Version of Record: https://www.sciencedirect.com/science/article/pii/S001046551830256X
Manuscript_a44c38acbbce256ff1b17d1ef124c5fe

Alternating Anderson-Richardson method: An efficient alternative to preconditioned Krylov methods for large, sparse linear systems

Phanish Suryanarayana[*,a], Phanisri P. Pratapa[a], John E. Pask[b]

[a]*College of Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA*
[b]*Physics Division, Lawrence Livermore National Laboratory, Livermore, CA 94550, USA*

Abstract

We present the Alternating Anderson-Richardson (AAR) method: an efficient and scalable alternative to preconditioned Krylov solvers for the solution of large, sparse linear systems on high performance computing platforms. Specifically, we generalize the recently proposed Alternating Anderson-Jacobi (AAJ) method (Pratapa et al., *J. Comput. Phys.* (2016), 306, 43–54) to include preconditioning, discuss efficient parallel implementation, and provide serial MATLAB and parallel C/C++ implementations. In serial applications to nonsymmetric systems, we find that AAR is comparably robust to GMRES, using the same preconditioning, while often outperforming it in time to solution; and find AAR to be more robust than Bi-CGSTAB for the problems considered. In parallel applications to the Helmholtz and Poisson equations, we find that AAR shows superior strong and weak scaling to GMRES, Bi-CGSTAB, and Conjugate Gradient (CG) methods, using the same preconditioning, with consistently shorter times to solution at larger processor counts. Finally, in massively parallel applications to the Poisson equation, on up to 110,592 processors, we find that AAR shows superior strong and weak scaling to CG, with shorter minimum time to solution. We thus find that AAR offers a robust and efficient alternative to current state-of-the-art solvers, with increasing advantages as the number of processors grows.

Key words: Linear systems of equations, Parallel computing, Anderson extrapolation, Richardson iteration, Electronic structure calculations

PROGRAM SUMMARY

Manuscript Title: Alternating Anderson-Richardson method: An efficient alternative to preconditioned Krylov methods for large, sparse linear systems
Authors: Phanish Suryanarayana, Phanisri P. Pratapa, John E. Pask
Program Title: AAR
Journal Reference:
Catalogue identifier:
Licensing provisions: GNU General Public License 3 (GPL)
Programming language: MATLAB for the MATLAB version. C/C++ for the PETSc version. C/C++ for the standalone version.

[*]Corresponding Author (*phanish.suryanarayana@ce.gatech.edu*)

Computer: Any system with MATLAB for the MATLAB version. Any system with C/C++ compiler and MPI library for the PETSc and standalone versions.
Operating system: Unix/Linux or Windows for the MATLAB version. Unix/Linux for the PETSc and standalone versions.
RAM: Problem dependent.
Number of processors used: As many as available via MPI for the PETSc and standalone versions.
Keywords: Linear systems of equations, Parallel computing, Anderson extrapolation, Richardson iteration, Electronic structure calculations
Classification: 4.8
External routines/libraries: MATLAB 2014a or later for the MATLAB version.
PETSc 3.5.3 (`http://www.mcs.anl.gov/petsc`) or later and
MVAPICH2 2.1 (`http://mvapich.cse.ohio-state.edu/`) or later for the PETSc version.
MVAPICH2 2.1 or later for the standalone version.
Nature of problem: Linear systems of equations.
Solution method: Anderson extrapolation at periodic intervals within a preconditioned Richardson iteration.
Restrictions: Jacobi preconditioning for the standalone version.
Running time: Problem dependent. Timing results for selected cases provided in paper.

1. Introduction

Linear systems of equations are encountered in the gamut of applications areas within computational physics, from quantum to continuum to celestial mechanics. The strategies adopted for solving such systems can be broadly classified into two categories: direct methods [1] and iterative methods [2]. For relatively small system sizes, direct methods such as QR decomposition and LU factorization are generally the preferred approaches. However, as the size of the system increases, direct methods become inefficient—in terms of both computational cost and storage requirements—due to poor scaling with system size relative to iterative approaches, particularly Krylov subspace based techniques such as the Generalized Minimal Residual (GMRES) [3] and Conjugate Gradient (CG) [4] methods. Therefore, such iterative approaches are often the methods of choice for the solution of large-scale linear systems of equations.

A number of physical applications require the repeated solution of large, sparse linear systems. For example, in real-space quantum molecular dynamics calculations [5, 6, 7, 8] or electronic structure calculations with exact exchange [9, 10], the Poisson equation may be solved hundreds of thousands of times within a single simulation. Therefore, it is critical to reduce the time to solution as far as possible in such situations, a goal typically achieved through parallel computing, wherein the number of floating point operations per second increases linearly with the number of processors. However, the cost associated with inter-processor communication, especially global communication, limits the parallel efficiency of linear solvers, which in turn limits the reduction in wall time that can be achieved in practice [11, 12, 13]. Therefore, there is wide interest in developing algorithms that scale well on modern large-scale parallel computers which can contain tens to hundreds of thousands of computational cores or more [14, 15, 16].

Krylov subspace methods such as GMRES and CG have limited parallel scalability due to the large number of global operations inherent to them [17, 18]. In this context, the classical Richard-

son and Jacobi fixed-point iterations [19, 2] are ideally suited for massive parallelization by virtue of the strict locality of operations required, i.e., they do not require the calculation of any dot products, other than those required for the calculation of the residual [20, 21]. However, they suffer from extremely large prefactors and poor scaling with system size compared to Krylov subspace methods, which has made them unattractive on even the largest modern platforms. This has motivated the development of strategies that significantly accelerate the convergence of the basic Richardson/Jacobi iterations while maintaining their underlying parallel scalability and simplicity to the maximum extent possible [22, 23]. Such approaches include the Chebyshev acceleration technique [2] and the recently developed Scheduled Relaxation Jacobi (SRJ) method [22]. However, Chebyshev acceleration requires the computation of the extremal eigenvalues of the coefficient matrix, which may be computationally expensive. Moreover, the current formulation of the SRJ method is applicable only to linear systems arising from the discretization of elliptic equations using second-order finite-differences. For such reasons, Krylov subspace methods have remained the methods of choice in general for the solution of large, sparse linear systems.

Recently, we proposed to employ Anderson extrapolation [24][1] at periodic intervals within the classical Jacobi iteration, resulting in the so called Alternating Anderson-Jacobi (AAJ) method [23]. This strategy was found to accelerate the Jacobi iteration by orders of magnitude, to the point in fact of outperforming GMRES significantly in serial computations without preconditioning[2]. In the present work, we generalize the AAJ method to include preconditioning, discuss efficient parallel implementation, and provide serial MATLAB and parallel C/C++ implementations. In serial applications to nonsymmetric systems, we find that AAR is comparably robust to GMRES, using the same preconditioning, while often outperforming it in time to solution. In parallel applications to the Helmholtz and Poisson equations, on up to 110,592 processors, we find that AAR shows superior strong and weak scaling to GMRES, Bi-CGSTAB, and Conjugate Gradient (CG) methods, using the same preconditioning, with consistently shorter times to solution at larger processor counts. We thus find that AAR offers a robust and efficient alternative to current state-of-the-art solvers, with increasing advantages as the number of processors grows.

