipelined GMRES methods is presented. After introducing the idea with a depth 1 pipelined algorithm, overlapping the reduction with one matrix-vector product, a more general version is derived with an arbitrary pipelining depth. Subsection 4.3 explains how stability can be improved by using a different Krylov basis.

4.1. One-step latency method. A method is presented to loosen the data dependencies that, in standard GMRES, dictate the strict ordering: SpMV, orthogonalization, normalization, SpMV, et cetera. This method will make use of the normalization as presented in Proposition 3.1.

PROPOSITION 4.1. *For $i > 0$ let $V_i := [v_0, v_1, \ldots, v_{i-2}, v_{i-1}]$ be an orthonormal basis for the Krylov subspace $\mathcal{K}_i(A, v_0)$ and let $Z_{i+1} := [z_0, z_1, \ldots, z_{i-1}, z_i]$ be a set of $i+1$ vectors such that for each $j > 0$ it holds that $z_j = (A - \sigma I)\, v_{j-1}$ with $\sigma \in \mathbb{C}$ and*

$z_0 = v_0$. The following steps, for $i > 0$, expand the basis of the Krylov subspace to V_{i+1} (and Z_{i+2}), calculate an additional column of the Hessenberg matrix, and allow simultaneous calculation of the dot-products and the matrix-vector product:

1. *Compute* $\langle z_i, v_j \rangle$ for $j = 0, \ldots, i-1$ and $\|z_i\|$.
2. $w = Az_i$.
3. $h_{j,i-1} = \langle z_i, v_j \rangle$ for $j = 0, \ldots, i-2$.
4. $h_{i-1,i-1} = \langle z_i, v_{i-1} \rangle + \sigma$.
5. $h_{i,i-1} = \sqrt{\|z_i\|^2 - \sum_{j=0}^{i-1} \langle z_i, v_j \rangle^2}$.
6. $v_i = \left(z_i - \sum_{j=0}^{i-1} \langle z_i, v_j \rangle v_j \right) / h_{i,i-1}$.
7. $z_{i+1} = \left(w - \sum_{j=0}^{i-1} h_{j,i-1} z_{j+1} \right) / h_{i,i-1}$.

Proof. In line 7, an additional vector z_{i+1} is constructed, expanding Z_{i+1} to Z_{i+2}. This relation is easily found starting from the Arnoldi relation $v_i = (Av_{i-1} - \sum_{j=0}^{i-1} h_{j,i-1} v_j)/h_{i,i-1}$. Multiplying on the left and right with $(A - \sigma I)$ leads to $z_{i+1} = (Az_i - \sum_{j=0}^{i-1} h_{j,i-1} z_{j+1})/h_{i,i-1}$ with Az_i already calculated in line 2 using the matrix-vector product. Note that the result of line 2 is used for the first time in line 7. The calculation of the dot-products and the matrix-vector are overlapped. Dot-products started in line 1 are first used in line 3.

The other steps are similar to Proposition 3.1, except here the V_{i+1} basis lags one iteration behind the z_{i+1} vector. ∎

4.2. Deeper pipelining: p(ℓ)-GMRES. When dot-product latency is longer than the time to compute an SpMV, the interleaving of dot-products and SpMV as in Proposition 4.1 will not be able to completely hide this latency. Next, Proposition 4.1 is extended such that the algorithm can hide a dot-product latency of up to ℓ iterations, including ℓ SpMVs.

Let $V_{i-\ell+1} = [v_0, v_1, \ldots, v_{i-\ell}]$ be an orthogonal basis for the Krylov subspace $\mathcal{K}_{i-\ell+1}(A, v_0)$. For these vectors the Arnoldi relation $v_j = (Av_{j-1} - \sum_k^{j-1} h_{k,j-1} v_k)/h_{j,j-1}$ holds. Again there are $i+1$ vectors in $Z_{i+1} = [z_0, z_1, \ldots, z_{i-\ell}, z_{i-\ell+1}, \ldots, z_i]$ with $z_0 = v_0$. The index i denotes the number of matrix-vector products necessary to construct the z_i vectors. Note that due to the latency ℓ, the size of the orthonormal Krylov subspace is only $i - \ell + 1$. The z_j and v_j vectors are now related through

$$z_j = \begin{cases} v_0\,, & j = 0\,, \\ P_j(A) v_0\,, & 0 < j \le \ell\,, \\ P_\ell(A) v_{j-\ell}\,, & j > \ell\,, \end{cases} \tag{4.1}$$

with the polynomials $P_i(t)$ defined as

$$P_i(t) = \prod_{j=0}^{i} (t - \sigma_j)\,, \quad i \le \ell, \tag{4.2}$$

where $\sigma_j \in \mathbb{C}$ will be chosen later. The order of the polynomials is limited to ℓ, hence there will only be ℓ different shifts σ_j.

Example 4.2. Take $\ell = 2$ and the shifts σ_0 and σ_1. The polynomials are $P_1(t) = (t - \sigma_0)$ and $P_2(t) = (t - \sigma_1)(t - \sigma_0)$. If the Krylov subspace has size 3 with $V_3 = [v_0, v_1, v_2]$, then the z_i vectors are

(4.3)
$$Z_5 = [v_0,\ (A - \sigma_0 I) v_0,\ (A - \sigma_1 I)(A - \sigma_0 I) v_0,\ (A - \sigma_1 I)(A - \sigma_0 I) v_1,\ (A - \sigma_1 I)(A - \sigma_0 I) v_2]\,.$$

For $j \leq \ell$, successive z_j are related as follows:

$$z_{j+1} = (A - \sigma_j I) z_j \,. \tag{4.4}$$

For $j > \ell$, an Arnoldi-like recurrence relation holds between successive z_j, since fixed polynomials $P_\ell(t)$ of order ℓ are used for $j > \ell$. Indeed, this relation holds between the $v_{j-\ell}$ and it translates to z_j for $j > \ell$ by multiplying the Arnoldi relation by $P_\ell(A)$,

$$P_\ell(A) v_{j-\ell} = \left(P_\ell(A) A v_{j-\ell-1} - \sum_{k=0}^{j-\ell-1} h_{k,j-\ell-1} P_\ell(A) v_k \right) / h_{j-\ell,j-\ell-1} \,, \tag{4.5}$$

$$z_j = \left(A z_{j-1} - \sum_{k=0}^{j-\ell-1} h_{k,j-\ell-1} z_{k+\ell} \right) / h_{j-\ell,j-\ell-1} \,. \tag{4.6}$$

These properties between the successive z_j can be summarized as

$$A Z_i = Z_{i+1} B_{i+1,i} \tag{4.7}$$

with the upper Hessenberg matrix $B_{i+1,i}$, called the *change of basis matrix* [24], given by

$$B_{i+1,i} = \begin{bmatrix} \sigma_0 & & & & & & \\ 1 & \ddots & & & & & \\ & \ddots & \sigma_{\ell-1} & & & & \\ & & 1 & h_{0,0} & \dots & h_{0,i-\ell} \\ & & & h_{1,0} & & \\ & & & & \ddots & \vdots \\ & & & & & h_{i+1-\ell,i-\ell} \end{bmatrix} . \tag{4.8}$$

PROPOSITION 4.3. *Let $\ell < k$ and let V_k be an orthonormal basis for the Krylov subspace $\mathcal{K}_k(A, v_0)$ and Z_k a set of vectors related to V_k as $z_j = P_j(A) v_{j-\ell}$ with polynomials defined in* (4.1). *Then the vectors z and v are related as $Z_j = V_j G_j$ for $j \leq k$ with G_j an upper triangular j by j matrix. Furthermore, the elements of the last column of G_k, i.e., $g_{j,k-1}$ with $j = 0, \dots, k-1$, can be calculated using only the dot-products $\langle z_{k-1}, v_j \rangle$ with $j \leq k-\ell-1$ and $\langle z_{k-1}, z_j \rangle$ with $k-\ell-1 < j \leq k-1$ and the elements of G_{k-1}.*

Proof. Since Z_k and V_k span the same space it is possible to write $Z_k = V_k G_k$. The matrix G is upper triangular since Z_j and V_j span the same space for all $j \leq k$. For $j \leq k-\ell-1$ the elements $g_{j,k-1}$ of the last column of G_k are directly available since the dot-products $g_{j,k-1} = \langle z_{k-1}, v_j \rangle$ are available. For $k-\ell-1 < j \leq k-1$ we have that

$$g_{j,k-1} = \langle z_{k-1}, v_j \rangle = \langle z_{k-1}, \left(z_j - \sum_{m=0}^{j-1} g_{m,j} v_m \right) / g_{j,j} \rangle \tag{4.9}$$

$$= \left(\langle z_{k-1}, z_j \rangle - \sum_{m=0}^{j-1} g_{m,j} g_{m,k-1} \right) / g_{j,j} \,, \tag{4.10}$$

where

$$g_{j,j} = \sqrt{\langle z_j, z_j \rangle - \sum_{m=0}^{j-1} g_{m,j}^2}\,. \tag{4.11}$$

This calculation requires the elements of the smaller G_{k-1}. ∎

Example 4.4. Suppose $\ell = 2$ and $i = 2$. The Krylov subspace is spanned by $[v_0, v_1]$ and there are vectors $Z_4 = [z_0, z_1, z_2, z_3]$. Since $z_0 = v_0$, we have that $g_{0,0} = 1$. The elements of G_2 are then calculated with the help of $\langle z_1, z_1 \rangle$ and $\langle z_1, z_0 \rangle$ as follows. First, we calculate $g_{0,1}$. Since $g_{0,0} = 1$ and the sum under the square root is zero, we find that $g_{0,1} = \langle z_1, z_0 \rangle$. We can now calculate $g_{1,1}$ as $\sqrt{\langle z_1, z_1 \rangle - g_{0,1}^2}$.

