Electronic Transactions on Numerical Analysis.
Volume 43, pp. 125-141, 2014.
Copyright © 2014, Kent State University.
ISSN 1068-9613.

ETNA
Kent State University
http://etna.math.kent.edu

AN EFFICIENT DEFLATION TECHNIQUE FOR THE COMMUNICATION-AVOIDING CONJUGATE GRADIENT METHOD*

ERIN CARSON†, NICHOLAS KNIGHT†, AND JAMES DEMMEL†‡

Abstract. By fusing s loop iterations, *communication-avoiding* formulations of Krylov subspace methods can asymptotically reduce sequential and parallel communication costs by a factor of $O(s)$. Although a number of communication-avoiding Krylov methods have been developed, there remains a serious lack of available communication-avoiding preconditioners to accompany these methods. This has stimulated active research in discovering which preconditioners can be made compatible with communication-avoiding Krylov methods and developing communication-avoiding methods which incorporate these preconditioners. In this paper we demonstrate, for the first time, that deflation preconditioning can be applied in communication-avoiding formulations of Lanczos-based Krylov methods such as the conjugate gradient method while maintaining an $O(s)$ reduction in communication costs. We derive a deflated version of a communication-avoiding conjugate gradient method, which is mathematically equivalent to the deflated conjugate gradient method of Saad et al. [SIAM J. Sci. Comput., 21 (2000), pp.1909–1926]. Numerical experiments on a model problem demonstrate that the communication-avoiding formulations can converge at comparable rates to the classical formulations, even for large values of s. Performance modeling illustrates that $O(s)$ speedups are possible when performance is communication bound. These results motivate deflation as a promising preconditioner for communication-avoiding Krylov subspace methods in practice.

Key words. Krylov subspace methods, deflation, iterative solvers, minimizing communication, sparse matrices

AMS subject classifications. 65F08, 65F10, 65F50, 65Y05, 65Y20

1. Introduction.

Krylov subspace methods (KSMs) are a class of iterative algorithms commonly used to solve the linear system $Ax = b$ when A is large and sparse. In each iteration m, updates to the next solution x_{m+1} and residual r_{m+1} consist of one or more sparse matrix-vector multiplications (SpMVs) and vector-vector operations in each iteration. On modern computers, these operations are *communication-bound*: the movement of data rather than the computation is the limiting factor in performance. Recent efforts have focused on *communication-avoiding* KSMs (CA-KSMs), which reorder the computations in classical KSMs to perform $O(s)$ computation steps of the algorithm for each communication step; see, e.g., [6, 10, 12, 14, 16, 20, 26, 27, 49, 51]. This formulation allows an $O(s)$ reduction in the total communication costs per iteration, which can translate into significant speedups [36].

In addition to speed per iteration, the performance of iterative methods also depends on the total number of iterations required for convergence. For the conjugate gradient method (CG), the KSM of choice for solving symmetric positive definite (SPD) systems, it is well-known that the rate of convergence in exact arithmetic can be bounded in terms of the eigenvalue distribution. Although these bounds are not always tight and additional complications arise in finite precision computations (see, e.g., [34]), nonetheless, preconditioning techniques, wherein the system's eigenvalue distribution is altered to improve the convergence bounds, have been employed successfully in practice; for a survey of approaches, see [44]. Unfortunately, except for simple examples like (block) Jacobi, polynomial, and sparse approximate inverse preconditioners, the ability to exploit temporal locality across KSM iterations is diminished by preconditioning, and the relative benefits of a CA-KSM, compared to its classical counterpart, decline; see, e.g., [27]. Avoiding communication in a general preconditioned method seems to necessitate significant modifications to the algorithm.

*Received October 13, 2013. Accepted November 2, 2014. Published online on December 12, 2014. Recommended by K. Jbilou.

†Department of EECS, University of California, Berkeley ({ecc2z, knight, demmel}@cs.berkeley.edu).

‡Department of Mathematics, University of California, Berkeley.

ETNA
Kent State University
http://etna.math.kent.edu

This has stimulated an active area of research in designing practical preconditioners for CA-KSMs with much recent progress. To this end, we propose deflation, which can be viewed as a type of preconditioning with a singular preconditioner, as a feasible technique for improving convergence in CA-KSMs. We derive a communication-avoiding deflated CG (CA-D-CG), based on the deflated CG formulation (which we refer to as 'D-CG') in [45]. Our analysis shows that the additional costs of CA-D-CG over CA-CG are of lower-order, which means that, as in (non-deflated) CA-CG, we still expect a possible $O(s)$ speedup per s steps over the classical implementation. This motivates deflation as a promising preconditioner for CA-KSMs in practice. We give a numerical example and performance modeling results which demonstrate that choosing the number of deflation vectors as well as the blocking factor s result in complex, machine-dependent tradeoffs between the convergence rate and the time per iteration.

2. Related work. Deflation and augmentation techniques have been applied to improve convergence in many KSMs since the mid 1980s; for a survey, see [46, Chapter 9]. Many approaches in the literature can be viewed as instances of more general deflation/augmentation frameworks [21, 25]. Connections have also been drawn between deflation and multilevel preconditioners; see, e.g., [48] and the references therein.

In this work, we consider the case of CG, the first KSM that was modified to perform deflation [17, 39]. We note that the potential for eigenvalue deflation was also known to Rutishauser before such methods gained popularity in the literature [18]. In this work, we study the D-CG formulation as given in [45].

CG is convenient since it allows us to concretely demonstrate our algorithmic reorganization while sidestepping technical issues involving breakdown, i.e., where KSM iterates may not exist in exact arithmetic. However, we see no obstacles beyond breakdown for extending our approach to other KSMs that deflate in a similar manner.

Many authors have reformulated KSMs to reduce communication costs. Our approach is most closely related to the CA-KSMs developed by Hoemmen et al. [27, 36], which in turn are based on s-step KSMs proposed in the 1980s; see [27] for a historical perspective on avoiding communication in KSMs. Works that consider CG in particular include [6, 10, 11, 27, 49, 51]. The derivation of the CA-D-CG algorithm here most closely follows [6].

As mentioned in Section 1, there has been much recent work in the development of practical preconditioners for CA-KSMs. Grigori et al. developed a CA-ILU(0) preconditioner for CA-GMRES [24]. For structured problems, their method exploits a novel mesh ordering to obtain triangular factors that can be applied with less communication. There is also recent work in developing a new "underlapping" technique in communication-avoiding domain decomposition preconditioners for CA-KSMs [5]. For the case of preconditioners with both sparse and low-rank components (e.g., hierarchical semiseparable matrices), applying the low-rank components dominates the communication costs. Techniques in [27, 30] block together several applications of the low-rank components in order to amortize communication costs over several KSM iterations. We also note that there has been recent work in developing a high-performance deflated "pipelined" conjugate gradient method [22].

Previous work has developed efficient deflation techniques for CA-KSMs in order to recover information lost after restarting the Arnoldi process. Wakam and Erhel [41] extended a special case of CA-GMRES [27, 36] with an adaptive augmentation approach. Both their algorithm and ours aim to reduce the frequency of global collectives incurred by deflation/augmentation compared to previous approaches. However, our applications differ: in our case we apply deflation as a more general preconditioning technique for Lanczos-based methods. We note that the algorithm presented in [41] restricts the constructed Krylov bases

ETNA
Kent State University
http://etna.math.kent.edu

to Newton polynomials. It may be beneficial to extend their approach to the more general family of polynomials which we consider here.

2.1. The communication-avoiding conjugate gradient method. We briefly review the communication-avoiding conjugate gradient method (CA-CG) for solving $Ax = b$, where A is SPD and given any vector x_0 interpreted as an initial approximation to x.

For reference, the classical CG method is shown in Algorithm 2.1. Within the iteration loop, communication occurs in the SpMV Ap_m required by lines 3 and 5 as well as in the inner products in lines 3 and 6.

ALGORITHM 2.1. Conjugate Gradient Method (CG).