The remainder of this paper is organized as follows. In Section 2, we describe the preconditioned AAR method. We demonstrate the efficiency and parallel scaling of the method in Section 3. Finally, we conclude in Section 4.

[1] Anderson's extrapolation has been successfully utilized for accelerating the convergence of non-linear fixed-point iterations arising in electronic structure calculations [25], coupled fluid-structure transient thermal problems [26], as well as neutronics and plasma physics [27]. In the context of linear systems of equations, Anderson's technique bears a close connection to the GMRES method [28, 29, 30].

[2] In the context of electronic structure calculations, the analogue of the AAJ method for nonlinear fixed-point iterations—referred to as Periodic Pulay [31]—is found to significantly accelerate the convergence of the self-consistent field (SCF) method.

2. Alternating Anderson-Richardson method

In this section, we present the preconditioned Alternating Anderson-Richardson (AAR) method for the solution of large, sparse linear systems:

$$\begin{aligned} &\mathbf{A}\mathbf{x} = \mathbf{b}\,, \\ &\mathbf{A} \in \mathbb{C}^{N\times N}\,,\ \ \mathbf{x} \in \mathbb{C}^{N\times 1}\,,\ \ \mathbf{b} \in \mathbb{C}^{N\times 1}\,, \end{aligned} \tag{1}$$

where $\mathbb{C}$ is the set of complex numbers. This approach generalizes the Alternating Anderson-Jacobi (AAJ) method presented previously [23] to include preconditioning and therefore accelerate convergence. In this work, we summarize and focus on the incorporation of preconditioning and the development of an efficient parallel formulation and implementation; a more complete discussion of the underlying alternating Anderson approach found in our previous work [23].

In the AAR method, Anderson extrapolation [24] is performed periodically within a preconditioned Richardson fixed-point iteration [2] to accelerate its convergence, while maintaining its parallel scalability to the maximum extent possible. Since the method makes no assumptions about the symmetry of $\mathbf{A}$, it is applicable to symmetric and nonsymmetric systems alike. Moreover, it is amenable to the three types of preconditioning: left, right, and split [2, 32]. For the sake of simplicity, we choose left preconditioning in the present work. Mathematically, the linear system in Eq. 1 is solved using a fixed-point iteration of the form

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{B}_k\mathbf{f}_k\,, \quad k = 0, 1, \ldots \tag{2}$$

where the matrix $\mathbf{B}_k \in \mathbb{C}^{N\times N}$ can be written as

$$\mathbf{B}_k = \begin{cases} \omega\mathbf{I} & \text{if } (k+1)/p \notin \mathbb{N}\,, \quad \text{(Richardson)} \\ \beta\mathbf{I} - (\mathbf{X}_k + \beta\mathbf{F}_k)(\mathbf{F}_k^T\mathbf{F}_k)^{-1}\mathbf{F}_k^T & \text{if } (k+1)/p \in \mathbb{N}\,. \quad \text{(Anderson)} \end{cases} \tag{3}$$

Above, $\omega \in \mathbb{C}$ is the relaxation parameter used in the Richardson update, $\beta \in \mathbb{C}$ is the relaxation parameter used in the Anderson update, $\mathbf{I} \in \mathbb{R}^{N\times N}$ is the identity matrix, the superscript T denotes the conjugate transpose, and p is the frequency of Anderson extrapolation. Additionally, $\mathbf{X}_k \in \mathbb{C}^{N\times m}$ and $\mathbf{F}_k \in \mathbb{C}^{N\times m}$ contain the iteration and residual histories at the k^{th} iteration:

$$\mathbf{X}_k = \left[(\mathbf{x}_{k-m+1} - \mathbf{x}_{k-m}) \quad (\mathbf{x}_{k-m+2} - \mathbf{x}_{k-m+1}) \quad \ldots \quad (\mathbf{x}_k - \mathbf{x}_{k-1})\right]\,, \tag{4}$$

$$\mathbf{F}_k = \left[(\mathbf{f}_{k-m+1} - \mathbf{f}_{k-m}) \quad (\mathbf{f}_{k-m+2} - \mathbf{f}_{k-m+1}) \quad \ldots \quad (\mathbf{f}_k - \mathbf{f}_{k-1})\right]\,, \tag{5}$$

where $m+1$ is the number of iterates used for Anderson extrapolation,[3] and the residual vector

$$\mathbf{f}_k = \mathbf{M}^{-1}(\mathbf{b} - \mathbf{A}\mathbf{x}_k)\,, \tag{6}$$

with preconditioner $\mathbf{M}^{-1} \in \mathbb{C}^{N\times N}$. As discussed in the context of AAJ [23], the key to the robustness and efficiency of the method is the Anderson extrapolation step which minimizes the

[3] In the initial iterations, $\mathbf{x}_j$ in Eq. 4 and $\mathbf{f}_j$ in Eq. 5 with $j < 0$ are omitted or can be set to null vectors if a pseudoinverse is used to evaluate $(\mathbf{F}_k^T\mathbf{F}_k)^{-1}$ in Eq. 3.

l_2 norm of the residual in the column space of $\mathbf{F}_k$, yielding consistent and substantial reductions with increasing history length m;[4] while the key to the parallel scalability of the method is that the extrapolation is performed only periodically, thus reducing nonlocal communications significantly.

We summarize the AAR method in Fig. 1, wherein $\mathbf{x}_0$ denotes the initial guess, r_k denotes the relative l_2 norm of the residual vector (i.e., $r_k = \|\mathbf{A}\mathbf{x}_k - \mathbf{b}\|/\|\mathbf{b}\|$), and ϵ is the tolerance specified for convergence. In order to enhance parallel scalability, we reduce global communications by checking for convergence of the fixed-point iteration (i.e., calculating r_k) only during Anderson extrapolation steps. Overall, AAR involves 1 matrix-vector product and 1 preconditioner application per iteration, and an average of $m(m+3)/2p$ inner product and $2(1+m/p)$ DAXPY[5] vector operations per iteration. Note that the inner product operations occur only during the Anderson extrapolation steps and can be carried out as dense matrix-matrix and matrix-vector operations: two important features that can be exploited to increase both serial and parallel efficiency, as we show in Section 3.