Proposition 4.3 makes it easy to extend the basis $V_{i-\ell}$ for the Krylov subspace $\mathcal{K}_{i-\ell}(A, v_0)$, given the set of vectors Z_i; simply apply the proposition with $k = i - \ell$. Indeed, as soon as the dot-products $\langle z_{i-\ell}, v_j \rangle$ are calculated for $j = 0, \ldots, i - 2\ell - 1$ and $\langle z_{i-1}, z_j \rangle$ for $i - 2\ell - 1 < j \leq i - \ell$, the last column of the $G_{i-\ell+1}$ matrix can be calculated. With these matrix elements it is then possible to construct $v_{i-\ell}$, the additional vector that extends $V_{i-\ell}$ to $V_{i-\ell+1}$ as

$$v_{i-\ell} = \left(z_{i-\ell} - \sum_{j=0}^{i-\ell-1} g_{j,i-\ell} v_j \right) / g_{i-\ell,i-\ell}\,. \tag{4.12}$$

To denote a column vector, the row index is replaced by a colon, and the height should be clear from context.

PROPOSITION 4.5. *Let G_k be the upper triangular matrix that relates the basis sets $Z_k = V_k G_k$ and let $B_{k+1,k}$ be the matrix that connects the successive Z_k as $AZ_k = Z_{k+1} B_{k+1,k}$. Then the Hessenberg matrix for the V_k basis can be calculated column by column as*

$$H_{k+1,k} = \begin{bmatrix} H_{k,k-1} & (G_k b_{:,k} + g_{:,k+1} b_{k+1,k} - H_{k,k-1} g_{:,k}) g_{k,k}^{-1} \\ 0 & g_{k+1,k+1} b_{k+1,k} g_{k,k}^{-1} \end{bmatrix}. \tag{4.13}$$

Proof. Combining (4.7) and the Arnoldi recurrence relation with the QR factorization for Z_k leads to

$$H_{k+1,k} = V_{k+1}^T A V_k = V_{k+1}^T A Z_k G_k^{-1} = V_{k+1}^T Z_{k+1} B_{k+1,k} G_k^{-1} \tag{4.14}$$

$$= G_{k+1} B_{k+1,k} G_k^{-1}\,, \tag{4.15}$$

which is a simple expression for the standard GMRES upper Hessenberg coefficient matrix $H_{k+1,k}$. Matrix H can be constructed column by column as follows. From (4.15),

(4.16)

$$H_{k+1,k} = \begin{bmatrix} G_k & g_{:,k+1} \\ 0 & g_{k+1,k+1} \end{bmatrix} \begin{bmatrix} B_{k,k-1} & b_{:,k} \\ 0 & b_{k+1,k} \end{bmatrix} \begin{bmatrix} G_{k-1} & g_{:,k} \\ 0 & g_{k,k} \end{bmatrix}^{-1}$$

$$= \begin{bmatrix} G_k B_{k,k-1} & G_k b_{:,k} + g_{:,k+1} b_{k+1,k} \\ 0 & g_{k+1,k+1} b_{k+1,k} \end{bmatrix} \begin{bmatrix} G_{k-1}^{-1} & -G_{k-1}^{-1} g_{:,k} g_{k,k}^{-1} \\ 0 & g_{k,k}^{-1} \end{bmatrix} \tag{4.17}$$

$$= \begin{bmatrix} G_k B_{k,k-1} G_{k-1}^{-1} & (-G_k B_{k,k-1} G_{k-1}^{-1} g_{:,k} + G_k b_{:,k} + g_{:,k+1} b_{k+1,k}) g_{k,k}^{-1} \\ 0 & g_{k+1,k+1} b_{k+1,k} g_{k,k}^{-1} \end{bmatrix} \tag{4.18}$$

$$= \begin{bmatrix} H_{k,k-1} & (G_k b_{:,k} + g_{:,k+1} b_{k+1,k} - H_{k,k-1} g_{:,k}) g_{k,k}^{-1} \\ 0 & g_{k+1,k+1} b_{k+1,k} g_{k,k}^{-1} \end{bmatrix}. \tag{4.19}$$

∎

Once this Hessenberg matrix is calculated it is easy to extend the set of Z_{i+1} vectors to Z_{i+2} by adding

$$z_{i+1} = \begin{cases} (A - \sigma_i I) z_i, & i < \ell\,, \\ \left(A z_i - \sum_{j=0}^{i-\ell} h_{j,i-\ell} z_{j+\ell} \right) / h_{i-\ell+1,i-\ell}, & i \geq \ell\,, \end{cases} \tag{4.20}$$

a relation that is based on (4.4) and (4.6). Note that the shifts are only explicitly used in the first ℓ iterations.

The above formulae directly translate to Algorithm 3. As the pipelining has introduced several potential sources of instability, the resulting algorithm has different numerical stability properties compared to classical GMRES. As in ℓ^1-GMRES (see the remark in section 3), the square root in line 7 can lead to a *square root breakdown*. The easiest solution in this case is to restart the GMRES algorithm. However, restarting slows down the convergence and, since the pipeline has to be filled again, parallel efficiency declines. When instead of restarting, a reorthogonalization step is performed, convergence is affected less. Furthermore, to maintain good convergence and hence avoid square root breakdown, proper choices for the shifts σ_i are crucial; see section 4.3. In section 5, convergence results are shown for several matrices and different shifts.

ALGORITHM 3. p(ℓ)-GMRES.

1: $r_0 \leftarrow b - Ax_0\,; \quad v_0 \leftarrow r_0 / ||r_0||\,; \quad z_0 \leftarrow v_0$
2: **for** $i = 0, \ldots, m + \ell$ **do**
3: $\quad z_{i+1} \leftarrow \begin{cases} (A - \sigma_i I) z_i, & i < \ell \\ A z_i, & i \geq \ell \end{cases}$
4: $\quad a \leftarrow i - \ell$
5: $\quad$ **if** $a \geq 0$ **then**
6: $\quad\quad g_{j,a+1} \leftarrow (g_{j,a+1} - \sum_{k=0}^{j-1} g_{k,j} g_{k,a+1}) / g_{j,j},\ j = a - \ell + 2, \ldots, a$
7: $\quad\quad g_{a+1,a+1} \leftarrow \sqrt{g_{a+1,a+1} - \sum_{k=0}^{a} g_{k,a+1}^2}$
8: $\quad\quad$ # Check for breakdown and restart or reorthogonalize if necessary
9: $\quad\quad$ **if** $a < \ell$ **then**
10: $\quad\quad\quad h_{j,a} \leftarrow (g_{j,a+1} + \sigma_a g_{j,a} - \sum_{k=0}^{a-1} h_{j,k} g_{k,a}) / g_{a,a},\ j = 0, \ldots, a$
11: $\quad\quad\quad h_{a+1,a} \leftarrow g_{a+1,a+1} / g_{a,a}$
12: $\quad\quad$ **else**
13: $\quad\quad\quad h_{j,a} \leftarrow (\sum_{k=0}^{a+1-\ell} g_{j,k+\ell} h_{k,a-\ell} - \sum_{k=j-1}^{a-1} h_{j,k} g_{k,a}) / g_{a,a},\ j = 0, \ldots, a$
14: $\quad\quad\quad h_{a+1,a} \leftarrow g_{a+1,a+1} h_{a+1-\ell,a-\ell} / g_{a,a}$
15: $\quad\quad$ **end if**
16: $\quad\quad v_a \leftarrow (z_a - \sum_{j=0}^{a-1} g_{j,a} v_j) / g_{a,a}$
17: $\quad\quad z_{i+1} \leftarrow (z_{i+1} - \sum_{j=0}^{a-1} h_{j,a-1} z_{j+\ell}) / h_{a,a-1}$
18: $\quad$ **end if**
19: $\quad g_{j,i+1} \leftarrow \begin{cases} \langle z_{i+1}, v_j \rangle\,, & j = 0, \ldots, a \\ \langle z_{i+1}, z_j \rangle\,, & j = a+1, \ldots, i+1 \end{cases}$
20: **end for**
21: $y_m \leftarrow \operatorname{argmin} ||(H_{m+1,m} y_m - ||r_0|| e_1)||_2$
22: $x \leftarrow x_0 + V_m y_m$

Remark. Since in the p(ℓ)-GMRES algorithm, two sets of basis vectors $V_{i-\ell+1}$ and Z_{i+1} are stored, the memory requirements are doubled compared to standard

GMRES. However, it is possible to rewrite the vectors in $Z_{i-\ell+1}$ as linear combinations of vectors in $V_{i-\ell+1}$. In this case, only the last ℓ vectors of Z_{i+1} have to be stored explicitly. Furthermore, the first ℓ vectors in Z_{i+1} are no longer used when $i > 2\ell$, so they do not have to be stored any longer. A possible advantage of explicitly storing the Z_{i+1} basis could be that in that case, the GMRES method can be made flexible, meaning that the preconditioner is allowed to change from iteration to iteration; see [30].