Require: Approximate solution x_0 to $Ax = b$

1: $r_0 = b - Ax_0$, $p_0 = r_0$
2: **for** $m = 0, 1, \ldots$ until convergence **do**
3: $\quad \alpha_m = (r_m^T r_m)/(p_m^T A p_m)$
4: $\quad x_{m+1} = x_m + \alpha_m p_m$
5: $\quad r_{m+1} = r_m - \alpha_m A p_m$
6: $\quad \beta_{m+1} = (r_{m+1}^T r_{m+1})/(r_m^T r_m)$
7: $\quad p_{m+1} = r_{m+1} + \beta_{m+1} p_m$
8: **end for**
9: **return** x_{m+1}

In CA-CG, iterations are split into an inner loop over $0 \leq j < s$ and an outer loop over k, whose range depends on the number of steps until convergence. We index the iteration m in CG as the iteration $m = sk+j$ in CA-CG. By induction on lines 4, 5, and 7 of Algorithm 2.1,

$$p_{sk+j}, r_{sk+j}, x_{sk+j} - x_{sk} \in \mathcal{K}_{s+1}(A, p_{sk}) + \mathcal{K}_s(A, r_{sk}),$$

for $0 \leq j \leq s$, where $\mathcal{K}_i(A, v)$ denotes the i-th Krylov subspace of A with respect to v, i.e.,

$$\mathcal{K}_i(A, v) = \text{span}\{v, Av, A^2 v, \ldots, A^{i-1} v\}.$$

Therefore, we let length-$(2s+1)$ vectors $x'_{k,j}$, $r'_{k,j}$, and $p'_{k,j}$ denote the coordinates for $x_{sk+j} - x_{sk}$, r_{sk+j}, and p_{sk+j}, respectively, in the columns of

$$V_k = [P_k,\ R_k] = [\rho_0(A)p_{sk},\ \ldots,\ \rho_s(A)p_{sk},\ \rho_0(A)r_{sk},\ \ldots,\ \rho_{s-1}(A)r_{sk}], \tag{2.1}$$

where ρ_i is a polynomial of degree i. That is, we have

$$x_{sk+j} - x_{sk} = V_k x'_{k,j}, \quad r_{sk+j} = V_k r'_{k,j}, \quad \text{and} \quad p_{sk+j} = V_k p'_{k,j}, \tag{2.2}$$

for $0 \leq j \leq s$. For brevity, we will refer to V_k as a basis, although the columns of V_k need not be linearly independent, e.g., for $k = 0$, $\mathcal{K}_s(A, r_0) \subseteq \mathcal{K}_{s+1}(A, p_0)$ since $p_0 = r_0$.

We assume that the polynomials in (2.1) can be computed via a three-term recurrence in terms of the parameters γ_i, θ_i, and σ_i, as

$$\begin{aligned} &\rho_0(A) = 1, \quad \rho_1(A) = (A - \theta_0 I)\rho_0(A)/\gamma_0, \quad \text{and} \\ &\rho_{i+1}(A) = ((A - \theta_i I)\rho_i(A) - \sigma_i \rho_{i-1}(A))/\gamma_i, \end{aligned} \tag{2.3}$$

for $1 \leq i < s$. This three-term recurrence covers a large class of polynomials including classical orthogonal polynomials. The monomial, Newton, and Chebyshev bases are common choices for generating Krylov bases; see, e.g., [43]. To simplify notation, we assume the basis parameters remain the same throughout the iteration.

ETNA
Kent State University
http://etna.math.kent.edu

The basis V_k is generated at the beginning of each outer loop using the current r_{sk} and p_{sk} vectors. Assuming A is sufficiently sparse, these $O(s)$-dimensional bases can be computed after only reading A (sequential case) or exchanging vector entries with neighbors (parallel case) $O(1)$ times, using the communication-avoiding 'matrix powers kernel' described in [16, 27, 36].

Substituting each polynomial ρ_i on the right-hand side of (2.1) by its recursive definition given in (2.3), rearranging terms, and postmultiplying by $p'_{k,j}$, we obtain

$$A\underline{V}_k p'_{k,j} = V_k B_k p'_{k,j}, \tag{2.4}$$

where $\underline{V}_k = [\underline{P}_k, 0, \underline{R}_k, 0]$, and $\underline{P}_k$ and $\underline{R}_k$ are P_k and R_k, respectively, with the last columns omitted, and

$$B_k = \begin{bmatrix} \begin{bmatrix} C_{k,s+1} & 0_{s+1,1} \end{bmatrix} & \\ & \begin{bmatrix} C_{k,s} & 0_{s,1} \end{bmatrix} \end{bmatrix},$$

with

$$C_{k,j+1} = \begin{bmatrix} \theta_0 & \sigma_1 & & \\ \gamma_0 & \theta_1 & \ddots & \\ & \gamma_1 & \ddots & \sigma_{j-1} \\ & & \ddots & \theta_{j-1} \\ & & & \gamma_{j-1} \end{bmatrix}.$$

Then by (2.2) and (2.4), the multiplication Ap_{sk+j} in the standard basis becomes $B_k p'_{k,j}$ in the basis V_k. Recall that B_k and $p'_{k,j}$ are both of dimension $O(s)$, which means that they either fit in fast memory (in the sequential case) or are local to each processor (in the parallel case), and thus the computation $B_k p'_{k,j}$ does not require data movement.

In each outer loop of Algorithm 2.2 below, we compute the $O(s)$-by-$O(s)$ Gram matrix $G_k = V_k^T V_k$. Then by (2.2) and (2.4), the inner products in lines 3 and 6 of Algorithm 2.1 can be written as

$$\begin{aligned} r_{sk+j}^T r_{sk+j} &= r'^T_{k,j} G_k r'_{k,j} && \text{for} \quad 0 \le j \le s, \quad \text{and} \\ p_{sk+j}^T A p_{sk+j} &= p'^T_{k,j} G_k B_k p'_{k,j} && \text{for} \quad 0 \le j < s. \end{aligned}$$

Thus, after G_k has been computed in the outer loop, the inner products can be computed without additional communication. Although many details are omitted, this gives the general idea behind avoiding data movement in Lanczos-based KSMs. The resulting CA-CG method is shown below in Algorithm 2.2.

ALGORITHM 2.2. Communication-Avoiding Conjugate Gradient (CA-CG).

Require: Approximate solution x_0 to $Ax = b$

$r_0 = b - Ax_0$, $p_0 = r_0$
for $k = 0, 1, \ldots$ until convergence **do**
  Compute P_k, R_k, let $V_k = [P_k, R_k]$; assemble B_k.
  $G_k = V_k^T V_k$
  $p'^T_{k,0} = [1, 0_{2s}^T]$, $r'^T_{k,0} = [0_{s+1}^T, 1, 0_{s-1}^T]$, $x'_{k,0} = 0_{2s+1}$
  for $j = 0, \ldots, s-1$ **do**
    $\alpha_{sk+j} = (r'^T_{k,j} G_k r'_{k,j}) / (p'^T_{k,j} G_k B_k p'_{k,j})$

$x'_{k,j+1} = x'_{k,j} + \alpha_{sk+j} p'_{k,j}$
$r'_{k,j+1} = r'_{k,j} - \alpha_{sk+j} B_k p'_{k,j}$
$\beta_{sk+j+1} = (r'^T_{k,j+1} G_k r'_{k,j+1})/(r'^T_{k,j} G_k r'_{k,j})$
$p'_{k,j+1} = r'_{k,j+1} + \beta_{sk+j+1} p'_{k,j}$
end for
$x_{sk+s} = V_k x'_{k,s} + x_{sk}$, $r_{sk+s} = V_k r'_{k,s}$, $p_{sk+s} = V_k p'_{k,s}$
end for
return x_{sk+s}

2.2. Deflated conjugate gradient method. Our CA-D-CG is based on D-CG by Saad et al. [45], shown in Algorithm 2.3 for reference. (As mentioned above, this was not the first appearance of deflated CG in the literature.)

We now summarize the motivation for the use of eigenvalue deflation in the CG method as presented in [45]. It is well-known (see, e.g., [23]) that in exact arithmetic, after m iterations of CG, the error is (loosely) bounded by

$$\|x - x_m\|_A \leq 2\|x - x_0\|_A \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^m,$$

with $\kappa(A) = \lambda_n/\lambda_1$, where $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ are the eigenvalues of the SPD matrix A. The authors of [45] prove that for some set of linearly independent vectors $W = [w_1, w_2, \ldots, w_c]$, D-CG applied to $Ax = b$ is mathematically equivalent to CG applied to the positive semidefinite system $H^T AH\tilde{x} = H^T b$, where $H = I - W(W^T AW)^{-1}(AW)^T$ is the A-orthogonal projection onto $W^{\perp_A}$ and $x = H\tilde{x}$. When the columns of W are exact eigenvectors associated with $\lambda_1, \ldots, \lambda_c$, the *effective* condition number (see [19]) is $\kappa_{\text{eff}}(H^T AH) = \lambda_n/\lambda_{c+1}$. When the columns of W are approximate eigenvectors associated with $\lambda_1, \ldots, \lambda_c$, one can expect that $\kappa_{\text{eff}}(H^T AH) \approx \lambda_n/\lambda_{c+1}$. Thus if $\lambda_{c+1} > \lambda_1$, deflation decreases the effective condition number of the system, thus theoretically improving the bounds on the (exact arithmetic) convergence rate.