The key difference between the AAR and AAJ [23] methods lies in the choice of residual vector Eq. 6. In the AAR method, any available preconditioner $\mathbf{M}^{-1}$ can be employed, whereas in AAJ, $\mathbf{M} = \mathbf{D}$ is the diagonal part of $\mathbf{A}$. The AAR method thus generalizes the AAJ method in the sense that the AAJ method is recovered for the particular choice of preconditioner $\mathbf{M}^{-1} = \mathbf{D}^{-1}$, i.e., the classical Jacobi preconditioner. Furthermore, just as the AAJ method can be understood as a generalization of the Jacobi [2] and Anderson-Jacobi (AJ) [24, 23] methods, the AAR method can be understood as a generalization of the Richardson [2] and Anderson-Richardson (AR) [28, 29, 30] methods. Specifically, the AR method is recovered for $p = 1$, while the Richardson iteration is obtained in the limit $p \to \infty$.

As discussed in the context of AAJ [23], the convergence of the AAR method can be understood through its connection to GMRES. First, we note that with complete history (i.e., $m = \infty$), AAR is equivalent to AR for $\omega \neq 0$ and $p \geq 1$ since, upon extrapolation, the residual norm is minimized over the same Krylov subspace regardless of previous extrapolations (in exact arithmetic).[6] Second, it has been shown [28, 29, 30] that AR with complete history is equivalent to GMRES without restart, in the sense that the iterates of one can be readily obtained from those of the other (in exact arithmetic).[7] Hence, in the above sense, AAR with complete history is equivalent to GMRES without restart and so must show corresponding convergence.

With finite history and restarts, however, as typical in practice to reduce storage and/or orthogonalization costs, both AAR and GMRES can require additional iterations to reach convergence. And in this context, as demonstrated in Section 3, we typically find shorter times to solution for AAR than for GMRES, with increasing advantages for AAR in parallel calculations as the number of processors grows. As discussed in the context of AAJ [23], this may be due in part to the fact that AAR retains and minimizes over the most recent m-vector history at each extrapolation, while GMRES begins anew at each restart. The key advantage of AAR, however, in parallel calculations

[4] In practice, m is limited by available memory and/or finite precision effects as the matrix $\mathbf{F}_k^T\mathbf{F}_k$ in Eq. 3 becomes ill-conditioned.

[5] Given a scalar α, and vectors $\mathbf{x}$ and $\mathbf{y}$, the operation DAXPY refers to: $\alpha\mathbf{x} + \mathbf{y}$.

[6] Excluding potential differences in stagnation [33, 34].

[7] Excluding potential differences in stagnation [30].

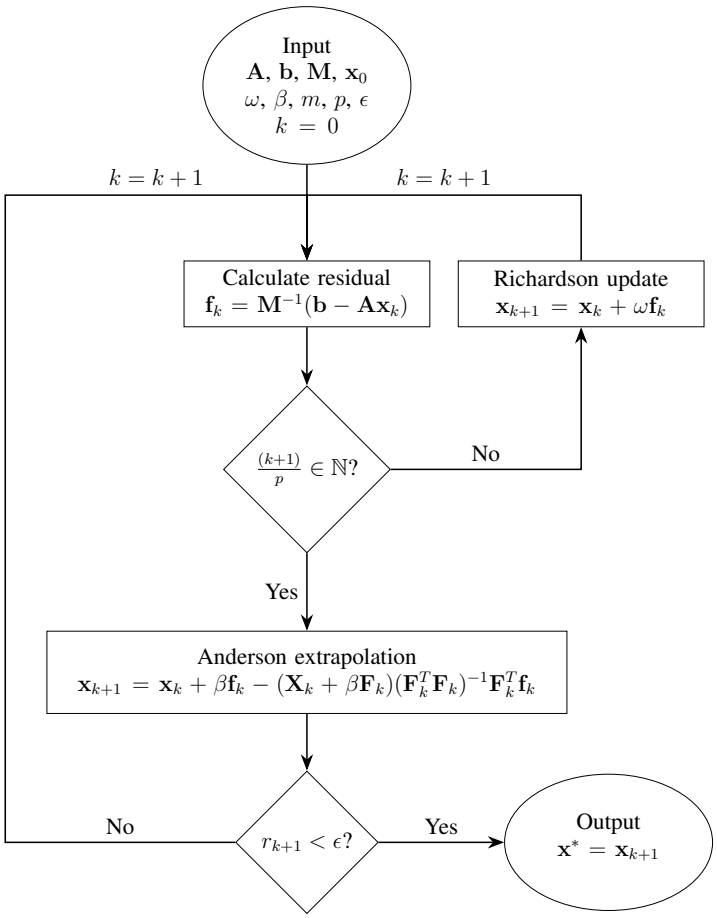


Figure 1: The preconditioned Alternating Anderson-Richardson (AAR) method.

in particular, is that the majority of iterations are simple, computationally local Richardson iterations, with Anderson extrapolations only every p iterations. A study of the mathematical properties of AAR in relation to GMRES and AR can be found in the recent work of Lupo Pasini [34].

Finally, in finite precision, other considerations come into play. For example, while for complete history and exact arithmetic, the iterates produced by AAR upon extrapolation are independent of ω and p, this no longer holds with finite history and floating point arithmetic. Nevertheless, as shown in the context of AAJ [23], the dependence is generally weak over a broad range of values so that the method is generally insensitive to the particular choice of values within the range. Similar insensitivity is found for the Anderson extrapolation parameter β, though larger values can accelerate convergence in better-conditioned (or well preconditioned) problems, consistent with findings in the nonlinear context [31]. Finally, while in exact arithmetic, a larger history length m must generally improve convergence, by providing a larger subspace over which to minimize, in finite precision, the increasing condition number of the matrix $\mathbf{F}_k^T\mathbf{F}_k$ in Eq. 3 with increasing m limits the effective history length in practice. Given the general insensitivity of the method to the particular choice of parameter values, we use the same default set $\{\omega, \beta, m, p\} = \{0.6, 0.6, 9, 8\}$

for all systems in the present work. While possible to optimize for a particular application area, we have found these to be sufficient in a broad range of applications, with available preconditioning in particular, as demonstrated in Section 3.

3. Results and discussion

In this section, we demonstrate the efficiency and scaling of the preconditioned Alternating Anderson-Richardson (AAR) method in the solution of large, sparse linear systems of equations. Specifically, we consider an assortment of nonsymmetric systems from Matrix Market[8] as well as Poisson and complex-valued Helmholtz equations arising in real-space electronic structure calculations, and use the default parameters $\{\omega, \beta, m, p\} = \{0.6, 0.6, 9, 8\}$ in AAR for all systems.

3.1. Matrix Market: assortment of nonsymmetric systems

In order to demonstrate the robustness and efficiency of AAR, we first study the relative performance of AAR, GMRES, and Bi-CGSTAB in MATLAB[9] for nonsymmetric linear systems available in the Matrix Market repository.[10] Specifically, we consider ten matrices that arise in various areas of computational physics, including oil reservoir modeling, fluid dynamics, and the study of plasmas. In cases where $\mathbf{b}$ is not provided, we use a random vector with unit norm. We use the default MATLAB parameters for GMRES (restart frequency of 30) and Bi-CGSTAB, with a vector of all ones as the starting guess $\mathbf{x}_0$ in all cases. The simulations are performed on a workstation with the following configuration: Intel Xeon Processor E3-1220 v3 (Quad Core, 3.10GHz Turbo, 8MB), 16GB (2x8GB) 1600MHz DDR3 ECC UDIMM.