4.3. Choosing the basis. Since the problem of determining good values for the shifts σ_i in p(ℓ)-GMRES is analogous to finding a good basis in s-step GMRES, we shall refer the reader to [24] for an excellent overview.

It is well known that the Krylov basis $\mathcal{K}_{i+1}(v_0, A) = [v_0, Av_0, \ldots, A^i v_0]$, called the monomial basis, can become ill-conditioned very quickly. As in the power method, for suitable starting vectors and certain condition on A, $A^i v_0$ converges to the principal eigenvector of the matrix. The basis condition number increases exponentially and eventually the basis can become numerically rank deficient or can overflow. These problems can be reduced significantly by choosing a different basis as $\mathcal{K}_{i+1}(v_0, A) = [v_0, P_1(A)v_0, \ldots, P_i(A)v_0]$. Similarly, when using a monomial basis in the p(ℓ)-GMRES algorithm, the $z_i = P_\ell(A)v_{j-\ell} = A^\ell v_{j-\ell}$ vectors also tend to converge to the principal eigenvector as ℓ increases. In this case, numerical stability can also be improved by proper choices of $P_\ell(A)$.

Bai, Hu, and Reichel [3] were the first to recognize the analogy with polynomial interpolation and to apply a Newton basis in s-step GMRES. For the Newton basis, we take $P_i(A) = \prod_{j=0}^{i} (A - \sigma_j I)$, where natural choices for the *shifts*[1] σ_j are the Ritz values of the matrix A. As p(ℓ)-GMRES requires ℓ shifts, this requires ℓ iterations of Arnoldi/GMRES. With the Leja ordering [28, 3], the shifts are ordered such that the distance between a shift and all previous shifts is maximized in some sense. Intuitively, it is clear that this ordering will also improve stability since two consecutive nearly identical shifts correspond to two iterations with the monomial basis for the shifted matrix.

In case all eigenvalues are known a priori to lie in a certain region of the complex plane, a polynomial $P_\ell(A)$ can be constructed which is minimal over the given region. For an ellipsoidal region with foci $d \pm c$, these are the complex, scaled, and shifted Chebyshev polynomials $\tilde{T}_i(t)$. Let the classical Chebyshev polynomials be defined as

$$T_0(t) = 1, \quad T_1(t) = t, \quad T_{i+1}(t) = 2tT_i(t) - T_{i-1}(t); \tag{4.21}$$

then the scaled and shifted Chebyshev polynomials are easily constructed using the three-term recurrence relation

$$\tilde{T}_0 = 1, \quad \tilde{T}_1(t) = 1 - \frac{t}{d}, \quad \tilde{T}_{i+1}(t) = \frac{\sigma_i}{\sigma_{i+1}} \left(2\frac{d-t}{c}\tilde{T}_i(t) - \frac{\sigma_{i-1}}{\sigma_i}\tilde{T}_{i-1}(t) \right), \tag{4.22}$$

where $\sigma_i = T_i(d/c)$. When this recurrence relation is used to construct new Krylov base vectors as $z_{i+1} = \sigma_i/\sigma_{i+1}[2/c(d-A)z_i - (\sigma_{i-1}/\sigma_i)z_{i-1}]$, the structure of the basis matrix (equation (4.8)) changes: its upper left $(\ell+1) \times \ell$ part becomes tridiagonal rather than lower bidiagonal [24]. However, this still affects only the first ℓ iterations. When all eigenvalues are real and lie in an interval $[\lambda_{\min}, \lambda_{\max}]$, not containing the origin, the zeros of the scaled and shifted Chebyshev polynomial of degree ℓ, which is

[1] Called interpolation nodes in the context of polynomial interpolation.

minimal over this interval, are given by

$$\sigma_i = \frac{\lambda_{\min} + \lambda_{\max}}{2} + \left(\frac{\lambda_{\max} - \lambda_{\min}}{2}\right) \cos\left(\frac{(2i+1)\pi}{2\ell}\right) \quad \text{for } i = 0, \ldots, \ell - 1\,, \tag{4.23}$$

and these σ_i (permuted to Leja ordering) can be used as shifts in the Newton basis, just as the Ritz values can be used as shifts.

For real matrices and a real starting vector, eigenvalues and Ritz values with nonzero imaginary part always come in complex conjugate pairs, say, λ_j and $\lambda_{j+1} = \bar{\lambda}_j$. In this case, complex arithmetic can be avoided by adding base vectors as [3, 24]

$$z_{j+1} = (A - \mathcal{R}(\lambda_j) I)\, z_j\,, \tag{4.24}$$

$$z_{j+2} = (A - \mathcal{R}(\lambda_j) I)\, z_{j+1} + \mathcal{I}(\lambda_j)^2 z_j \quad \left(\equiv (A - \bar{\lambda}_j I)(A - \lambda_j I) z_j\right)\,. \tag{4.25}$$

This trick can be combined with the *modified* version of the Leja ordering [3, 24] to prevent the complex conjugate pairs from getting separated.

4.4. Variation: Explicit normalization. In this section we present a one-step pipelined GMRES algorithm, which differs from p(1)-GMRES as presented above in the way the normalization is performed. In p(1)-GMRES, the norm of $\tilde{v}$ was computed using the trick presented in section 3. However, this way of computing $\|\tilde{v}\|$ is less accurate due to the subtraction in (4.11) and introduces potential square root breakdown. Another possibility is to explicitly compute the norm after orthogonalization, exactly like in standard GMRES. It is possible to combine the reduction for normalization with the reduction for orthogonalization of the next iteration. However, this doubles the delay, meaning that the orthonormal basis V and the Hessenberg matrix are two iterations behind on the construction of the Z basis, opposed to just 1 in p(1)-GMRES.

PROPOSITION 4.6. *For $i > 0$ let $V_i := [v_0, v_1, \ldots, v_{i-2}, v_{i-2}, \tilde{v}_{i-1}]$ be an orthogonal basis for the Krylov subspace $\mathcal{K}_i(A, v_0)$ with $\|v_j\| = 1$ for $j = 0, \ldots, i-2$. Furthermore, let $Z_{i+1} := [z_0, z_1, \ldots, z_{i-1}, \tilde{z}_i]$ be a set of $i+1$ vectors with $z_0 = v_0$ and such that $z_j = Av_{j-1}$ for $j > 0$. Furthermore, let $\tilde{z}_i = A\tilde{v}_{i-1}$. The following steps, for $i > 0$, expand the basis of the Krylov subspace to V_{i+1} (and Z_{i+2}), calculate an additional column of the Hessenberg matrix, and overlap the calculation of dot-products and the matrix-vector product:*

1. *Compute* $\|\tilde{v}_{i-1}\|$ and $\langle \tilde{z}_i, \tilde{v}_{i-1}\rangle$ and $\langle \tilde{z}_i, v_j\rangle$ for $j = 0, \ldots, i-2$.
2. $\tilde{w} = A\tilde{z}_i$.
3. $h_{i-1,i-2} = \|\tilde{v}_{i-1}\|$.
4. $v_{i-1} = \tilde{v}_{i-1}/h_{i-1,i-2}$.
5. $z_i = \tilde{z}_i/h_{i-1,i-2}$.
6. $w = \tilde{w}/h_{i-1,i-2}$.
7. $h_{j,i-1} = \langle \tilde{z}_i, v_j\rangle/h_{i-1,i-2}$ for $j = 0, \ldots, i-2$.
8. $h_{i-1,i-1} = \langle \tilde{z}_i, \tilde{v}_{i-1}\rangle/h_{i-1,i-2}^2$.
9. $\tilde{z}_{i+1} = w - \sum_{j=0}^{i-1} h_{j,i-1} z_{j+1}$.
10. $\tilde{v}_i = z_i - \sum_{j=0}^{i-1} h_{j,i-1} v_j$.