We note that it is well-known that in reality, the convergence of the conjugate gradient method is much more accurately described by considering the spacing between eigenvalues of the matrix A, and even these more descriptive bounds do not hold in finite precision [32]. However, the reduction of the effective condition number described above nonetheless remains the motivation behind deflation and in practice can lead to an improved convergence rate in many cases. We also note that there has been recent work in developing tighter bounds on the rate of convergence in the deflated conjugate gradient method; see, e.g., [29].

For consistency, we assume we have an initial guess x_0 such that $r_0 = b - Ax_0 \perp W$. To satisfy this initial requirement, one can choose $x_0 = x_{-1} + W(W^T AW)^{-1}W^T r_{-1}$, where x_{-1} is arbitrary and $r_{-1} = b - Ax_{-1}$. Note that the selection of the subspace W is out of the scope of this paper. This topic is covered extensively in the literature; see, e.g., [1, 3, 9, 15, 37, 38, 47, 52].

ALGORITHM 2.3. Deflated Conjugate Gradient (D-CG).

Require: Approximate solution x_{-1} to $Ax = b$ with residual $r_{-1} = b - Ax_{-1}$; n-by-c matrix W of rank c
Compute and factorize $W^T AW$
$x_0 = x_{-1} + W(W^T AW)^{-1}W^T r_{-1}$, $r_0 = b - Ax_0$
$\mu = (W^T AW)^{-1}W^T Ar_0$, $p_0 = r_0 - W\mu$
for $m = 0, 1, \ldots$ until convergence **do**
$\alpha_m = (r_m^T r_m)/(p_m^T Ap_m)$

6: $x_{m+1} = x_m + \alpha_m p_m$
7: $r_{m+1} = r_m - \alpha_m A p_m$
8: $\beta_{m+1} = (r_{m+1}^T r_{m+1})/(r_m^T r_m)$
9: Solve $W^T A W \mu_{m+1} = W^T A r_{m+1}$ for μ_{m+1}
10: $p_{m+1} = r_{m+1} + \beta_{m+1} p_m - W \mu_{m+1}$
11: **end for**
12: **return** x_{m+1}

3. Deflated communication-avoiding conjugate gradient method. We now derive CA-D-CG based on D-CG (Algorithm 2.3). As before, we denote the iteration m in Algorithm 2.3 with $m = sk + j$ to distinguish inner and outer loop iterations. By induction on lines 6, 7, and 10 of Algorithm 2.3, we can write

$$p_{sk+j}, r_{sk+j} \in \mathcal{K}_{s+1}(A, p_{sk}) + \mathcal{K}_s(A, r_{sk}) + \mathcal{K}_{s-1}(A, W), \tag{3.1}$$

$$x_{sk+j} - x_{sk} \in \mathcal{K}_s(A, p_{sk}) + \mathcal{K}_{s-1}(A, r_{sk}) + \mathcal{K}_{s-2}(A, W), \tag{3.2}$$

for $0 \le j \le s$. Deflation also requires the product $A r_{sk+j+1}$ in the computation of μ_{sk+j+1} in line 9. Again, by induction, we can write

$$A r_{sk+j+1} \in \mathcal{K}_{s+2}(A, p_{sk}) + \mathcal{K}_{s+1}(A, r_{sk}) + \mathcal{K}_s(A, W). \tag{3.3}$$

As before, we define matrices P_k and R_k whose columns span the Krylov subspaces $\mathcal{K}_{s+2}(A, p_{sk})$ and $\mathcal{K}_{s+1}(A, r_{sk})$, respectively. For deflation, we now also require a basis $\mathcal{W}$ for $\mathcal{K}_s(A, W)$. Note that, assuming W does not change throughout the iteration, $\mathcal{W}$ needs only be computed once. For the deflated method, we now define the n-by-$(2s+3+cs)$ matrix

$$\begin{aligned} V_k &= [P_k,\ R_k, \mathcal{W}] \\ &= [\rho_0(A)p_{sk},\ \ldots,\ \rho_{s+1}(A)p_{sk},\ \rho_0(A)r_{sk},\ \ldots,\ \rho_s(A)r_{sk},\ \rho_0(A)W,\ \ldots,\ \rho_{s-1}(A)W], \end{aligned}$$

where ρ_i is defined as in (2.1). By (3.2), (3.1), and (3.3), we can then write, $0 \le j \le s$, $p_{sk+j} = V_k p'_{k,j}$, $r_{sk+j} = V_k r'_{k,j}$, and $x_{sk+j} - x_{sk} = V_k x'_{k,j}$, i.e., the length-$(2s+3+cs)$ vectors $p'_{k,j}$, $r'_{k,j}$, and $x'_{k,j}$ are coordinates for p_{sk+j}, r_{sk+j}, and $x_{sk+j} - x_{sk}$, respectively, in terms of the columns of V_k.

As in CA-CG, we can write a recurrence for computing multiplications with A, that is, for $0 \le j < s$,

$$A p_{sk+j} = A V_k p'_{k,j} = V_k B_k p'_{k,j} \quad \text{and} \quad A r_{sk+j+1} = A V_k r'_{k,j+1} = V_k B_k r'_{k,j+1},$$

where, for the deflated method, we now define block diagonal matrix

$$B_k = \begin{bmatrix} [C_{k,s+2} \quad 0_{s+2,1}] & & \\ & [C_{k,s+1} \quad 0_{s+1,1}] & \\ & & [C_{k,s} \otimes I_c \quad 0_{cs,1}] \end{bmatrix}.$$

Thus, a multiplication by B_k in the basis V_k is equivalent to a multiplication by A in the standard basis.

Assuming the same W is used throughout the iterations, $W^T A W$ can be precomputed and factorized offline. The small c-by-c factors of $W^T A W$ are assumed to fit in local/fast memory. If we compute the $(2s+3+cs)$-by-$(2s+3+cs)$ matrix $G_k = V_k^T V_k$ and extract the c-by-$(2s+3+cs)$ submatrix $Z_k = W^T V_k$, then we can form the right-hand side in the solve for μ_{sk+j+1} in line 9 of Algorithm 2.3 by $W^T Ar_{sk+j+1} = Z_k B_k r'_{k,j+1}$, replacing a global reduction with a small, local operation. Note that the formulas for computing α_{sk+j} and β_{sk+j+1} in Algorithm 2.3 remain the same as in Algorithm 2.1. Thus, using G_k, we can compute these inner products in CA-D-CG using the same formulas as in CA-CG (lines 7 and 10 of Algorithm 2.2).

Similarly, the formulas for the updates x_{sk+j+1} and r_{sk+j+1} are the same for D-CG and CG, so the formulas for $x'_{k,j+1}$ and $r'_{k,j+1}$ in CA-D-CG remain the same as those in CA-CG (lines 8 and 9). The formula for p_{sk+j+1} in D-CG can be written as

$$V_k p'_{k,j+1} = V_k r'_{k,j+1} + \beta_{sk+j+1} V_k p'_{k,j} - V_k [0^T_{2s+3},\, \mu^T_{k,j+1},\, 0^T_{c(s-1)}]^T,$$

for $0 \leq j \leq s-1$. Thus, in CA-D-CG, $p'_{k,j+1}$ is updated by

$$p'_{k,j+1} = r'_{k,j+1} + \beta_{sk+j+1} p'_{k,j} - [0^T_{2s+3},\, \mu^T_{k,j+1},\, 0^T_{c(s-1)}]^T.$$

The resulting CA-D-CG method is shown in Algorithm 3.1.

ALGORITHM 3.1. Deflated Communication-Avoiding Conjugate Gradient (CA-D-CG).