In Table 1, we present the timings (in seconds) obtained for achieving a convergence tolerance of $\epsilon = 10^{-6}$ for two cases: (i) Jacobi preconditioner and (ii) ILU(0) preconditioner. We first note that the simple Jacobi preconditioner is insufficient to obtain convergence for a number of these systems, though AAR is able to converge for more systems than GMRES and Bi-CGSTAB. On the other hand, we see that ILU(0) preconditioning is sufficient to obtain convergence for all ten systems for AAR, all but one for GMRES, but only half the systems for Bi-CGSTAB. Moreover, we see that when ILU(0) preconditioning is employed, AAR outperforms GMRES significantly for all but one system; while Bi-CGSTAB can outperform both AAR and GMRES significantly when it converges, but even then outperforms AAR in only two of those five cases. We thus find that AAR shows comparable robustness to GMRES while often outperforming it, while Bi-CGSTAB, though highly competitive when it converges, shows considerably less robustness in the applications considered.

To better understand the timings in Table 1, we show in Table 2 the numbers of matrix-vector, inner product, and DAXPY vector operations for cases where all methods converge and the number of iterations is at least p. From the results, it is clear that Bi-CGSTAB requires the fewest operations, resulting in high efficiency when it does converge. By profiling, we find that the similar

[8] http://math.nist.gov/MatrixMarket/

[9] https://www.mathworks.com/

[10] The MATLAB implementation of AAR is available as part of the code accompanying this paper. For GMRES and Bi-CGSTAB, the inbuilt functions in MATLAB are utilized.

timings of AAR and Bi-CGSTAB for sherman5 are due to AAR's ability to leverage Level 2 and Level 3 BLAS to compute inner products, as well as the inbuilt checks performed by MATLAB for Bi-CGSTAB. In contrast, the superior performance of AAR relative to GMRES for utm1700b is due to the substantially smaller number of operations, while for sherman5 it is again due to AAR's ability to leverage Level 2 and Level 3 BLAS to compute inner products as well as inbuilt checks performed by MATLAB for GMRES.

Importantly, we note that while the default parameters for AAR work well for a wide range of applications, as demonstrated above, the flexibility to tune the parameters can be leveraged to tailor AAR for particular applications areas. For example, by simply tuning β, the solution times for the utm3060, utm1700b, and sherman5 applications above can be brought down to 0.087 s, 0.026 s, and 0.010 s for β = 1.7, 1.3, and 0.8, respectively. More importantly, however, by virtue of the superior parallel scaling of AAR, times to solution can be brought down substantially further still relative to standard solvers such as GMRES and Bi-CGSTAB, as we now show.

Matrix	N	Jacobi preconditioner			ILU(0) preconditioner		
		AAR	GMRES	Bi-CGSTAB	AAR	GMRES	Bi-CGSTAB
utm3060	3060	15.104	-	1.805	0.283	0.191	0.043
utm1700a	1700	-	-	-	0.004	0.013	0.006
utm1700b	1700	4.733	-	-	0.045	0.159	0.023
fidap029	2870	0.004	0.019	0.007	0.005	0.016	0.008
sherman5	3312	0.028	0.095	0.016	0.014	0.025	0.014
mcfe	765	-	0.392	-	0.007	0.018	-
memplus	17758	0.521	1.344	-	0.735	2.185	-
add32	4960	0.021	0.070	-	0.021	0.061	-
mcca	180	0.019	0.023	0.008	0.013	0.021	-
fs_680_3	680	0.211	-	-	0.004	-	-

Table 1: Time taken in seconds by AAR, GMRES, and Bi-CGSTAB in MATLAB for nonsymmetric linear systems from Matrix Market. The symbol '-' is used to indicate that convergence was not achieved within 1000 sec.

Matrix	AAR			GMRES			Bi-CGSTAB		
	MVP	IP*	DAXPY*	MVP	IP	DAXPY	MVP	IP	DAXPY
utm3060	1809	12204	7686	474	7299	7299	203	406	609
utm1700b	479	3186	2020	645	9900	9900	157	314	471
sherman5	73	486	308	28	434	434	47	94	141

Table 2: Number of matrix-vector products (MVP), inner products (IP), and DAXPY vector operations required by AAR, GMRES, and Bi-CGSTAB in MATLAB employing the ILU(0) preconditioner. IP and DAXPY denote that all operations are performed using Level 1 BLAS. IP* denotes that operations are performed using Level 2 (17%) and Level 3 (83%) BLAS. DAXPY* denotes that operations are performed using Level 1 (76%) and Level 2 (24%) BLAS.

3.2. Orbital-free Density Functional Theory: Helmholtz equation

Next, we study the relative performance of AAR, GMRES with standard restarts [3], GMRES with augmented restarts [35] (LGMRES), and Bi-CGSTAB [36] in PETSc [37, 38][11]. We consider the periodic Helmholtz problem arising in real-space orbital-free Density Functional Theory (OF-DFT) calculations [39, 40, 41]:

$$-\frac{1}{4\pi}\nabla^2 V(\mathbf{r}) + Q\,V(\mathbf{r}) = P\,\rho^{\alpha}(\mathbf{r}) \;\text{ in }\; \Omega, \qquad \begin{cases} V(\mathbf{r}) = V(\mathbf{r} + L_i\hat{\mathbf{e}}_i) \;\text{ on }\; \partial\Omega\,, \\ \hat{\mathbf{e}}_i \cdot \nabla V(\mathbf{r}) = \hat{\mathbf{e}}_i \cdot \nabla V(\mathbf{r} + L_i\hat{\mathbf{e}}_i) \;\text{ on }\; \partial\Omega\,, \end{cases} \tag{7}$$