Proof. First, $\tilde{v}_{i-1}$ is normalized to v_{i-1} in lines 3 and 4. Then, note that $w = Az_i$, since

$$w = \frac{\tilde{w}}{h_{i-1,i-2}} = \frac{A\tilde{z}_i}{\|\tilde{v}_{i-1}\|} = \frac{A^2\tilde{v}_{i-1}}{\|\tilde{v}_{i-1}\|} = A^2 v_{i-1} = Az_i\,, \tag{4.26}$$

and $z_i = Av_{i-1}$, since $z_i = \tilde{z}_i/\|\tilde{v}_{i-1}\| = A\tilde{v}_{i-1}/\|\tilde{v}_{i-1}\| = Av_{i-1}$. The Hessenberg elements are computed as

$$h_{j,i-1} = \langle Av_{i-1}, v_j\rangle = \langle z_i, v_j\rangle = \frac{\langle \tilde{z}_i, v_j\rangle}{h_{i-1,i-2}} \text{ for } j = 0, \ldots, i-2\,, \tag{4.27}$$

$$h_{i-1,i-1} = \langle Av_{i-1}, v_{i-1}\rangle = \frac{\langle z_i, \tilde{v}_{i-1}\rangle}{h_{i-1,i-2}} = \frac{\langle \tilde{z}_i, \tilde{v}_{i-1}\rangle}{h^2_{i-1,i-2}}\,. \tag{4.28}$$

The vector $\tilde{v}_i = z_i - \sum_{j=0}^{i-1} h_{j,i-1}v_j = Av_{i-1} - \sum_{j=0}^{i-1} h_{j,i-1}v_j$ is added to the orthogonal basis V_i just as in standard GMRES. Multiplying by A leads again to $\tilde{z}_{i+1} = w - \sum_{j=0}^{i-1} h_{j,i-1}z_{j+1}$.

All dot-products and calculation of the norm of $\tilde{v}_{i-1}$ are started in line 1 and they can be combined in a single global reduction. The SpMV is computed in line 2 and the results of the reduction in line 1 are first used in line 3. ∎

Note that it should be possible to generalize Proposition 4.6 to include a shift in the matrix-vector product as well as to deeper pipelining depths, but we have not explored this further.

Algorithm 4 shows the p^1-GMRES method, which is based on the steps from Proposition 4.6. In this code, the superimposed tildes on $\tilde{z}_i$, $\tilde{v}_i$, $\tilde{w}$, and $\tilde{h}$ are already dropped since this simplifies the startup phase of the algorithm. Actual implementations will also overwrite $\tilde{v}_i$ with v_i, et cetera. Algorithm 4 has been implemented in the PETSc library (from release 3.3) as KSPPGMRES.[2]

ALGORITHM 4. p^1-GMRES.

1: $r_0 \leftarrow b - Ax_0\,; \quad v_0 \leftarrow r_0/||r_0||_2\,; \quad z_0 \leftarrow v_0$
2: **for** $i = 0, \ldots, m+1$ **do**
3: $\quad w \leftarrow Az_i$
4: $\quad$ **if** $i > 1$ **then**
5: $\quad\quad v_{i-1} \leftarrow v_{i-1}/h_{i-1,i-2}$
6: $\quad\quad z_i \leftarrow z_i/h_{i-1,i-2}$
7: $\quad\quad w \leftarrow w/h_{i-1,i-2}$
8: $\quad\quad h_{j,i-1} \leftarrow h_{j,i-1}/h_{i-1,i-2},\ j = 0, \ldots, i-2$
9: $\quad\quad h_{i-1,i-1} \leftarrow h_{i-1,i-1}/h^2_{i-1,i-2}$
10: $\quad$ **end if**
11: $\quad z_{i+1} \leftarrow w - \sum_{j=0}^{i-1} h_{j,i-1}z_{j+1}$
12: $\quad$ **if** $i > 0$ **then**
13: $\quad\quad v_i \leftarrow z_i - \sum_{j=0}^{i-1} h_{j,i-1}v_j$
14: $\quad\quad h_{i,i-1} \leftarrow ||v_i||_2$
15: $\quad$ **end if**
16: $\quad h_{j,i} \leftarrow \langle z_{i+1}, v_j\rangle\,,\ j = 0, \ldots, i$
17: **end for**
18: $y_m \leftarrow \text{argmin}||(H_{m+1,m}y_m - ||r_0||_2 e_1)||_2$
19: $x \leftarrow x_0 + V_m y_m$

Figure 4.1 shows the p^1-GMRES iteration schematically. Again, the phases are not to scale. The figure shows the duration of a single iteration. However, due to the pipelining, several iterations are fused. In this figure, the matrix-vector product does not take long enough to overlap the complete reduction and broadcast step.

[2] http://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/KSP/KSPPGMRES.html.

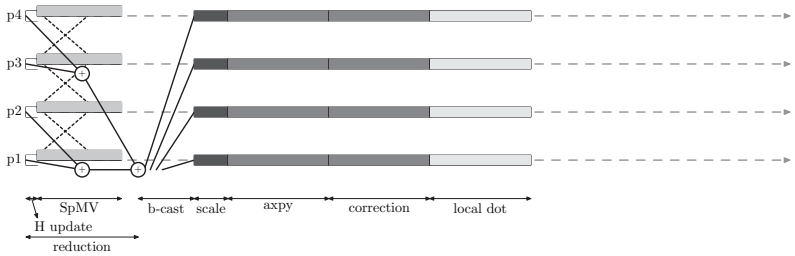


FIG. 4.1. *Schematic representation of an iteration of p^1-GMRES on four nodes. Note that the reduction follows on the* local dot *phase from the previous iteration. As in Figure* 2.1*, local communication for the SpMV is also overlapped with computations. The redundant computations are represented by the* correction *block. Note that only a single global reduction is needed per iteration.*

5. Numerical results. In this section, we present convergence results for the presented algorithms applied on benchmark matrices. We give results for an artificial test problem as well as for two matrices which are more relevant for applications. The right-hand-side vector was always a random vector with elements uniformly distributed between 0 and 1.

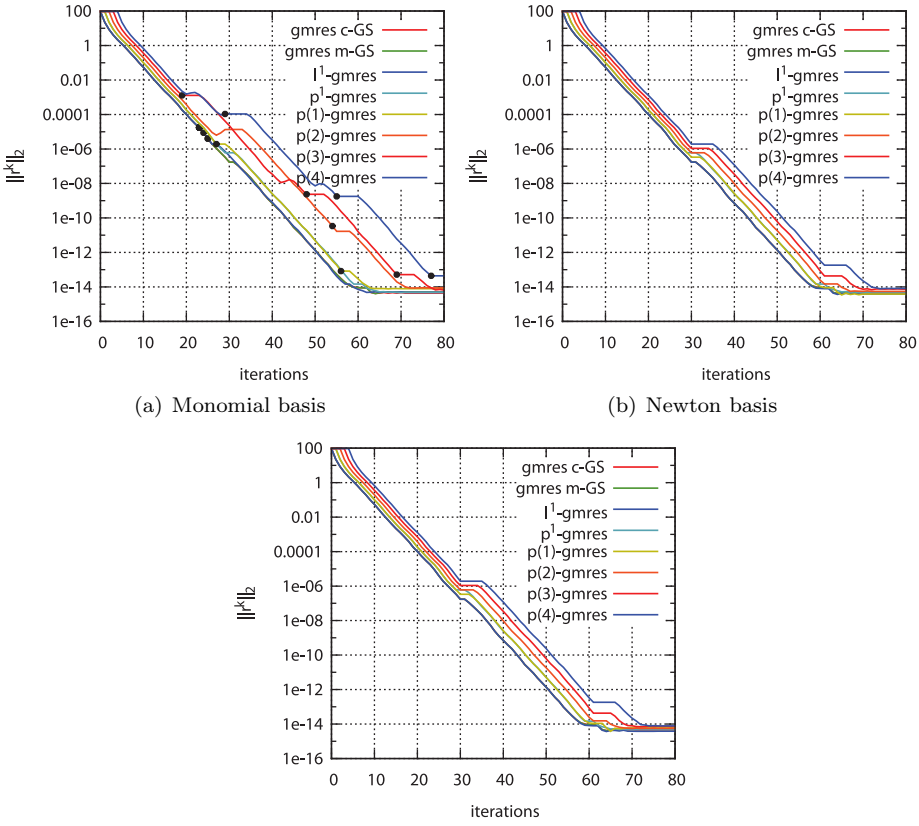
The first test matrix is an artificial example. We take a lower bidiagonal matrix $A \in \mathbb{R}^{500\times 500}$ with elements $A_{i,i} = 1 + p_i$ on the diagonal and $A_{i+1,i} = q_i$ on the subdiagonal, where p_i and q_i are random numbers uniformly distributed on $[0, 1]$. The matrix has a quite low condition number, $\kappa_2(A) \approx 5$, is unsymmetric, nonnegative, not normal, positive definite, and diagonally dominant. The eigenvalues are randomly distributed between 1 and 2.