Require: Approximate solution x_{-1} to $Ax = b$ with residual $r_{-1} = b - Ax_{-1}$; n-by-c matrix W of rank c

Compute and factorize $W^T AW$
Compute $\mathcal{W}$
$x_0 = x_{-1} + W(W^T AW)^{-1} W^T r_{-1}$, $r_0 = b - Ax_0$
$\mu = (W^T AW)^{-1} W^T Ar_0$, $p_0 = r_0 - W\mu$
for $k = 0, 1, \ldots$ until convergence **do**
  Compute P_k, R_k, let $V_k = [P_k, R_k, \mathcal{W}]$; assemble B_k.
  $G_k = V_k^T V_k$; extract $Z_k = W^T V_k$
  $p'^T_{sk} = [1, 0^T_{2s+2+cs}]$, $r'^T_{sk} = [0^T_{s+2}, 1, 0^T_{s+cs}]$, $x'_{sk} = 0_{2s+3+cs}$
  for $j = 0, \ldots, s-1$ **do**
    $\alpha_{sk+j} = (r'^T_{k,j} G_k r'_{k,j}) / (p'^T_{k,j} G_k B_k p'_{k,j})$
    $x'_{k,j+1} = x'_{k,j} + \alpha_{k,j} p'_{k,j}$
    $r'_{k,j+1} = r'_{k,j} - \alpha_{k,j} B_k p'_{k,j}$
    $\beta_{sk+j+1} = (r'^T_{k,j+1} G_k r'_{k,j+1}) / (r'^T_{k,j} G_k r'_{k,j})$
    Solve $W^T AW \mu_{k,j+1} = Z_k B_k r'_{k,j+1}$ for $\mu_{k,j+1}$
    $p'_{k,j+1} = r'_{k,j+1} + \beta_{sk+j+1} p'_{k,j} - [0^T_{2s+3}, \mu^T_{k,j+1},\, 0^T_{c(s-1)}]^T$
  end for
  $x_{sk+s} = V_k x'_{k,s} + x_{sk}$, $r_{sk+s} = V_k r'_{k,s}$, $p_{sk+s} = V_k p'_{k,s}$
end for
return x_{sk+s}

3.1. Algorithmic extensions. We assume in our derivation that the matrix of the deflation vectors W is constant through the iterations. We could, however, extend CA-D-CG to allow for updating of W_k in (some or all) outer loop iterations k; see, e.g., [1, 3, 33, 42, 47] for example applications. (Additional considerations arise when changing the operator during the iterations due to the loss of orthogonality properties [2, Chapter 12]; see also [40].) Updating W_k in the outer loop k requires recomputing $\mathcal{W}_k$, a basis for $\mathcal{K}_s(A, W_k)$. This computation could potentially be fused with the computation of P_k and R_k such that no extra latency cost is incurred. The quantity $W_k^T AW_k$ can be recovered from the computation of G_k, so no additional communication is required. A factorization of the c-by-c matrix $W_k^T AW_k$ can also be performed locally. Note the number of deflation vectors c could be allowed to vary over outer loop iterations as well. This extension is considered future work.

ETNA
Kent State University
http://etna.math.kent.edu

4. Numerical experiments. For the numerical experiments, our goal is to show that CA-D-CG is competitive with D-CG in terms of the convergence rate. While the approaches are equivalent in exact arithmetic, there is no reason to expect that the CA-D-CG iterates will exactly equal the D-CG iterates in finite precision, given the different sets of floating-point operations performed. There are many open questions about CA-KSMs' finite precision behavior, some of which we hope to address in future work. But in lieu of theoretical results, we will rely on our practical experience that CA-KSMs' iterates deviate from their classical counterparts as the s-step Krylov bases become ill-conditioned (increasingly with s), and this effect can be diminished by picking different polynomial bases [6, 27, 43]. To focus on this potential instability that grows with s, we chose a test problem for which the classical methods are relatively insensitive to rounding errors. Thus, our experiments do not address the possibility that the deviation between the D-CG and CA-D-CG iterates is much larger when the convergence of the classical methods is highly perturbed by rounding errors.

We test the stability of our reformulation on a similar model problem to the one considered in [45] using codes written in a combination of MATLAB and C with linear algebra routines from Intel's Math Kernel Library. We generate a discrete 2D Laplacian by `gallery('poisson',512)` in MATLAB, so A is an SPD matrix of order $n = 512^2$. We pick the right-hand side b equal to A times the vector with entries all $n^{-1/2}$. Our deflation vectors are the eigenvectors corresponding to the eigenvalues of smallest magnitude computed using MATLAB's `eigs`. Note that the study in [45] used (known) exact eigenvalues; this difference does not significantly affect the results for this test.

In Figures 4.1–4.3, we compare convergence for the model problem using D-CG and CA-D-CG with the monomial, Newton, and Chebyshev polynomial basis, respectively, each for a few representative s values. We report the 2-norm of the true residual computed by $b - Ax_m$ rather than the recursively updated residual r_m and normalize by the 2-norm of the starting residual $r_0 = b$ (i.e., the starting guess x_0 is the vector of zeros). We declare convergence after a reduction by a factor of 10^8 in the normalized residual 2-norm. The solid curves correspond to D-CG, and circles correspond to CA-D-CG. We deflate with $c \in \{0, 4, 8\}$ eigenvectors, plotted in black, red, and blue, respectively (when $c = 0$, D-CG is just CG and CA-D-CG is just CA-CG). Based on the formulas above, this suggests that the condition number $\kappa(A) \approx 1.07 \cdot 10^5$ in the undeflated case ($c = 0$) should improve to $\approx 2.13 \cdot 10^4$ in the case $c = 4$ and to $\approx 1.25 \cdot 10^4$ when $c = 8$.

We implemented the Newton basis by choosing parameters in (2.3) as $\sigma_i = 0$, $\gamma_i = 1$, and θ_i is the i-th element in a set of Leja-ordered points on the real line segment $[\lambda_{c+1}, \lambda_n]$; see, e.g., [43]. We implemented the Chebyshev basis by setting the basis parameters in (2.3) as $\gamma_i = |\lambda_n - \lambda_{c+1}|/2$ (except γ_0, which is not divided by 2), $\theta_i = \lambda_{c+1} + |\lambda_n - \lambda_{c+1}|/2$, and $\sigma_i = |\lambda_n - \lambda_{c+1}|/8$. These recurrence coefficients are based on the bounding ellipse of the spectrum of A, which is, in the present case of a symmetric matrix A, an interval on the real line; see, e.g., [28]. In practice, only a few Ritz values (estimates for the eigenvalues of A) need to be computed up front to sufficiently determine the parameters for the Newton or Chebyshev polynomials. One can also incorporate information about new Ritz values obtained as a byproduct of the iterations to improve the basis conditioning; see [43] for practical details and experiments.

Note that λ_{c+1} is used as the smallest eigenvalue in selecting the Newton and Chebyshev parameters above. This is because if the columns of W are the exact eigenvectors of A corresponding to the eigenvalues $\lambda_1, \ldots, \lambda_c$, then using λ_1 as a basis parameter in the computation of the basis $\mathcal{W}$ can cause cancellation and can thus produce a rank-deficient basis. Although this cancellation does not occur in the computation of the bases P_k and R_k, we used the same

ETNA
Kent State University
http://etna.math.kent.edu

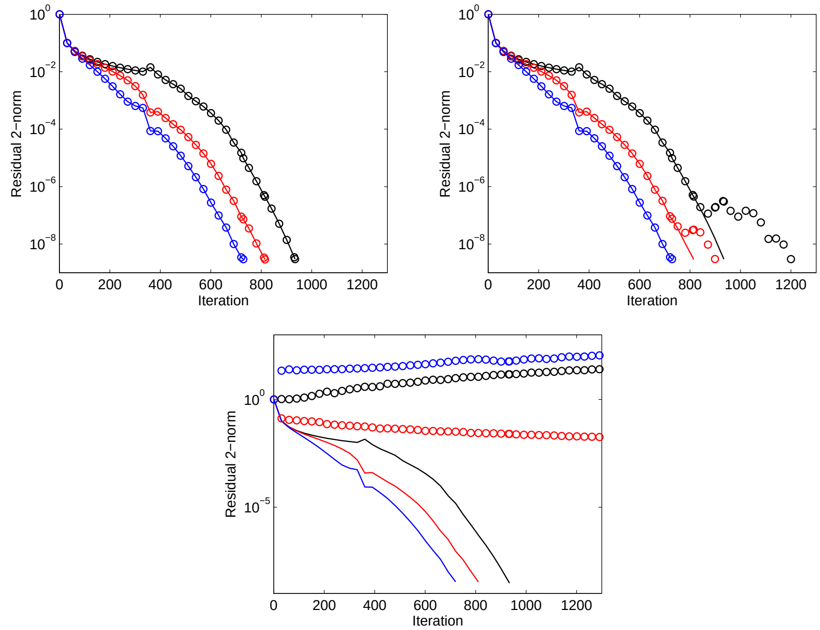

FIG. 4.1. *Monomial basis tests. Top: $s = 4$ (left), $s = 8$ (right), bottom: $s = 16$. Plots show the 2-norm of the true residual $b - Ax_m$ for tests with $c = 0$ (black), $c = 4$ (red), and $c = 8$ (blue) for both D-CG ($-$) and CA-D-CG ($\circ$) using monomial bases of size s. Note that the y-axis in the bottom plot differs.*

basis parameters chosen for $\mathcal{W}$ (i.e., using λ_{c+1}) to compute P_k and R_k for simplicity with no ill effects.