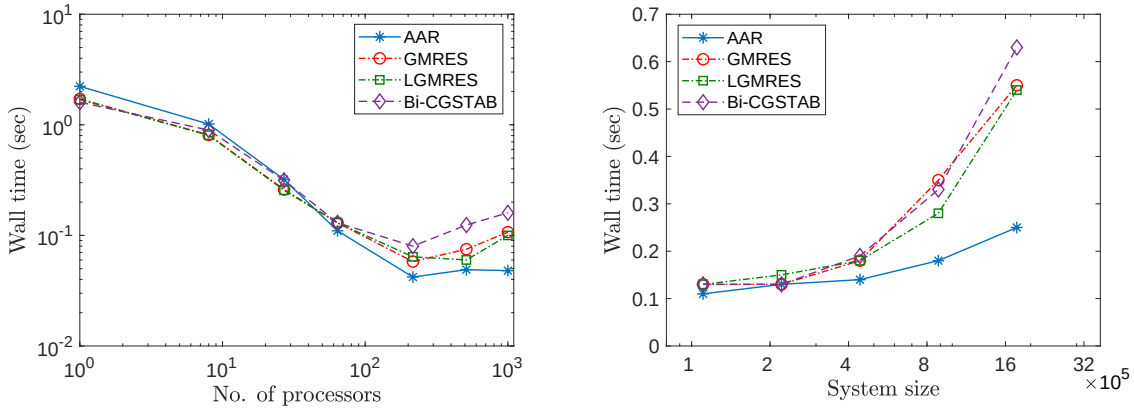
where $V(\mathbf{r})$ is the kernel potential [42, 43], $\rho(\mathbf{r})$ is the electron density, $\alpha = \frac{5}{6} + \frac{\sqrt{5}}{6}$, $P = 0.003277 - i0.009081$, $Q = -0.134992 - i0.070225$, $i = \sqrt{-1}$, and Ω is a cuboidal domain with side lengths L_i, unit vectors $\hat{\mathbf{e}}_i$ along each edge, and boundary $\partial\Omega$. The equation is discretized using sixth-order accurate finite-differences on a uniform grid with mesh-size h. The computations are parallelized by decomposing the domain into cubical subdomains of equal size, with communication between processors handled via Message Passing Interface (MPI) in the PETSc framework.

In the Anderson extrapolation step of AAR, we perform only one global communication call (i.e., `MPI_Allreduce`) to simultaneously determine r and the complete matrix $\mathbf{F}_k^T\mathbf{F}_k$. Since the matrix $\mathbf{F}_k^T\mathbf{F}_k$ is generally ill-conditioned, we compute its inverse using the Moore-Penrose pseudoinverse [44]. We employ the default PETSc parameters for GMRES, LGMRES, and Bi-CGSTAB. In all the simulations, we use a vector of all zeros as the starting guess $\mathbf{x}_0$ and a convergence tolerance of $\epsilon = 10^{-6}$ on the relative residual. We perform the calculations on a computer cluster consisting of 16 nodes with the following configuration: Altus 1804i Server - 4P Interlagos Node, Quad AMD Opteron 6276, 16C, 2.3 GHz, 128GB, DDR3-1333 ECC, 80GB SSD, MLC, 2.5" HCA, Mellanox ConnectX 2, 1-port QSFP, QDR, memfree, CentOS, Version 5, and connected through InfiniBand cable.

We first consider a $3 \times 3 \times 3$ aluminum supercell based on a face-centered cubic (FCC) unit cell having lattice constant 7.78 Bohr, with atoms randomly displaced from ideal positions. We discretize the domain using a finite-difference grid with a mesh-size of $h = 0.486$ Bohr, which is sufficient to achieve chemical accuracy in the energy and atomic forces. For the resulting linear system, we employ block Jacobi preconditioning with ILU(0) factorization on each block. Fig. 2a shows the wall time taken by AAR, GMRES, LGMRES, and Bi-CGSTAB on 1, 8, 27, 64, 216, 512, and 1000 cores. We observe that even though the performances of all approaches are similar at low core counts (a consequence of requiring similar number of iterations), AAR starts demonstrating superior performance as the core count is increased. In particular, the minimum wall time achieved by AAR is a factor of 1.38, 1.43, and 1.90 smaller than GMRES, LGMRES, and Bi-CGSTAB, respectively. This is a consequence of the significantly less global communication in AAR compared to the other methods.

We next periodically replicate the above $3 \times 3 \times 3$ aluminum system along one direction by factors of 1, 2, 4, 8, and 16, i.e., we generate $3 \times 3 \times 3$, $6 \times 3 \times 3$, $12 \times 3 \times 3$, $24 \times 3 \times 3$, and

[11] The PETSc implementation of AAR for complex-valued systems is available as part of the code accompanying this paper. For GMRES, LGMRES, and Bi-CGSTAB, the inbuilt functions in PETSc are utilized.



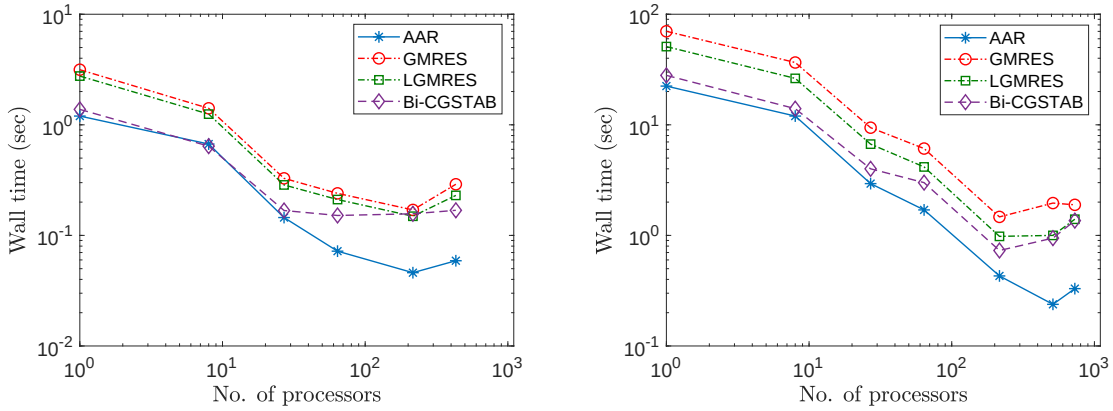
(a) Strong scaling

(b) Weak scaling

Figure 2: Strong and weak scaling of AAR, GMRES, LGMRES, and Bi-CGSTAB for the periodic Helmholtz equation using block Jacobi preconditioning with ILU(0) factorization on each block in PETSc. In strong scaling, the minimum wall time taken by AAR is 1.38, 1.43, and 1.90 times smaller than GMRES, LGMRES, and Bi-CGSTAB, respectively. In weak scaling, the CPU time taken by AAR, GMRES, LGMRES, and Bi-CGSTAB scales with system size as $\mathcal{O}(N^{1.28})$, $\mathcal{O}(N^{1.57})$, $\mathcal{O}(N^{1.51})$, $\mathcal{O}(N^{1.59})$, respectively.

$48 \times 3 \times 3$ supercells. Correspondingly, we choose 64, 128, 256, 512, and 1024 computational cores. Again, we select $h = 0.486$ Bohr and employ block Jacobi preconditioning with ILU(0) factorization on each block. We plot the results of this weak scaling study in Fig. 2b, from which we obtain $\mathcal{O}(N^{1.28})$, $\mathcal{O}(N^{1.57})$, $\mathcal{O}(N^{1.51})$, and $\mathcal{O}(N^{1.59})$ scaling with system size for AAR, GMRES, LGMRES, and Bi-CGSTAB, respectively. Notably, all approaches demonstrate slightly superlinear scaling even though the number of iterations remain constant. This is due to the increased cost of global communications at larger core counts. Therefore, the performance of AAR relative to the other methods is expected to further improve as the number of cores increases.