Figure 5.1 compares convergence of standard GMRES, both with classical and modified Gram–Schmidt, with the pipelined solvers presented earlier for $\ell = 1, 2, 3, 4$ as well as with the variation p^1-GMRES and with ℓ^1-GMRES. The residual norm is computed explicitly rather than via the recurrence relation. A restart length of 30 is used. Figure 5.1, left, uses the standard monomial basis, i.e., all shifts zero. Note that for longer pipelining depths, square root breakdown, denoted by the black dots, occurs earlier and more frequently, hence slowing down convergence. In Figure 5.1, right, the pipelined solvers use a Newton basis with the ℓ Ritz values in Leja ordering as shifts. For ℓ^1-GMRES, the first Ritz value is used as a shift in all iterations. These Ritz values are computed from ℓ standard GMRES iterations before starting the pipelined iteration, which then rebuilds the Krylov basis from scratch. The shifts are not updated when the pipelined solver is restarted. Convergence is nearly identical to that of standard GMRES, except for a small delay. In Figure 5.1, bottom, the Newton basis is used with the zeros of the ℓth order scaled and shifted (to $[1, 2]$) Chebyshev polynomial as shifts, again in Leja ordering. Convergence is similar to standard GMRES.

The delay observed for the pipelined solvers in Figure 5.1 is caused by the dot-product latency, which is hidden by the pipelining. After a restart, at 30 and 60, the pipeline has to be filled again, resulting is a new delay of ℓ steps.

5.1. Matrices from applications. Figure 5.2 shows convergence results for the matrix pde900[3] from the Matrix Market collection of sparse matrices. This is a real unsymmetric 900×900 matrix based on a five point central difference discretization of a linear elliptic equation with Dirichlet boundary conditions on a regular 30×30 grid. The condition number is $\kappa(A) \approx 2.9 \cdot 10^2$. All GMRES methods are restarted

[3] http://math.nist.gov/MatrixMarket/data/NEP/matpde/pde900.html.



(a) Monomial basis

(b) Newton basis

(c) Chebyshev basis

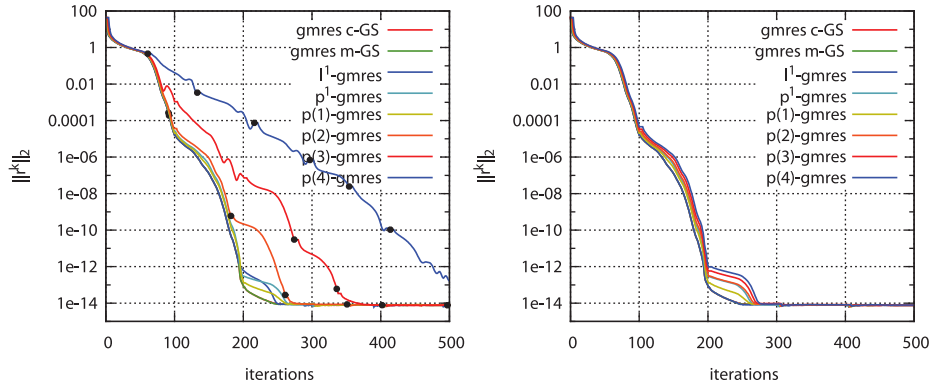
FIG. 5.1. *Bidiagonal matrix with eigenvalues uniformly distributed between* 1 *and* 2. *Comparison between standard (both with classical and modified Gram–Schmidt), pipelined GMRES, and* ℓ^1*-GMRES. Each solver uses a restart length* 30. *Black dots denote restart points due to square root breakdown. Left: Monomial basis, i.e., all shifts zero. Right: Newton basis using* ℓ *Ritz values as shifts. Bottom: Newton basis with the shifts chosen as the zeros of the scaled and shifted Chebyshev polynomials of degree* ℓ.

after 100 iterations. In Figure 5.2, left, the monomial basis is used. The number of square root breakdowns increases with increasing pipelining depth and convergence slows down correspondingly. Figure 5.2, right, reports convergence using the Newton basis with ℓ Ritz values as shifts.

The **orsirr-1**[4] matrix, from Matrix Market, is generated from an oil reservoir simulation. It is a real unsymmetric 1030×1030 matrix with 6858 nonzero entries and condition number $\kappa(A) \approx 100$. Figure 5.3 show convergence using the monomial basis (left) and the Newton basis with Ritz values as shifts (right). All GMRES versions are restarted after 40 iterations. Here, the pipelined solver, with $\ell = 3$ and $\ell = 4$, requires many restarts due to square root breakdown; see the black dots in the figure. With the Newton basis, using ℓ Ritz values as shifts, convergence is slightly improved and is more or less in line with that of the standard GMRES solver.

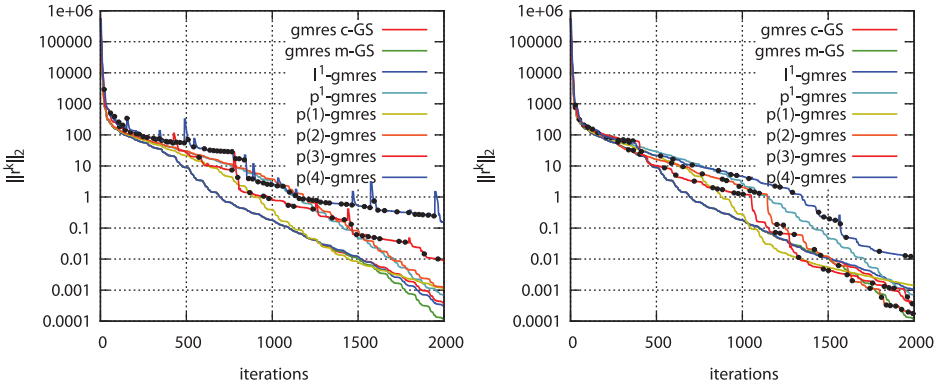
Apart from the numerical results shown in this section, we have performed tests on a variety of different matrices. We believe that the results reported here are

[4] http://math.nist.gov/MatrixMarket/data/Harwell-Boeing/oilgen/orsirr_1.html.



(a) Monomial basis (b) Newton basis

FIG. 5.2. *Convergence results for the different GMRES versions applied to the pde*900 *test matrix. The GMRES methods are restarted after* 100 *iterations. As the pipelining depth increases, square root breakdowns (black dots) occur more frequently and convergence slows down correspondingly. Left: Using the monomial basis, i.e., all shifts zero. Right: Using* ℓ *Ritz values as shifts.*



(a) Monomial basis (b) Newton basis

FIG. 5.3. *Convergence results for the different GMRES versions applied to the orsirr*-1 *test matrix. All methods are restarted after* 40 *iterations, or when a square root breakdown occurs, illustrated by the black dots. Left: Using the monomial basis, i.e., all shifts zero. Right: Using* ℓ *Ritz values as shifts.*

representative for the general behavior of the pipelined solvers. Generally, the Newton or Chebyshev bases greatly improve numerical stability. However, in some cases the monomial basis outperforms the Newton basis, as also observed for two-sided s-step Krylov methods [7].

6. Performance evaluation. This section presents runtime and scalability results for the different GMRES variations for a two-dimensional (2D) Poisson problem on a medium scale parallel machine. Next, an extrapolation to a hypothetical exascale machine is presented using an analytical performance model.

6.1. Timings on Carver. The parallel performance of the different GMRES variations is evaluated on an IBM iDataPlex machine from NERSC.[5] The full

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

system has 1,202 compute nodes (9,984 processor cores) with a theoretical peak performance of 106 Tflops/sec. All nodes are interconnected by 4×QDR InfiniBand technology, providing 32 Gb/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 2.67 GHz processors (eight cores/node) and since parallel jobs on Carver are limited to 64 nodes, only these Nehalem nodes were used. The problem being solved is the 2D Poisson equation discretized on a regular grid ($N = 1024^2$) using a standard five-point finite difference stencil. The algorithms are implemented in C++ using the message passing interface (MPI).

The asynchronous communication of the local boundary and the ghost points is started using nonblocking `MPI_Isend` and `MPI_Irecv`, respectively. This communication can be overlapped with computation on the locally interior points. Afterward, a call to `MPI_Waitall` blocks until communication for the boundary is finished. Finally, the local boundary points may be computed. Communication for the matrix-vector product is overlapped with stencil computations in this manner. Nonblocking versions of collective operations, such as nonblocking reductions, broadcasts or barriers, are not included in MPI-1/2 but are proposed for the upcoming MPI-3 standard. Alternatively, libNBC [19, 23, 20] implements nonblocking collectives on top of (standard MPI-1/2) nonblocking point-to-point communication. To maximize the potential overlap with the nonblocking reduction, a progress thread should be used. Our tests are performed using MPICH2 version 1.5rc1,[6] configured to use the nemesis channel with TCP and IP-over-Infiniband (IPoIB) network interfaces. When a progress thread is used, one should configure MPICH2 with `--enable-threads=runtime`, set the environment variable `MPICH_ASYNC_PROGRESS=1`, and initialize the MPI library with `MPI_Init_thread(.., .., MPI_THREAD_MULTIPLE, ..)`. A nonblocking reduction can then be performed by calling `MPI_Iallreduce` in combination with a matching `MPI_Wait`.