For the monomial basis (Figure 4.1), convergence is nearly identical for $s = 4$, but we begin to see a delay in the convergence of CA-D-CG for $s = 8$ (top-left) and a failure to converge by $s = 16$. For the Newton basis (Figure 4.2), the two methods have similar convergence behavior past $s = 16$; only around $s = 100$ (bottom-right) we begin to notice a significant delay in convergence for CA-D-CG. The situation is similar for the Chebyshev basis (Figure 4.3); only the bottom-right figure now depicts the case $s = 220$. These results clearly demonstrate that the basis choice plays an important role for the convergence of CA-D-CG, at least on this well-behaved model problem. In the next section, we will introduce a coarse performance model to ask about the practical benefits of values as large as $s = 220$.

5. Performance modeling. In this section, we give a qualitative description of the performance tradeoffs between the four KSMs mentioned above—CG, CA-CG, and their deflated counterparts—on massively parallel machines. For CA-CG and CA-D-CG, we estimate the time for s inner loop iterations and then divide by s to estimate the time per iteration. Note that this ignores relative rates of convergence treated in Section 4. We were motivated to develop CA-D-CG based on the high relative cost of interprocessor communication on large-scale parallel computers. In addition to parallel implementations, CA-KSMs can avoid communication on sequential machines, namely data movement within the memory hierarchy. Indeed, the parallel and sequential approaches naturally compose hierarchically as has been exploited in previous high-performance CA-KSM implementations [36], and we suggest the same for a future CA-D-CG implementation. However, in this section, we will restrict ourselves to a parallel model which ignores sequential communication costs to illustrate the changes in parallel communication costs. We do not claim that this model's predicted

ETNA
Kent State University
http://etna.math.kent.edu

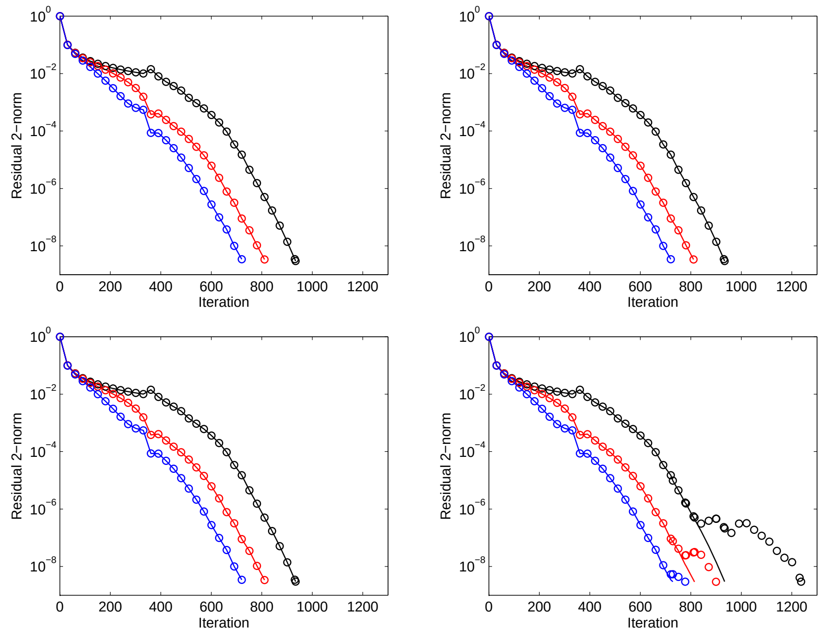

FIG. 4.2. *Newton basis tests. Top: $s = 4$ (left), $s = 8$ (right), bottom: $s = 16$ (left) $s = 100$ (right). Plots show the 2-norm of the true residual $b - Ax_m$ for tests with $c = 0$ (black), $c = 4$ (red), and $c = 8$ (blue) for both D-CG $(-)$ and CA-D-CG $(\circ)$ using Newton bases of size s.*

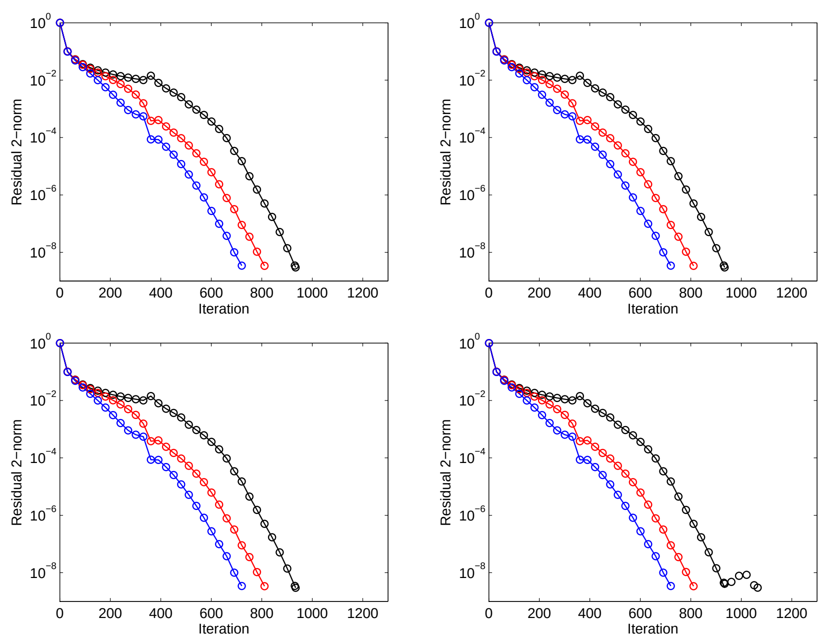

FIG. 4.3. *Chebyshev basis tests. Top: $s = 4$ (left), $s = 8$ (right), bottom: $s = 16$ (left), $s = 220$ (right). Plots show the 2-norm of the true residual $b - Ax_m$ for tests with $c = 0$ (black), $c = 4$ (red), and $c = 8$ (blue) for both D-CG $(-)$ and CA-D-CG $(\circ)$ using Chebyshev bases of size s.*

ETNA
Kent State University
http://etna.math.kent.edu

speedups are always attainable on real hardware. However, we believe that such models can help to detect the feasibility of a communication-avoiding approach relative to a classical approach as well as to efficiently explore and prune the (machine-specific) parameter tuning space for an optimized implementation.

5.1. Machine model. We model parallel computation as a fully connected network of p homogeneous processors that perform local computations and exchange point-to-point messages. Each processor executes asynchronously and can send or receive at most one message at a time. Passing an n-word message takes $\alpha+\beta n$ seconds on both the sending and receiving processors, which cannot perform computation during this time. The constant α represents a latency cost incurred by every message, while β represents a bandwidth cost linear in the message size. There is no notion of distance on the network, and we assume the network has unbounded buffer capacity. While this simple model's assumptions are not all realistic, similar models are widely used to analyze communication costs on distributed-memory machines; see, e.g., [8]. One could refine this model to obtain, e.g., the LogP model [13], which distinguishes between network latency, software overhead, and network injection bandwidth (blurred between our α and β terms), allows overlap of communication and computation, and introduces constraints on the message size and the network congestion. We quantify the computation time in terms of the floating-point operations (flops) performed: a processor can only operate on data residing in its local memory (of unbounded capacity), and each flop takes γ seconds. If a processor performs F flops and sends/receives S messages containing a total of W words, then we model its runtime as $\gamma F+\alpha S+\beta W$. This is a poor cost model for certain programs like the one where the p processors relay a value from processor 1 to processor p: each processor sends/receives at most 2 words, but the actual runtime grows linearly in p. To count correctly in such situations, one can consider the runtime along critical paths, e.g., in a program activity graph [54]. For our algorithms here, we will only consider certain balanced parallelizations where one processor is always the slowest, so we can simply count F, S, W for that processor to bound the total runtime.

Our assumptions that each processor has unbounded local memory and can execute each flop at the peak rate $1/\gamma$ may be unrealistic when the neglected sequential costs are nontrivial. However, when considering large p and small local problems and when performance is dominated by interprocessor communication, we expect that the sequential costs would not significantly increase our models' estimated costs.