It is worth noting that the performance gap between AAR and other approaches increases as the linear system becomes less well conditioned, as demonstrated in previous work for AAJ vs. GMRES in serial computations [23]. In order to verify this result for AAR in parallel computations, we consider the Helmholtz equation for a $1 \times 1 \times 1$ Al supercell with lattice constant of 7.78 Bohr and randomly displaced atoms. To demonstrate the effect of conditioning, we choose a simple Jacobi preconditioner $\mathbf{M}^{-1} = \mathbf{D}^{-1}$, where $\mathbf{D}$ is the diagonal part of $\mathbf{A}$. In Fig. 3, we present the strong scaling of AAR, GMRES, LGMRES, and Bi-CGSTAB for mesh-sizes of $h = 0.216$ and $h = 0.108$ Bohr. We observe that as the mesh gets finer and condition number of $\mathbf{A}$ becomes larger, the speedup of AAR over the other methods increases at both small and large core counts. Specifically, for $h = 0.216$ Bohr, the minimum wall time taken by AAR is 3.70, 3.26, and 3.28 times smaller than GMRES, LGMRES, and Bi-CGSTAB, respectively. The corresponding numbers for $h = 0.108$ Bohr are 6.13, 4.08, and 3.04, respectively. Therefore, we conclude that as the solution of the linear system becomes more challenging (e.g., in the absence of an effective preconditioner), the speedup of AAR over Krylov subspace approaches like GMRES and



(a) $h = 0.216$ Bohr (b) $h = 0.108$ Bohr

Figure 3: Strong scaling of AAR, GMRES, LGMRES, and Bi-CGSTAB for the periodic Helmholtz equation with Jacobi preconditioning in PETSc. For $h = 0.216$ Bohr, the minimum wall time taken by AAR is 3.70, 3.26, and 3.28 times smaller than GMRES, LGMRES, and Bi-CGSTAB, respectively. For $h = 0.108$ Bohr, the minimum wall time taken by AAR is 6.13, 4.08, and 3.04 times smaller than GMRES, LGMRES, and Bi-CGSTAB, respectively.

Bi-CGSTAB is expected to become more substantial in both the serial and parallel settings.[12]

3.3. Density Functional Theory: Poisson equation

Next, we study the relative performance of AAR and Conjugate Gradient (CG) methods for solving the periodic Poisson problem arising in real-space Density Functional Theory (DFT) calculations [45, 46, 47, 48]:

$$-\frac{1}{4\pi}\nabla^2\phi(\mathbf{r}) = \rho(\mathbf{r}) + b(\mathbf{r}) \;\text{ in }\; \Omega, \qquad \begin{cases} V(\mathbf{r}) = V(\mathbf{r} + L_i\hat{\mathbf{e}}_i) \;\text{ on }\; \partial\Omega\,, \\ \hat{\mathbf{e}}_i \cdot \nabla V(\mathbf{r}) = \hat{\mathbf{e}}_i \cdot \nabla V(\mathbf{r} + L_i\hat{\mathbf{e}}_i) \;\text{ on }\; \partial\Omega\,, \end{cases} \tag{8}$$

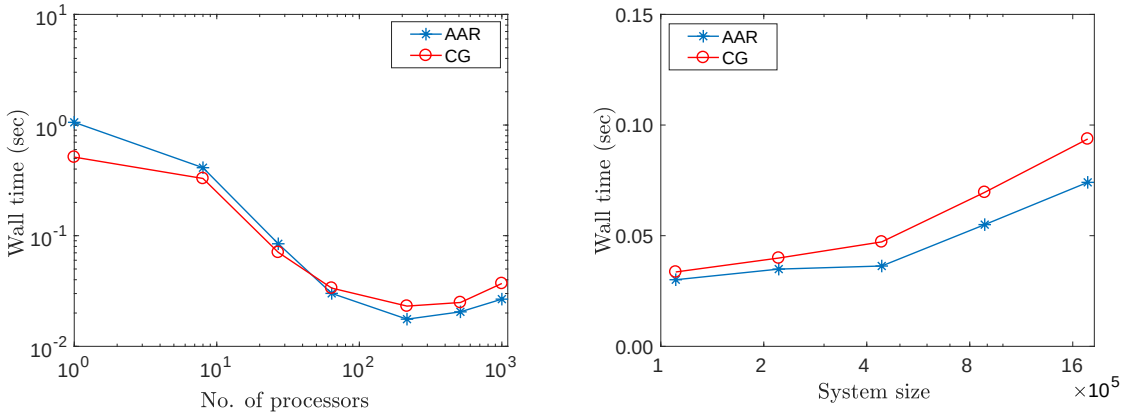
where $\phi(\mathbf{r})$ is the electrostatic potential, $\rho(\mathbf{r})$ is the electron density, $b(\mathbf{r})$ is the pseudocharge density [45, 49, 50, 51], and Ω is a cuboidal domain with side lengths L_i, unit vectors $\hat{\mathbf{e}}_i$ along each edge, and boundary $\partial\Omega$. The Poisson equation is discretized using sixth-order accurate finite-differences on a uniform grid with mesh-size $h = 0.486$ Bohr, which is sufficient to achieve chemical accuracy in the energy and atomic forces. The computations are parallelized by decomposing the domain into cubical subdomains of equal size, with communication between processors handled via Message Passing Interface (MPI).

In the Anderson extrapolation step of AAR, we again perform only one global communication call (i.e., `MPI_Allreduce`) to simultaneously determine r and the complete matrix $\mathbf{F}_k^T\mathbf{F}_k$, whose inverse in computed using the Moore-Penrose pseudoinverse. In all calculations, we again choose

[12]The upturns in strong scaling plots at largest core counts arise due to insufficient computational work per core relative to local inter-processor communications when the chosen computation is spread beyond a certain number of cores. This indicates the strong scaling limit of the current implementation for the chosen problem size.

a vector of all zeros as the starting guess $\mathbf{x}_0$, and a convergence tolerance of $\epsilon = 10^{-6}$ on the relative residual. Calculations on up to 1,024 cores were carried out on the same computer cluster as for the Helmholtz problem in Section 3.2. Larger calculations, up to 110,592 cores, were carried out on the Vulcan IBM BG/Q machine at the Lawrence Livermore National Laboratory, consisting of 24,576 compute nodes, with 16 computational cores and 16 GB memory per node, for a total of 393,216 cores and 1.6 PB memory.