Figure 6.1, left, shows the average runtime per iteration for the different GMRES methods on Carver. The runtime is an average over 10 iterations; this is repeated 30 times and from this the minimum runtime is taken. Figure 6.1, right, shows for the same experiment the speedup compared to the run on a single node. For the pipelined solvers, filling and draining of the pipeline are included in the timing. For instance, p(ℓ)-GMRES actually requires $10 + \ell$ iterations to achieve the same reduction in residual as standard GMRES achieves in 10 iterations. However, for p(ℓ)-GMRES, the first ℓ iterations do not perform orthogonalization, whereas the last ℓ iterations do not apply the matrix-vector product. On a single node, the ℓ^1 and standard GMRES methods perform best, but on a larger number of nodes the pipelined methods outperform the classical ones due to the improved scalability. We refer to Table 6.1 for timings and for the speedup compared to standard GMRES with classical Gram–Schmidt (cGS GMRES). On a single node, pipelining does not pay off. On 256 cores (using 32) nodes, p(4)-GMRES already achieves a 1.45× speedup compared to cGS GMRES, which goes up to 1.84× on 512 cores (64 nodes).

6.2. Extrapolation to exascale. In this section, an analytical model for the parallel execution time of the presented algorithms is used to predict their parallel performance and scalability properties on future hardware. As in [16], we assume that

[6] http://www.mcs.anl.gov/research/projects/mpich2/. When a progress thread is used, the MPI implementation should be thread safe, which turned out to be problematic for most of the currently available MPI implementations. We found MPICH2 has good multithreading support, but unfortunately it has no Infiniband support.

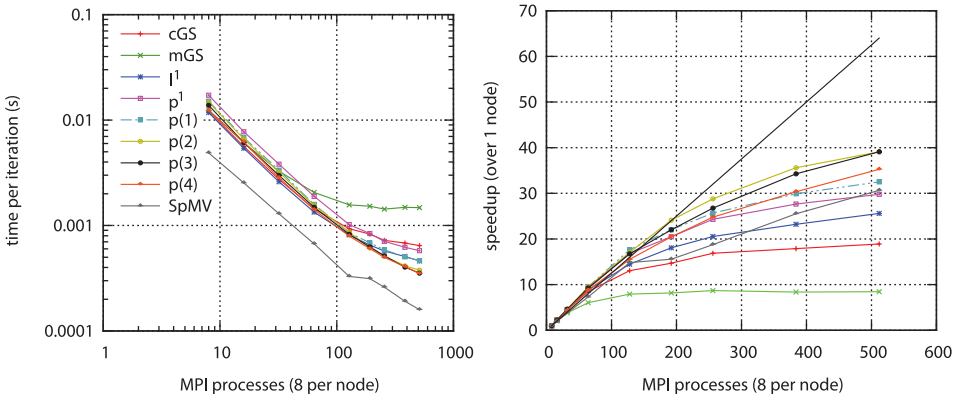


FIG. 6.1. *Average runtime per iteration (left) and speedup over a single node (right) for a 2D Poisson problem on Carver (*IBM *i*DATAPLEX *from* NERSC*) for the different GMRES variations described in the text. On a single node, the ℓ^1 and standard GMRES methods perform best, but on larger number of nodes the pipelined methods outperform the classical ones due to the improved scalability.*

TABLE 6.1

Results for different GMRES variations applied to the 2D Poisson problem on Carver. The first number is runtime per iteration in seconds, the next is speedup compared to standard GMRES with classical Gram–Schmidt (cGS). Different columns give timings for different numbers of nodes with different numbers of processes per node. For instance, 64×8 denotes a run on 64 nodes with 8 MPI processes per node.

method	1×1	1×8	32×8	64×8
cGS	7.23e-2 / 1	1.21e-2 / 1	7.22e-4 / 1	6.45e-4 / 1
mGS	8.59e-2 / .84	1.25e-2 / .98	1.43e-3 / .50	1.48e-3 / 0.44
ℓ^1	7.00e-2 / 1.03	1.18e-2 / 1.03	5.73e-4 / 1.26	4.61e-4 / 1.40
p^1	9.67e-2 / .75	1.72e-2 / .71	7.07e-4 / 1.02	5.44e-4 / 1.12
p(1)	8.40e-2 / .86	1.51e-2 / .81	5.88e-4 / 1.23	4.64e-4 / 1.39
p(2)	7.80e-2 / .93	1.48e-2 / .82	5.12e-4 / 1.41	3.77e-4 / 1.71
p(3)	7.67e-2 / .94	1.38e-2 / .88	5.16e-4 / 1.40	3.53e-4 / 1.83
p(4)	7.51e-2 / .96	1.23e-2 / .99	4.98e-4 / 1.45	3.50e-4 / 1.84

it takes roughly $t_s + mt_w$ time for a simple exchange of an m-word message between two processes running on different nodes. Here t_s is the latency or startup time for the data transfer and t_w is the per-word transfer time, which is inversely proportional to the available bandwidth between the nodes. The parallel execution time T_P of a single GMRES iteration on a parallel machine is modeled as $T_P = T_{\text{spmv}} + T_{\text{red}} + T_{\text{calc}}$, with T_{spmv} the time per processor spent in the matrix-vector product, T_{calc} the time spent in local calculations for a single iteration (excluding the matrix-vector product), and T_{red} the global communication (reduction) overhead per iteration per processor. The local calculation time, T_{calc}, is found by counting all floating point operations[7] in an iteration and multiplying by t_c, the floating point performance of the machine (in seconds per flop). We will assume the machine reaches 1 exaflop, i.e., 10^{18} floating point operations per second.

Assuming the system matrix corresponds to a stencil on a regular three-dimensional (3D) grid, it will have about $n_z = 7$ nonzero elements per row, and hence

[7]This ignores the fact that on many modern architectures a scalar AXPY operation can be done in a single floating point instruction, a so-called fused multiply add.

TABLE 6.2

Cost functions for the ith iteration of the different GMRES (or Arnoldi) formulations, only counting work on the vectors, i.e., ignoring the manipulations on H and G.

GMRES cGS	T_{calc}	$t_c((4i+3)N/P_c + (i+1)\lceil \log_r(P_n)\rceil)$
	T_{red}	$2\lceil \log_r(P_n)\rceil(t_s + it_w) + 2\lceil \log_r(P_n)\rceil(t_s + t_w)$
GMRES mGS	T_{calc}	$t_c((4i+3)N/P_c + (i+1)\lceil \log_r(P_n)\rceil)$
	T_{red}	$(2i+2)\lceil \log_r(P_n)\rceil(t_s + t_w)$
ℓ^1-GMRES	T_{calc}	$t_c((4i+4)N/P_c + (i+1)\lceil \log_r(P_n)\rceil)$
	T_{red}	$2\lceil \log_r(P_n)\rceil(t_s + (i+1)t_w)$
p^1-GMRES	T_{calc}	$t_c((6i+1)N/P_c + (i+1)\lceil \log_r(P_n)\rceil)$
	T_{red}	$2\lceil \log_r(P_n)\rceil(t_s + it_w) - \min(3t_cN/P_c + T_{\text{spmv}}, 2\lceil \log_r(P_n)\rceil(t_s + it_w))$
p(ℓ)-GMRES	T_{calc}	$t_c((6i - 4\ell + 4)N/P_c + (i+1)\lceil \log_r(P_n)\rceil)$
	T_{red}	$2\lceil \log_r(P_n)\rceil(t_s + (i+1)t_w) - \min(4\ell(i-\ell)t_cN/P_c + \ell T_{\text{spmv}}, 2\lceil \log_r(P_n)\rceil(t_s + (i+1)t_w))$

the matrix-vector product requires $2n_z = 14$ flops per element in the vector. With a regular 3D division of the domain over all nodes, neighboring nodes have to communicate $(N/P_n)^{2/3}$ doubles during the SpMV with N the total problem size and P_n the number of nodes. This gives a sparse matrix-vector communication cost $T_{\text{Comm\_spmv}} = 6(t_s + (N/P_n)^{2/3}t_w)$ for all six sides of the cube, assuming send and receive operations are overlapped. In actual implementations, this time can also be overlapped with local matrix-vector calculations, but this is not taken into account here. Typically, in stencil codes almost all matrix-vector computations can be used to overlap the nearest-neighbor communication, so that the total time for the matrix-vector product can be modeled as $T_{\text{spmv}} = \max(T_{\text{Comm\_spmv}}, 2n_z t_c N/P_c)$, where P_c is the total number of cores in the system.