We consider two parallel machine models, which we call 'Exa' and 'Grid.' We use $\gamma = 1 \cdot 10^{-13}$ seconds per (double precision) flop in both cases based on predictions for a 'node' of an exascale machine [7, 50]. This flop rate corresponds to a node with a 1024-core processor and its own memory hierarchy with 256 GB of capacity at the last level. However, as discussed above, we ignore this intranode structure. For Exa, the interconnect has parameters $\alpha = 4 \cdot 10^{-7}$ seconds per message and $\beta = 3.7 \cdot 10^{-11}$ seconds per word (4-byte double precision value). For the second machine, Grid, we replace this interconnect by the Internet (via Ethernet) using the parameters $\alpha = 10^{-1}$ and $\beta = 2.5 \cdot 10^{-8}$ given in [35]. In contrast to their predecessors, our models allow an arbitrary number of processors in order to illustrate the asymptotic scaling behavior—we do not claim that every machine configuration modeled is physically realizable.

5.2. Experiments. We assume the same model problem (5-point 2D stencil) and deflation vectors as in the numerical experiments. However, since we are not modeling convergence here, the actual (nonzero) values do not influence the performance model. We assume the $\sqrt{n}$-by-$\sqrt{n}$ mesh is partitioned across a $\sqrt{p}$-by-$\sqrt{p}$ grid of processors, so that each processor owns a contiguous $\sqrt{n/p}$-by-$\sqrt{n/p}$ subsquare, and we assume these fractions are

ETNA
Kent State University
http://etna.math.kent.edu

integers. This layout minimizes communication within a factor of $\sqrt{2}$ (for this particular stencil, a diamond layout would be asymptotically optimal [4, Chapter 4.8]). We summarize the S, W, F costs for the four algorithms in Appendix A. To simplify the analysis, we restrict $s \in \{1, \ldots, \sqrt{n/p}\}$ for the CA-KSMs, which means that the sparse computations only require communication with the (logical) nearest neighbors. The communication-avoiding approach is correct for any s, but the latency cost rises sharply when each processor needs information from a larger neighborhood. We also simplify by using the same blocking parameter s for the sparse and dense computations. In general, one can compute the s-step Krylov bases in smaller blocks and then compute a (larger) Gram matrix, or vice versa, i.e., constructing the Gram matrix blockwise as the s-step bases are computed. In practice, we have observed significant speedups from the Gram matrix construction alone (with no blocking of the SpMV operations) [53], and we suggest tuning the block sizes independently. Also to simplify the analysis, we ignore the preprocessing costs of computing the deflation matrix W (not the algorithmic costs) and computing and factorizing W^TAW, assuming that they can be amortized over many iterations. In practice, these costs may not be negligible especially if the number of iterations is small.

We first consider weak scaling in Figure 5.1. We fix $n/p = 4^6$ and vary the grid parameter $p \in \{4^x : x \in \{2, \ldots, 14\}\}$. The black curves correspond to the runtime of a single iteration of the classical KSMs: CG (no markers), D-CG with $c = 4$ (square markers), and D-CG with $c = 8$ (asterisk markers); the logarithmic dependence on p, due to the collective communications, is evident. The red curves allow us to vary the parameter $s \in \{1, \ldots, \sqrt{n/p}\}$, where $s = 1$ corresponds to the classical KSMs and $s > 1$ corresponds to their deflated counterparts (markers mean the same as for the black curves). For comparison with the classical methods, we compute the runtime of one CA-KSM outer loop (with s inner-loop iterations) and then divide by s. On both Exa and Grid, it was beneficial to pick $s > 1$ for every problem although the optimal s varies as illustrated in Figure 5.3. The best speedups, i.e., the ratio of the runtime with $s = 1$ to the best runtime with $s \geq 1$, were about 55, 38, and 28 for $c = 0$, 4, and 8, respectively, on Exa, while the corresponding best speedups on Grid were about 116, 174, and 173.

We now consider strong scaling, presented in Figure 5.2. The curves represent the same algorithms as in the previous figure, except that now we use different problems for the two machines (we use the same range of p as before). Note that the red and black curves coincide for some points on the left of both plots. As the local problem size decreases, so does the range of s values over which the CA-KSMs optimize. For Exa, we fix $n = 4^{15}$, so for the largest p, for instance, the processors' subsquares are 2-by-2 and $s \in \{1, 2\}$; for Grid, we fix $n = 4^{22}$. While all tested KSMs scale when the local problem is large, the CA-KSMs are able to exploit more parallelism than the classical KSMs on both machines. In both cases, the CA-KSM runtime eventually begins to increase, too. The best speedups on Exa were about 49, 42, and 31 for $c = 0$, 4, and 8, respectively, while the corresponding best speedups on Grid were about 1152, 872, and 673.

Lastly, in Figure 5.3, we demonstrate the benefits of increasing the parameter s for a fixed problem and a varying number $c \in \{0, \ldots, 50\}$ of deflation vectors. The case $c = 0$ indicates the non-deflated KSMs and is depicted separately. We plot the CA-KSMs' speedups relative to the classical KSMs, i.e., the points along the line $s = 1$. For both machines, we fix $p = 4^9$, but to illustrate the tradeoffs on both, we pick $n = 4^{14}$ for Exa and $n = 4^{20}$ for Grid. In both cases, we see decreased relative benefits of avoiding communication as c increases as the network bandwidth becomes saturated by the larger reductions. For Exa, for small c it is beneficial to increase s to the maximum $\sqrt{n/p}$ we consider; for Grid, however, it is never beneficial to increase s to its maximum for any c.

ETNA
Kent State University
http://etna.math.kent.edu

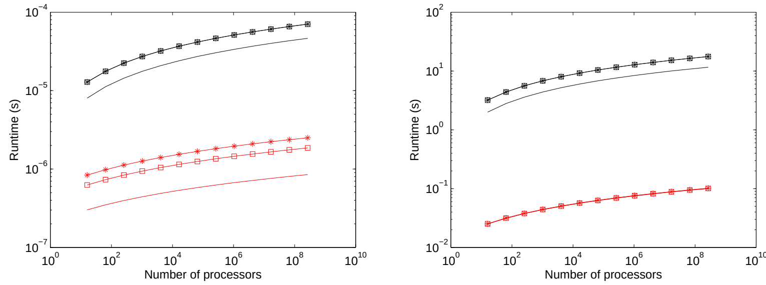

FIG. 5.1. *Modeled weak scaling for a model problem on Exa (left) and Grid (right). Black curves correspond to the runtime of a single iteration of the classical KSMs: CG (no markers), D-CG with $c = 4$ (square markers), and D-CG with $c = 8$ (asterisk markers). Red curves correspond to the runtime of a single iteration of the CA-KSMs: CA-CG (no markers), CA-D-CG with $c = 4$ (square markers), and CA-D-CG with $c = 8$ (asterisk markers) using the optimal value of $s \in \{1, \ldots, \sqrt{n/p}\}$ for each point.*

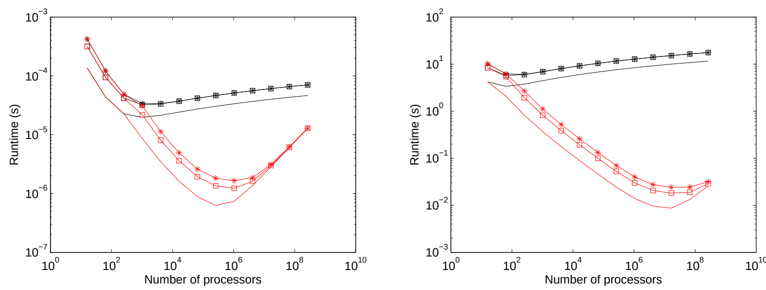

FIG. 5.2. *Modeled strong scaling for a model problem on Exa (left) and Grid (right). Black curves correspond to the runtime of a single iteration of the classical KSMs: CG (no markers), D-CG with $c = 4$ (square markers), and D-CG with $c = 8$ (asterisk markers). Red curves correspond to the runtime of a single iteration of the CA-KSMs: CA-CG (no markers), CA-D-CG with $c = 4$ (square markers), and CA-D-CG with $c = 8$ (asterisk markers) using the optimal value of $s \in \{1, \ldots, \sqrt{n/p}\}$ for each point.*

6. Future work and conclusions. In this work, we have demonstrated that deflation can be performed in a communication-avoiding way and is thus suitable for the use as a preconditioner for CA-KSMs. We derived CA-D-CG, which is equivalent to D-CG in exact arithmetic but can be implemented such that parallel latency is reduced by a factor of $O(s)$ over a fixed number of iterations. Performance modeling shows predicted speedups of CA-D-CG over D-CG for a number of n/p ratios on two model architectures for s values constrained to $s \leq \sqrt{n/p}$. We performed numerical experiments for a model problem to illustrate the benefits of deflation for the convergence rate. Our results also demonstrate that by using better conditioned bases of Newton and Chebyshev polynomials, s can be made very large before the convergence behavior of CA-D-CG deviates significantly from D-CG. However, for more difficult problems to be studied in future work, we expect the practical range of s to be more restricted.