We first consider a $3 \times 3 \times 3$ Al supercell based on a FCC unit cell with lattice constant 7.78 Bohr, with atoms randomly displaced from ideal positions, and again employ block Jacobi preconditioning with ILU(0) factorization on each block (default in PETSc) in the solution of the resulting linear systems. In Fig. 4a, we plot the wall time taken by AAR and CG as implemented in PETSC[13] on 1, 8, 27, 64, 216, 512, and 1000 computational cores. At small core counts, CG demonstrates better performance than AAR by virtue of requiring fewer iterations to achieve convergence. However, as the number of cores is increased, the performance of AAR relative to CG improves, with the minimum wall time taken by AAR being a factor of 1.31 smaller than CG by virtue of the lesser global communication required by AAR.



(a) Strong scaling

(b) Weak scaling

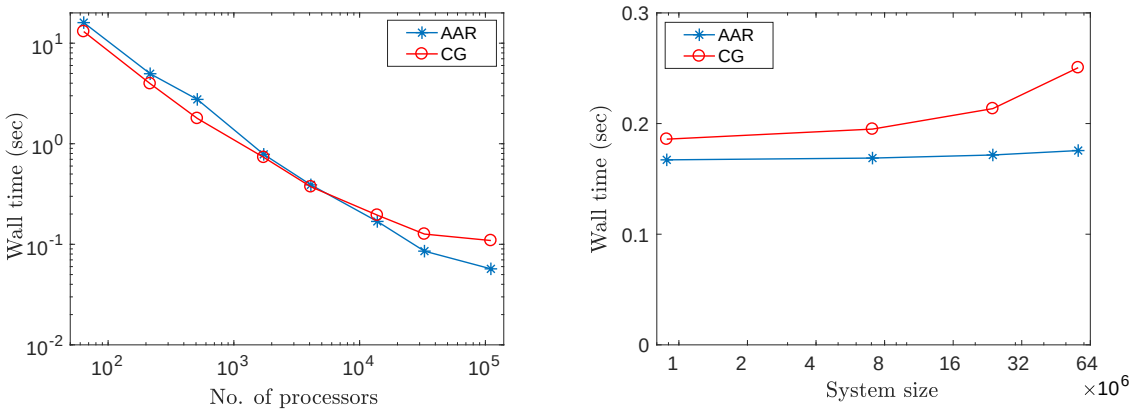
Figure 4: Strong and weak scaling of AAR and CG for the periodic Poisson equation using block Jacobi preconditioning with ILU(0) factorization on each block in PETSc. In strong scaling, the minimum wall time taken by AAR is 1.31 times smaller than CG. In weak scaling, the CPU time taken by AAR and CG scales with system size as $\mathcal{O}(N^{1.33})$ and $\mathcal{O}(N^{1.38})$, respectively.

We next perform a weak scaling study by periodically replicating the above $3 \times 3 \times 3$ system along one direction by factors of 1, 2, 4, 8, and 16, i.e., we generate $3 \times 3 \times 3$, $6 \times 3 \times 3$, $12 \times 3 \times 3$, $24 \times 3 \times 3$, and $48 \times 3 \times 3$ supercells. Correspondingly, we choose 64, 128, 256, 512, and 1024 computational cores. Again, we employ block Jacobi preconditioning with ILU(0) factorization on each block. In Fig. 4b, we plot the wall time taken by AAR and CG for the resulting systems, from

[13]The PETSc implementation of AAR for real-valued systems is available as part of the code accompanying this paper. For CG, the inbuilt function in PETSc is utilized.

which we obtain the weak scaling with system size to be $\mathcal{O}(N^{1.33})$ and $\mathcal{O}(N^{1.38})$, respectively. As before, even though the number of iterations does not vary with system size, the increasing cost associated with global communications results in superlinear scaling for both approaches[14]. Therefore, the performance of AAR relative to CG is expected to further improve as core counts are increased, a result which we verify next.

To assess the efficiency and scaling of AAR in larger-scale parallel calculations, up to 110,592 cores, we consider strong and weak scaling on the Vulcan IBM BG/Q machine at the Lawrence Livermore National Laboratory. We implement AAR and CG using C and MPI directly[15], and choose a simple Jacobi preconditioner for parallel scalability. For the strong scaling study, we choose a $12 \times 12 \times 12$ Al supercell with atoms randomly displaced, and a maximum of 110,592 computational cores. For the weak scaling study, we go from $6 \times 6 \times 6$ supercell on 1728 processors to $24 \times 24 \times 24$ supercell on 110,592 processors, with atomic displacements, electron density, and pseudocharge density periodically repeated from the $6 \times 6 \times 6$ system. We present the results obtained in Fig. 5. We find that the minimum wall time achieved by AAR within the number of cores available for this study is 1.91 times smaller than that achieved by CG. In addition, the weak scaling with system size for AAR and CG are $\mathcal{O}(N^{1.01})$ and $\mathcal{O}(N^{1.07})$, respectively. This demonstrates again the increased advantage of AAR over current state-of-the-art Krylov solvers in parallel computations as the number of processors is increased.



(a) Strong scaling

(b) Weak scaling

Figure 5: Strong and weak scaling of AAR and CG for the periodic Poisson problem with Jacobi preconditioning. In strong scaling, the minimum wall time taken by AAR is 1.91 times smaller than CG. In weak scaling, the CPU time taken by AAR and CG scales with system size as $\mathcal{O}(N^{1.01})$ and $\mathcal{O}(N^{1.07})$, respectively.

[14] Another factor contributing to the superlinear scaling for both AAR and CG is the inefficiency of communications between processors in the computer cluster used for the study.

[15] The standalone implementation of AAR for real-valued systems using C and MPI directly is available as part of the code accompanying this paper.

4. Concluding remarks

We generalized the recently proposed Alternating Anderson-Jacobi (AAJ) method to include preconditioning and make it particularly well suited for scalable high-performance computing, and demonstrated its efficiency and scaling in the solution of large, sparse linear systems on parallel computers. Specifically, the AAR method employs Anderson extrapolation at periodic intervals within a preconditioned Richardson iteration to accelerate convergence while maintaining its underlying parallel scalability and simplicity to the maximum extent possible. In serial applications to nonsymmetric systems, we find that AAR is comparably robust to GMRES, using the same preconditioning, while often substantially outperforming it in time to solution; and find AAR to be more robust than Bi-CGSTAB for the problems considered. In parallel applications to the Helmholtz and Poisson equations, we find that AAR shows superior strong and weak scaling to GMRES, Bi-CGSTAB, and Conjugate Gradient (CG) methods, using the same preconditioning, with consistently shorter times to solution at larger processor counts. Finally, in massively parallel applications to the Poisson equation, on up to 110,592 processors, we find that AAR shows superior strong and weak scaling to CG, with shorter minimum time to solution.

Our findings suggest that the AAR method provides an efficient and scalable alternative to current state-of-the-art preconditioned Krylov solvers for the solution of large, sparse linear systems on high performance computing platforms, with increasing advantage as the number of processors is increased. Moreover, the method is simple and general, applying to symmetric and nonsymmetric systems, real and complex alike. Additional mathematical analysis which provides further insights into the performance of the AAR method and therefore enables the development of effective preconditioners tailored to it will enable still larger-scale applications, and so constitutes a potentially fruitful direction for future research.