Implementations for global reductions typically use a tree-like communication pattern. The height of the tree is given by $\lceil \log_r(P_n)\rceil$ with r the tree radix. For the broadcast, which follows the reduce in an all-reduce operation, the same tree can be used. Hence the communication cost for reducing mP_c doubles, with P_c the total number of cores, to m doubles and broadcasting them again over the entire machine is $T_{\text{red}} = 2\lceil \log_r(P_n)\rceil(t_s + mt_w)$ [32, 22]. The communication cost for the on-node reduction and broadcast has been neglected. Since in the pipelined GMRES algorithms the reduction can be overlapped by other computations and communications, the total cost for global reductions will not include the part that is effectively overlapped but only that part that causes the processors to sit idle.

In Table 6.2, cost functions for the different variations of GMRES, presented in this paper, are listed. It is assumed that all dot-products and norms that can easily be combined in a single all-reduce are indeed combined. These models give the cost for a typical iteration, not modeling the startup and draining of the pipeline. Furthermore, all calculations on the smaller Hessenberg matrices are ignored in these models.

To model performance on a hypothetical exascale machine, we take parameters from the Swimlane 1 extrapolation in report [31]: 2^{20} nodes, 2^{10} cores per node, 100GB/s interconnect bandwidth, and 1μs interconnect latency. The flop rate per core is 1 Gflop/s, or $t_c = 2^{30}/10^{18}$s. For the reduction tree radix we take $r = 8$ instead of the more common, but much slower, binomial tree. We believe a higher radix is a better approximation for a dedicated reduction network as found in some of the high-end supercomputers.

Since the amount of work in Gram–Schmidt increases with the iteration number, we consider iteration 15, which is the average when using GMRES(30), i.e., restarting

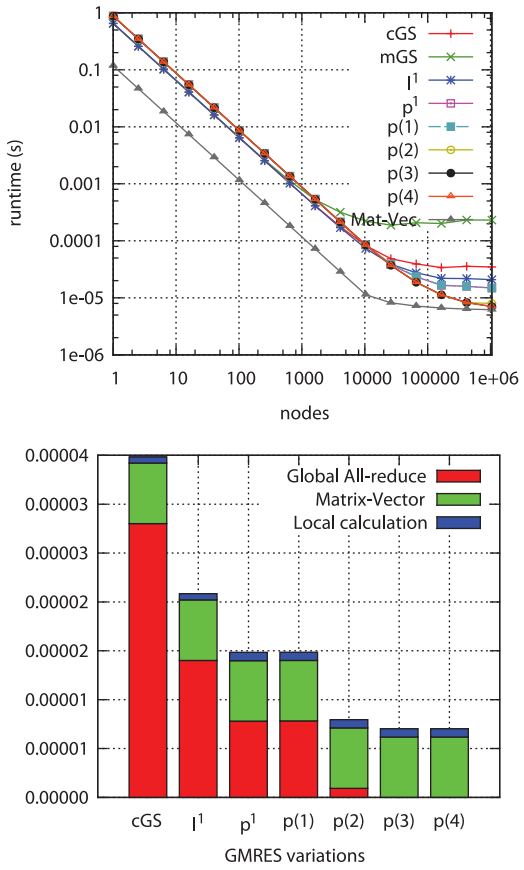


FIG. 6.2. *Left: Predicted runtime for a strong scaling experiment ($N = 2000^3$) with different GMRES versions on a hypothetical exascale machine. The exascale machine parameters are taken from* [31]. *Right: Prediction of the breakdown of the different GMRES variations in time spent at local computation, matrix-vector product, and global all-reduce communication on 2^{20} nodes with $N = 2000^3$. For $\ell \geq 3$ the global reduction latency can be completely hidden, which results in a 4.9× speedup for both p(3)- and p(4)-GMRES compared to normal GMRES with classical Gram–Schmidt.*

after 30 iterations. Figure 6.2, left, shows the runtime per iteration for the different GMRES formulations as a function of number of nodes for a strong scaling experiment with a total of $N = 8 \cdot 10^9 = 2000^3$ unknowns. This clearly shows the unattractiveness of modified Gram–Schmidt on large parallel machines. Due to the redundant work in the pipelined algorithms, GMRES with classical Gram–Schmidt and ℓ^1-GMRES are most efficient on a small number of nodes. The lines for p(1)-, p(2)-, and p(3)-GMRES coincide for a small number of nodes, until at certain points the reduction latency becomes longer than one, two, or three iterations, and the corresponding p(1)-, p(2)-, and p(3)-GMRES curves level off. These predictions for a hypothetical exascale machine show similar qualitative behavior as the measurements presented in Figure 6.1.

Figure 6.2, right, shows the breakdown of the different GMRES methods in their different components: T_{calc}, T_{spmv}, and T_{red}. Note that the T_{calc} part grows slightly for the pipelined solvers, while the reduction is partly or completely overlapped. For $\ell \geq 3$ the global reduction latency can be completely hidden, which results in a 4.9× speedup for both p(3)- and p(4)-GMRES compared to normal GMRES with classical Gram–Schmidt. The speedups for ℓ^1, p^1, p(1), and p(2) are 1.7×, 2.3×,

2.3×, and 4.4×, respectively. Modified Gram–Schmidt GMRES was left out from Figure 6.2, right, since the total runtime of 2.3e-4 was an order of magnitude bigger than the times for the other methods.

We did not include the underlying network topology in the model but refer the reader to [2] for a more detailed study of the feasibility of pipelining for different network configurations. The simplified cost model as presented here also does not include variability due to OS jitter, core speed variability, or load imbalance. It has been observed [21, 4] that these effects can have a serious impact on the scaling behavior of collective operations and hence the entire solver or application. Finally, the pipelined solvers are compared to GMRES using classical Gram–Schmidt. In practice for stability reasons, iterated classical or even modified Gram–Schmidt are used, which scale much worse. We believe iterated classical Gram–Schmidt can also be pipelined relatively easily, but we leave this for future work. In [18], the reorthogonalization step of iterated Gram–Schmidt in an eigenvalue solver is performed asynchronously.

7. Discussion. In this section, the use of preconditioning is discussed and a short comparison with so-called s-step GMRES is given.

7.1. Preconditioning. The GMRES algorithm, including the pipelined variants, can easily be applied to preconditioned linear systems. The preconditioned matrix-vector product will likely be much more expensive than the unpreconditioned SpMV. This means that fewer iterations will be required to hide global communication latency, leading to shorter pipelines and hence better stability properties. In such cases, the ℓ^1-, p^1-, or p(1)-GMRES variants might do.

Of course, it would not make much sense to hide the latency of global reductions in GMRES by overlapping them with much more global communication, for instance, when doing a multigrid V-cycle all the way down to the coarsest level. However, preconditioners based on the matrix-vector product, like Jacobi or polynomial preconditioning, seem like a good fit for the pipelined GMRES solvers.

When using multigrid on a large parallel machine (either standalone or as a preconditioner for some Krylov method), it makes sense to stop coarsening before the mesh becomes too small and then switch to either a direct solver or some Krylov method. This is also referred to as the truncated V-cycle or U-cycle [34]. On the coarsest grid there would be relatively few gridpoints per node and the communication overhead would be considerable. Hence, the application of a pipelined Krylov solver for the coarsest grid of a truncated V-cycle might be a good scenario.

7.2. Comparison with *s*-step GMRES. As this work has been inspired by the s-step or communication avoiding Krylov solvers by Hoemmen [24], Chronopoulos and Gear [8], and various other authors, a short discussion on the differences and similarities is appropriate here. The s-step Krylov solvers are designed to avoid communication on two different levels. On the one hand, they reduce the number of global synchronization phases. On the other hand, by the use of their matrix powers kernels [13], local synchronization as well as communication between processor and slow memory are reduced. The matrix powers kernel combines s matrix-vector products in such a way that the matrix has to be fetched from main memory only once. However, this is hard to combine with preconditioning. In s-step GMRES, orthogonalization is done using a combination of block modified Gram–Schmidt and a (communication optimal) tall and skinny QR (TSQR) factorization. The TSQR has the benefit that it can be implemented using BLAS3 routines, which typically achieve a larger percentage of the machine's peak performance than the vector oper-

ations from standard GMRES (matrix-vector, BLAS2, for classical Gram–Schmidt, and vector-vector, BLAS1, for modified Gram–Schmidt).

In terms of numerical stability, the s-step methods make some sacrifices by computing s vectors at once (using an appropriate basis) without intermediate orthogonalization. However, their orthogonalization method, the TSQR, is very accurate. Although in this work classical Gram–Schmidt was used, in practice a check for orthogonality will be used in combination with reorthogonalization. The pipelined algorithms can be modified to also hide the latency of the reorthogonalization. For the solution of linear equations, modified Gram–Schmidt or reorthogonalization in every iteration is overkill for most applications. However, for eigenvalue solvers, the accuracy of the orthogonalization is more important.