We also point out that, as in the classical case, our CA-D-CG method is mathematically equivalent to applying CA-CG (Algorithm 2.2) to the transformed system $H^T A H \tilde{x} = H^T b$. If we were to instead perform deflation in this way, communication-avoiding techniques related to blocking covers for linear relaxation [30, 31] and other ideas from those works could be applied in the Krylov basis computation.

ETNA
Kent State University
http://etna.math.kent.edu

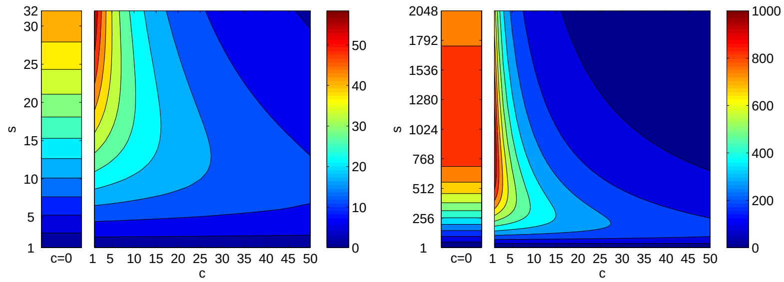

FIG. 5.3. *Modeled speedup per iteration for the model problem versus s and c on Exa (left) and Grid (right).*

The communication-avoiding reorganization applied here can also be applied to many other deflated KSMs including adaptive deflation approaches (see, e.g., [1, 3, 33, 42, 47]), where the matrix W is allowed to change. Our future work will address these applications, as well as a distributed-memory implementation to evaluate the performance of our approaches on real parallel machines.

Acknowledgments. This material is based upon work supported by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program under Award Numbers DE-SC0004938, DE-SC0003959, and DE-SC0010200, by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research, X-Stack program under Award Numbers DE-SC0005136, DE-SC0008699, DE-SC0008700, and AC02-05CH11231, by DARPA Award HR0011-12-2-0016, as well as contributions from Intel, Oracle, and MathWorks.

Appendix A: Complexity analysis. We can identify the elements of the vector iterates with vertices on a $\sqrt{n}$-by-$\sqrt{n}$ 2D mesh. As explained above, each processor is assigned a $\sqrt{n/p}$-by-$\sqrt{n/p}$ subsquare. The matrix A for the model problem is a stencil with constant coefficients and can be represented in $O(1)$ words. In the case of variable coefficients, we would partition A in a overlapping block rowwise fashion as explained in [35]. The number of flops, words moved, and messages required for s steps of CG, CA-CG, D-CG, and CA-D-CG are as follows:

$$\begin{aligned}
\text{Flops}_{\text{CG}} &= s(19n/p + 2\log_2 p)\\
\text{Words}_{\text{CG}} &= s(4\sqrt{n/p} + 4\log_2 p)\\
\text{Mess}_{\text{CG}} &= s(4\log_2 p + 4)\\
\text{Flops}_{\text{CA-CG}} &= 18(n/p)s + s(20s + 3(2s+1)(4s+1) + 10) + 12s^3\\
&\quad + 2(n/p)(4s+1) + (n/p)(4s+3) + 36\sqrt{n/p}s^2\\
&\quad + ((2s+1)(2s+2)(2(n/p) + \log_2 p - 1))/2\\
\text{Words}_{\text{CA-CG}} &= 8\sqrt{n/p}s + 4s^2 + \log_2 p(2s+1)(2s+2)\\
\text{Mess}_{\text{CA-CG}} &= 2\log_2 p + 8\\
\text{Flops}_{\text{D-CG}} &= s(30(n/p) + 2\log_2 p + c(2(n/p) + \log_2 p - 1) + 2c^2 + (n/p)(2c-1))\\
\text{Words}_{\text{D-CG}} &= s((4+2c)\log_2 p + 8\sqrt{n/p})\\
\text{Mess}_{\text{D-CG}} &= s(6\log_2 p + 8)
\end{aligned}$$

ETNA
Kent State University
http://etna.math.kent.edu

$$\begin{aligned}
\text{Flops}_{\text{CA-D-CG}} &= 12(s+1)^3 + 2(n/p)(4s+2cs+5) + (n/p)(4s+2cs+7) \\
&\quad + 36\sqrt{n/p}(s+1)^2 + 18(n/p)(s+1) + s(24s + c(4s+2cs+5) \\
&\quad + 4(2s+cs+3)(4s+2cs+5) + 12cs + 2c^2 + 36) \\
&\quad + (2s+3)(s+cs+2)(2(n/p) + \log_2 p - 1) \\
\text{Words}_{\text{CA-D-CG}} &= 4(s+1)^2 + 8\sqrt{n/p}(s+1) + 2\log_2 p(2s+3)(s+cs+2) \\
\text{Mess}_{\text{CA-D-CG}} &= 2\log_2 p + 8
\end{aligned}$$

For the CA-KSMs, we exploit neither the symmetry of G_k nor the nonzero structure of B_k and the length-$O(s)$ coefficient vectors.

For D-CG, we note that one can compute AW offline (in line 1) and avoid the SpMV Ar_{m+1} in line 9. While this may improve some constant factors by up to 2, it does not avoid the global reduction in the subsequent application of $(AW)^T$, which our performance modeling suggests is often the dominant cost.

We note that the $\log_2 p$ terms in the computation and bandwidth costs can often be reduced by exploiting efficient collectives based on recursive halving/doubling approaches; see [8] for a survey. These approaches require that the number of words in the collective is at least p, which was not always true in our experiments, hence our use of simpler tree-based collectives.

REFERENCES

[1] A. M. Abdel-Rehim, R. B. Morgan, D. A. Nicely, and W. Wilcox, *Deflated and restarted symmetric Lanczos methods for eigenvalues and linear equations with multiple right-hand sides*, SIAM J. Sci. Comput., 32 (2010), pp. 129–149.

[2] O. Axelsson, *Iterative Solution Methods*, Cambridge University Press,Cambridge, 1994.

[3] J. Baglama, D. Calvetti, G. H. Golub, and L. Reichel, *Adaptively preconditioned GMRES algorithms*, SIAM J. Sci. Comput., 20 (1998), pp. 243–269.

[4] R. H. Bisseling, *Parallel Scientific Computation*, Oxford University Press, Oxford, 2004.

[5] E. Boman, M. Heroux, M. Hoemmen, S. Rajamanickam, S. Tomov, and I. Yamazaki, *Domain decomposition preconditioners for communication-avoiding Krylov methods on a hybrid CPU/GPU cluster*, in Proc. ACM/IEEE Conference on Supercomputing, to appear, 2014.

[6] E. Carson, N. Knight, and J. Demmel, *Avoiding communication in nonsymmetric Lanczos-based Krylov subspace methods*, SIAM J. Sci. Comput., 35 (2013), pp. S42–S61.

[7] C. Chan, D. Unat, M. Lijewski, W. Zhang, J. Bell, and J. Shalf, *Software design space exploration for exascale combustion co-design*, in Supercomputing, J. Kunkel, T. Ludwig, and H. Meuer, eds., vol. 7905 of Lecture Notes in Comput. Sci., Springer, Berlin, 2013, pp. 196–212.

[8] E. Chan, M. Heimlich, A. Purkayastha, and R. van de Geijn, *Collective communication: theory, practice, and experience*, Concurrency Computat.: Pract. Exper., 19 (2007), pp. 1749–1783.

[9] A. Chapman and Y. Saad, *Deflated and augmented Krylov subspace techniques*, Numer. Linear Algebra Appl., 4 (1997), pp. 43–66.

[10] A. T. Chronopoulos and C. W. Gear, *On the efficient implementation of preconditioned s-step conjugate gradient methods on multiprocessors with memory hierarchy*, Parallel Comput., 11 (1989), pp. 37–53.

[11] ———, *s-step iterative methods for symmetric linear systems*, J. Comput. Appl. Math., 25 (1989), pp. 153–168.

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

[13] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, and T. von Eicken, *LogP: towards a realistic model of parallel computation*, in Proc. 4th Symp. Principles Pract. Parallel Program., ACM, New York, 1993, pp. 1–12.

[14] E. de Sturler, *A performance model for Krylov subspace methods on mesh-based parallel computers*, Parallel Comput., 22 (1996), pp. 57–74.