5. Acknowledgements

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. We gratefully acknowledge support from the Laboratory Directed Research and Development Program. P.S. and P.P. also gratefully acknowledge the support of National Science Foundation under Grant Number 1333500.

References

[1] T. A. Davis, Direct Methods for Sparse Linear Systems, SIAM, 2006.

[2] Y. Saad, Iterative Methods for Sparse Linear Systems (2nd ed), SIAM, 2003.

[3] Y. Saad, M. H. Schultz, SIAM Journal on Scientific and Statistical Computing 7 (1986) 856–869.

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

[5] X. Jing, N. Troullier, D. Dean, N. Binggeli, J. R. Chelikowsky, K. Wu, Y. Saad, Physical Review B 50 (1994) 12234.

[6] F. Shimojo, R. K. Kalia, A. Nakano, P. Vashishta, Computer Physics Communications 167 (2005) 151–164.

[7] D. Osei-Kuffuor, J.-L. Fattebert, Physical Review Letters 112 (2014) 046401.

[8] P. Suryanarayana, P. P. Pratapa, A. Sharma, J. E. Pask, Computer Physics Communications 224 (2018) 288 – 298.

[9] J. P. Perdew, M. Ernzerhof, K. Burke, The Journal of Chemical Physics 105 (1996) 9982–9985.

[10] L. Lin, Journal of Chemical Theory and Computation 12 (2016) 2242–2249.

[11] E. de Sturler, H. A. van der Vorst, in: International Conference on High-Performance Computing and Networking, Springer, pp. 190–195.

[12] I. S. Duff, H. A. Van Der Vorst, Parallel Computing 25 (1999) 1931–1970.

[13] L. T. Yang, R. P. Brent, in: Parallel and Distributed Processing Symposium, 2003. Proceedings. International, IEEE, pp. 11–pp.

[14] X.-Y. Zuo, T.-X. Gu, Z.-Y. Mo, Applied Mathematics and Computation 215 (2010) 4101–4109.

[15] P. Ghysels, T. J. Ashby, K. Meerbergen, W. Vanroose, SIAM Journal on Scientific Computing 35 (2013) C48–C71.

[16] L. C. McInnes, B. Smith, H. Zhang, R. T. Mills, Parallel Computing 40 (2014) 17–31.

[17] E. De Sturler, H. A. van der Vorst, Applied Numerical Mathematics 18 (1995) 441–459.

[18] P. Ghysels, W. Vanroose, Parallel Computing 40 (2014) 224–238.

[19] W. Hackbusch, Iterative Solution of Large Sparse Systems of Equations, volume 40, Springer, 1994.

[20] G. H. Golub, H. A. Van Der Vorst, The State of the Art in Numerical Analysis (2001) 63–92.

[21] R. Barrett, M. W. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. Van der Vorst, Templates for the solution of linear systems: building blocks for iterative methods, volume 43, SIAM, 1994.

[22] X. I. Yang, R. Mittal, Journal of Computational Physics 274 (2014) 695–708.

[23] P. P. Pratapa, P. Suryanarayana, J. E. Pask, Journal of Computational Physics 306 (2016) 43–54.

[24] D. G. Anderson, Journal of the Association for Computing Machinery 12 (1965) 547–560.

[25] P. Pulay, Chemical Physics Letters 73 (1980) 393–398.

[26] V. Ganine, N. Hills, B. Lapworth, International Journal for Numerical Methods in Fluids 71 (2013) 939–959.

[27] J. Willert, W. T. Taitano, D. Knoll, Journal of Computational Physics 273 (2014) 278–286.

[28] T. Rohwedder, R. Schneider, Journal of Mathematical Chemistry 49 (2011) 1889–1914.

[29] H. F. Walker, P. Ni, SIAM Journal on Numerical Analysis 49 (2011) 1715–1735.

[30] F. A. Potra, H. Engler, Linear Algebra and its Applications 438 (2013) 1002–1011.

[31] A. S. Banerjee, P. Suryanarayana, J. E. Pask, Chemical Physics Letters 647 (2016) 31–35.

[32] M. Benzi, Journal of Computational Physics 182 (2002) 418–477.

[33] M. L. Pasini, Deterministic and stochastic acceleration techniques for Richardson-type iterations, Ph.D. thesis, Emory University, 2018.

[34] M. L. Pasini, Preprint (2018).

[35] A. H. Baker, E. R. Jessup, T. Manteuffel, SIAM Journal on Matrix Analysis and Applications 26 (2005) 962–984.

[36] H. A. Van der Vorst, SIAM Journal on Scientific and Statistical Computing 13 (1992) 631–644.

[37] S. Balay, J. Brown, , K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, H. Zhang, PETSc Users Manual, Technical Report ANL-95/11 - Revision 3.4, Argonne National Laboratory, 2013.

[38] S. Balay, W. D. Gropp, L. C. McInnes, B. F. Smith, in: E. Arge, A. M. Bruaset, H. P. Langtangen (Eds.), Modern Software Tools in Scientific Computing, Birkhäuser Press, 1997, pp. 163–202.

[39] N. Choly, E. Kaxiras, Solid State Communications 121 (2002) 281–286.

[40] S. Ghosh, P. Suryanarayana, Journal of Computational Physics 307 (2016) 634–652.

[41] P. Suryanarayana, D. Phanish, Journal of Computational Physics 275 (2014) 524–538.

[42] Y. A. Wang, N. Govind, E. A. Carter, Physical Review B 58 (1998) 13465.

[43] Y. A. Wang, N. Govind, E. A. Carter, Physical Review B 60 (1999) 16350.

[44] A. J. Laub, Matrix Analysis for Scientists and Engineers, SIAM, 2005.

[45] J. E. Pask, P. A. Sterne, Physical Review B 71 (2005) 113101.

[46] P. Suryanarayana, K. Bhattacharya, M. Ortiz, Journal of the Mechanics and Physics of Solids 61 (2013) 38–60.

[47] J. E. Pask, N. Sukumar, S. E. Mousavi, International Journal for Multiscale Computational Engineering 10 (2012) 83–99.

[48] S. Ghosh, P. Suryanarayana, Computer Physics Communications 216 (2017) 109 – 125.

[49] P. Suryanarayana, V. Gavini, T. Blesgen, K. Bhattacharya, M. Ortiz, Journal of the Mechanics and Physics of Solids 58 (2010) 256–280.

[50] P. Suryanarayana, K. Bhattacharya, M. Ortiz, Journal of Computational Physics 230 (2011) 5226–5238.

[51] S. Ghosh, P. Suryanarayana, Computer Physics Communications 212 (2017) 189 – 204.