The pipelined solvers do not address the local communication to slow memory like the matrix powers kernel does in the s-step methods. However, this can easily be covered by a preconditioner. For instance, a stationary iterative method, such as ω-Jacobi and red-black Gauss–Seidel, or a polynomial preconditioner, for instance, Chebyshev iteration, can also be implemented using the matrix powers kernel [15]. Cache reuse in the matrix powers kernel is also better for a polynomial preconditioner than when used in s-step GMRES, since the intermediate vectors do not have to be stored.

8. Conclusion and outlook. We reported on our initial efforts to hide global communication latencies in Krylov solvers. We derived a pipelined version of GMRES in which this communication cost can be hidden by using nonblocking all-reduce operations. If global communication latency would be longer than the time required to perform a single SpMV, deeper pipelining can be used. However, the technique can also be combined with preconditioning, in which case only a short pipelining depth should be necessary.

We have shown speedups of the pipelined GMRES solver compared to classical GMRES of up to 1.84× in a parallel 2D Poisson benchmark using 512 cores. Larger speedups are to be expected on bigger machines. We predict scalability of the presented pipelined algorithms on a hypothetical exascale machine with an analytical performance model. We note predicted speedups of up to 4.9× for the pipelined algorithm compared to standard GMRES for solving a problem on a regular grid with $N = 2000^3$ unknowns.

Although the pipelined GMRES solvers do require some additional floating point operations, they certainly show potential. The amount of additional flops in p(ℓ)-GMRES increases linearly with the iteration number, just as the work in Gram–Schmidt does, and does not depend on ℓ. However, when the same pipelining idea would be applied to a short recurrence relation Krylov method, like CG or BiCGStab, the amount of redundant work would stay limited (it would probably only grow linearly for increasing ℓ). This investigation is future work.

Apart from the ability to overlap the network latency of the global reduction operations, other additional sources of parallelism were introduced in the pipelined GMRES algorithms. When executing on a system with a so-called accelerator, like a typical GPGPU or Intel's MIC, the communication between main memory, main CPU, accelerator processing cores, and accelerator memory can become possible bottlenecks. By relaxing the data dependencies, the pipelined GMRES algorithms allow better scheduling of finer grained tasks. Serial tasks such as updating the Hessenberg matrix H and I/O could be handled by the CPU, while the more lightweight accelerator cores can do the massively parallel number crunching. The pipelined solvers can also help

to overlap communication latency between host CPU and accelerator. We will explore this in future work.

Acknowledgment. We thank Jed Brown from Argonne National Laboratory for implementing p[1]-GMRES in PETSc.[8]

REFERENCES

[1] E. Agullo, J. Demmel, J. Dongarra, B. Hadri, J. Kurzak, J. Langou, H. Ltaief, P. Luszczek, and S. Tomov, *Numerical linear algebra on emerging architectures: The PLASMA and MAGMA projects*, J. Phys. Conf. Ser. 180 (2009), 012037.

[2] T. Ashby, P. Ghysels, W. Heirman, and W. Vanroose, *The impact of global communication latency at extreme scales on Krylov methods*, in Proceedings of the 12th International Conference on Algorithms and Architectures for Parallel Processing, 2012, pp. 428–442.

[3] Z. Bai, D. Hu, and L. Reichel, *A Newton basis GMRES implementation*, IMA J. Numer. Anal., 14 (1994), pp. 563–581.

[4] P. Beckman, K. Iskra, K. Yoshii, and S. Coghlan, *The influence of operating systems on the performance of collective operations at extreme scale*, in Proceedings of the 2006 IEEE International Conference on Cluster Computing, 2006, pp. 1–12.

[5] A. Bhatele, P. Jetley, H. Gahvari, L. Wesolowski, W. D. Gropp, and L. Kalé, *Architectural constraints to attain* 1 *Exaflop/s for three scientific application classes*, in Proceedings of the Parallel & Distributed Processing Symposium (IPDPS), IEEE, 2011, pp. 80–91.

[6] Å. Björck, *Numerics of Gram-Schmidt orthogonalization*, Linear Algebra Appl., 197 (1994), pp. 297–316.

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

[8] A. T. Chronopoulos and C. W. Gear, *s-step iterative methods for symmetric linear systems*, J. Comput. Appl. Math., 25 (1989), pp. 153–168.

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

[10] A. T. Chronopoulos, *s-step iterative methods for (non) symmetric (in) definite linear systems*, SIAM J. Numer. Anal., 28 (1991), pp. 1776–1789.

[11] J. Daniel, W. B. Gragg, L. Kaufman, and G. W. Stewart, *Reorthogonalization and stable algorithms for updating the Gram-Schmidt QR factorization*, Math. Comp., 30 (1976), pp. 772–795.

[12] J. Demmel, M. Heath, and H. van der Vorst, *Parallel Numerical Linear Algebra*, Acta Numer., Cambridge University Press, Cambridge, UK, 1993, pp. 111–197.

[13] J. Demmel, M. Hoemmen, M. Mohiyuddin, and K. Yelick, *Avoiding communication in sparse matrix computations*, in Proceedings of the IEEE International Symposium on Parallel and Distributed Processing, 2008, pp. 1–12.

[14] 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*, Appl. Numer. Math., 18 (1995), pp. 441–459.

[15] P. Ghysels, P. Kłosiewicz, and W. Vanroose, *Improving the arithmetic intensity of multigrid with the help of polynomial smoothers*, Numer. Linear Algebra Appl., 19 (2012), pp. 253–267.

[16] A. Grama, G. Karypis, V. Kumar, and A. Gupta, *Introduction to Parallel Computing*, 2nd ed., Addison-Wesley, Reading, MA, 2003.

[17] W. Gropp, *Update on Libraries for Blue Waters*, http://jointlab.ncsa.illinoisedu/events/workshop3/pdf/presentations/Gropp-Update-on-Libraries.pdf.

[18] V. Hernandez, J. E. Roman, and A. Tomas, *Parallel Arnoldi eigensolvers with enhanced scalability via global communications rearrangement*, Parallel Comput., 33 (2007), pp. 521–540.

[19] T. Hoefler, P. Kambadur, R. Graham, G. Shipman, and A. Lumsdaine, *A Case for Standard Non-Blocking Collective Operations*, in Recent Advances in Parallel Virtual Machine and Message Passing Interface, Springer, Berlin, 2007, pp. 125–134.

[8] http://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/KSP/KSPPGMRES.html.

[20] T. Hoefler and A. Lumsdaine, *Optimizing non-blocking collective operations for infiniband*, in Proceedings of the IEEE International Symposium on Parallel and Distributed Processing, 2008, pp. 1–8.
[21] T. Hoefler, T. Schneider, and A. Lumsdaine, *Characterizing the influence of system noise on large-scale applications by simulation*, in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2010.
[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, ACM, 2010, pp. 597–604.
[23] T. Hoefler, J. Squyres, G. Bosilca, G. Fagg, A. Lumsdaine, and W. Rehm, *Non-Blocking Collective Operations for MPI-2*, Technical report 8, Open Systems Lab, Indiana University, 2006.
[24] M. Hoemmen, *Communication-Avoiding Krylov Subspace Methods*, Ph.D. thesis, University of California, 2010.
[25] W. D. Joubert and G. F. Carey, *Parallelizable restarted iterative methods for nonsymmetric linear systems. Part* I: *Theory*, Int. J. Comput. Math., 44 (1992), pp. 243–267.
[26] S. K. Kim and A. T. Chronopoulos, *An efficient parallel algorithm for extreme eigenvalues of sparse nonsymmetric matrices*, Int. J. High Performance Comput. Appl., 6 (1992), pp. 407–420.
[27] C. Moler and D. Morrison, *Replacing square roots by Pythagorean sums*, IBM J. Res. Develop., 27 (1983), pp. 577–581.
[28] L. Reichel, *Newton interpolation at Leja points*, BIT, 30 (1990), pp. 332–346.
[29] Y. Saad and M. H. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.
[30] Y. Saad, *A flexible inner-outer preconditioned GMRES algorithm*, SIAM J. Sci. Comput., 14 (1993), pp. 461–461.
[31] J. Shalf, S. S. Dosanjh, and J. Morrison, *Exascale computing technology challenges*, in High Performance Computing for Computational Science, J. Palma, M. Daydé, O. Marques, and J. Lopes, eds., Lecture Notes in Comput. Sci. 6449, Springer, Berlin, 2010, pp. 1–25.
[32] R. Thakur, R. Rabenseifner, and W. Gropp, *Optimization of collective communication operations in MPICH*, Int. J High Performance Comput. Appl., 19 (2005), pp. 49–66.
[33] H. F. Walker and L. Zhou, *A simpler GMRES*, Numer. Linear Algebra Appl., 1 (1994), pp. 571–581.
[34] D. Xie and L. R. Scott, *The parallel U-cycle multigrid method*, in Proceedings of the 8th Copper Mountain Conference on Multigrid Methods, 1997.