[15] ———, *Truncation strategies for optimal Krylov subspace methods*, SIAM J. Numer. Anal., 36 (1999), pp. 864–889.

[16] J. DEMMEL, M. HOEMMEN, M. MOHIYUDDIN, AND K. YELICK, *Avoiding communication in computing Krylov subspaces*, Tech. Report UCB/EECS-2007-123, Dept. of EECS, Univ. of California, Berkeley, October 2007.

[17] Z. DOSTÁL, *Conjugate gradient method with preconditioning by projector*, Int. J. Comput. Math., 23 (1988), pp. 315–323.

[18] M. ENGELI, T. GINSBURG, H. RUTISHAUSER, AND E. STIEFEL, *Refined iterative methods for computation of the solution and the eigenvalues of self-adjoint boundary value problems*, Mitt. Inst. Angew. Math. Zürich. No., 8 (1959), (107 pages).

[19] J. FRANK AND C. VUIK, *On the construction of deflation-based preconditioners*, SIAM J. Sci. Comput., 23 (2001), pp. 442–462.

[20] D. GANNON AND J. VAN ROSENDALE, *On the impact of communication complexity on the design of parallel numerical algorithms*, IEEE Trans. Comput., 33 (1984), pp. 1180–1194.

[21] A. GAUL, M. H. GUTKNECHT, J. LIESEN, AND R. NABBEN, *A framework for deflated and augmented Krylov subspace methods*, SIAM J. Matrix Anal. Appl., 34 (2013), pp. 495–518.

[22] P. GHYSELS, W. VANROOSE, AND K. MEERBERGEN, *High performance implementation of deflated preconditioned conjugate gradients with approximate eigenvectors*, in Book of Abstracts of the Householder Symposium XIX, Spa, Belgium, 2014, pp. 84–85.

[23] G. GOLUB AND C. VAN LOAN, *Matrix Computations*, 4th ed., Johns Hopkins University Press, 2013.

[24] L. GRIGORI AND S. MOUFAWAD, *Communication avoiding ILU(0) preconditioner*, Tech. Report, Rapport de recherche RR-8266, INRIA, Paris-Rocquencourt, March 2013.

[25] M. H. GUTKNECHT, *Spectral deflation in Krylov solvers: a theory of coordinate space based methods*, Electron. Trans. Numer. Anal., 39 (2012), pp. 156–185.
http://etna.mcs.kent.edu/vol.39.2012/pp156-185.dir

[26] A. HINDMARSH AND H. WALKER, *Note on a Householder implementation of the GMRES method*, Tech. Report UCID-20899, Lawrence Livermore National Lab., Livermore, 1986.

[27] M. HOEMMEN, *Communication-Avoiding Krylov Subspace Methods*, PhD Thesis, Dept. of EECS, University of California, Berkeley, 2010.

[28] W. JOUBERT AND G. CAREY, *Parallelizable restarted iterative methods for nonsymmetric linear systems. Part I: theory*, Int. J. Comput. Math., 44 (1992), pp. 243–267.

[29] K. KAHL AND H. RITTICH, *Analysis of the deflated conjugate gradient method based on symmetric multigrid theory*, Preprint on arXiv.
http://arxiv.org/abs/1209.1963

[30] N. KNIGHT, E. CARSON, AND J. DEMMEL, *Exploiting data sparsity in parallel matrix powers computations*, in Parallel Processing and Applied Mathematics, R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Waśniewski, eds., vol. 8384 of Lecture Notes in Computer Science, Springer, Berlin, 2014, pp. 15–25.

[31] C. E. LEISERSON, S. RAO, AND S. TOLEDO, *Efficient out-of-core algorithms for linear relaxation using blocking covers*, J. Comput. System Sci., 54 (1997), pp. 332–344.

[32] J. LIESEN AND Z. STRAKOŠ, *Krylov Subspace Methods: Principles and Analysis*, Oxford University Press, Oxford, 2013.

[33] K. MEERBERGEN AND Z. BAI, *The Lanczos method for parameterized symmetric linear systems with multiple right-hand sides*, SIAM J. Matrix Anal. Appl., 31 (2009/10), pp. 1642–1662.

[34] G. MEURANT AND Z. STRAKOŠ, *The Lanczos and conjugate gradient algorithms in finite precision arithmetic*, Acta Numer., 15 (2006), pp. 471–542.

[35] M. MOHIYUDDIN, *Tuning Hardware and Software for Multiprocessors*, PhD Thesis, Dept. of EECS, University of California, Berkeley, May 2012.

[36] M. MOHIYUDDIN, M. HOEMMEN, J. DEMMEL, AND K. YELICK, *Minimizing communication in sparse matrix solvers*, in Proc. of ACM/IEEE Conference on High Performance Computing Networking, Storage and Analysis, 2009, IEEE Conference Proceedings, Los Alamitos, 2009, (12 pages).

[37] R. B. MORGAN, *Implicitly restarted GMRES and Arnoldi methods for nonsymmetric systems of equations*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1112–1135.

[38] ———, *GMRES with deflated restarting*, SIAM J. Sci. Comput., 24 (2002), pp. 20–37.

[39] R. A. NICOLAIDES, *Deflation of conjugate gradients with applications to boundary value problems*, SIAM J. Numer. Anal., 24 (1987), pp. 355–365.

[40] Y. NOTAY, *Flexible conjugate gradients*, SIAM J. Sci. Comput., 22 (2000), pp. 1444–1460.

[41] D. NUENTSA WAKAM AND J. ERHEL, *Parallelism and robustness in GMRES with the Newton basis and the deflated restarting*, Electron. Trans. Numer. Anal., 40 (2013), pp. 381–406.
http://etna.mcs.kent.edu/vol.40.2013/pp381-406.dir

[42] M. L. PARKS, E. DE STURLER, G. MACKEY, D. D. JOHNSON, AND S. MAITI, *Recycling Krylov subspaces for sequences of linear systems*, SIAM J. Sci. Comput., 28 (2006), pp. 1651–1674.

[43] B. PHILIPPE AND L. REICHEL, *On the generation of Krylov subspace bases*, Appl. Numer. Math., 62 (2012), pp. 1171–1186.

ETNA
Kent State University
http://etna.math.kent.edu

[44] Y. SAAD, *Iterative Methods for Sparse Linear Systems*, 2nd ed., SIAM, Philadelphia, 2003.
[45] Y. SAAD, M. YEUNG, J. ERHEL, AND F. GUYOMARC'H, *A deflated version of the conjugate gradient algorithm*, SIAM J. Sci. Comput., 21 (2000), pp. 1909–1926.
[46] V. SIMONCINI AND D. B. SZYLD, *Recent computational developments in Krylov subspace methods for linear systems*, Numer. Linear Algebra Appl., 14 (2007), pp. 1–59.
[47] A. STATHOPOULOS AND K. ORGINOS, *Computing and deflating eigenvalues while solving multiple right-hand side linear systems with an application to quantum chromodynamics*, SIAM J. Sci. Comput., 32 (2010), pp. 439–462.
[48] J. M. TANG, S. P. MACLACHLAN, R. NABBEN, AND C. VUIK, *A comparison of two-level preconditioners based on multigrid and deflation*, SIAM J. Matrix Anal. Appl., 31 (2009/10), pp. 1715–1739.
[49] S. TOLEDO, *Quantitative performance modeling of scientific computations and creating locality in numerical algorithms*, PhD Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, 1995.
[50] D. UNAT, *Personal communication*, 2013.
[51] J. VAN ROSENDALE, *Minimizing inner product data dependencies in conjugate gradient iteration*, Tech. Report 172178, ICASE-NASA, 1983.
[52] S. WANG, E. DE STURLER, AND G. H. PAULINO, *Large-scale topology optimization using preconditioned Krylov subspace methods with recycling*, Internat. J. Numer. Methods Engrg., 69 (2007), pp. 2441–2468.
[53] S. WILLIAMS, M. LIJEWSKI, A. ALMGREN, B. VAN STRAALEN, E. CARSON, N. KNIGHT, AND J. DEMMEL, *s-step Krylov subspace methods as bottom solvers for geometric multigrid*, in Proceedings of the 28th International Parallel and Distrtibuted Processing Symposium, IEEE Conference Proceedings, Los Alamitos, 2014, pp. 1149–1158.
[54] C.-Q. YANG AND B. MILLER, *Critical path analysis for the execution of parallel and distributed programs*, in Proceedings of the 8th International Conference on Distributed Computing Systems, IEEE Conference Proceedings, Los Alamitos, 1988, pp. 366–373.