s-step Krylov Subspace Methods as Bottom Solvers for Geometric Multigrid

Samuel Williams, Mike Lijewski,
Ann Almgren, Brian Van Straalen
Lawrence Berkeley National Lab
SWWilliams@lbl.gov

Erin Carson, Nicholas Knight,
James Demmel
University of California at Berkeley
ecc2z@eecs.berkeley.edu

***Abstract*—Geometric multigrid solvers within adaptive mesh refinement (AMR) applications often reach a point where further coarsening of the grid becomes impractical as individual subdomain sizes approach unity. At this point the most common solution is to use a bottom solver, such as BiCGStab, to reduce the residual by a fixed factor at the coarsest level. Each iteration of BiCGStab requires multiple global reductions (MPI collectives). As the number of BiCGStab iterations required for convergence grows with problem size, and the time for each collective operation increases with machine scale, bottom solves in large-scale applications can constitute a significant fraction of the overall multigrid solve time. In this paper, we implement, evaluate, and optimize a communication-avoiding s-step formulation of BiCGStab (CABiCGStab for short) as a high-performance, distributed-memory bottom solver for geometric multigrid solvers. This is the first time s-step Krylov subspace methods have been leveraged to improve multigrid bottom solver performance. We use a synthetic benchmark for detailed analysis and integrate the best implementation into BoxLib in order to evaluate the benefit of a s-step Krylov subspace method on the multigrid solves found in the applications LMC and Nyx on up to 32,768 cores on the Cray XE6 at NERSC. Overall, we see bottom solver improvements of up to 4.2× on synthetic problems and up to 2.7× in real applications. This results in as much as a 1.5× improvement in solver performance in real applications.**

***Keywords*—Multigrid; Communication-avoiding; BiCGStab;**

I. Geometric Multigrid

Many large-scale numerical simulations in a wide variety of scientific disciplines require the solution of elliptic and/or parabolic partial differential equations on a single-level domain-decomposed grid or on a hierarchy of adaptively refined meshes (AMR levels). Simulations that solve time-dependent systems of equations may run for many time steps and may require multiple solves per timestep. Often the cost of solving the elliptic or parabolic equations is a substantial fraction of the total cost of the simulation.

A variety of techniques are available to solve these equations. The most widely used are direct solvers, Krylov subspace methods, fast multipole methods, and multigrid techniques. For any particular simulation of a given size, one technique or combination of techniques may be better. Here, we focus exclusively on geometric multigrid with a Krylov subspace method coarse grid (bottom of the U-cycle) solver due to its widespread applicability and use. This combination is available as an option (or the default) in a number of publicly available software packages, such as BoxLib [1], Chombo [2], PETSc [3], and hypre [4].

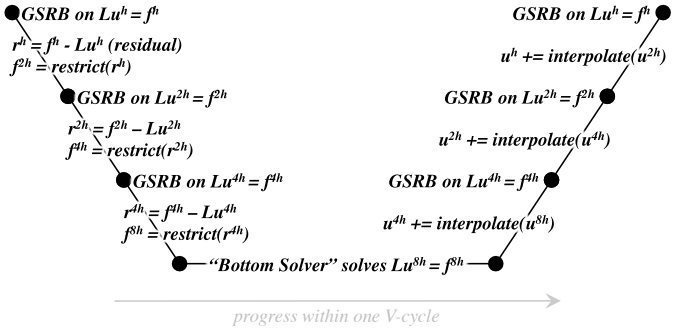

Fig. 1. A four-level multigrid V-cycle for solving $Lu^h = f^h$. Superscripts represent grid spacing. Restriction is terminated at $8h$. The performance and scalability of the bottom solver is the focus of this work.

As shown in Figure 1, a typical geometric multigrid V-cycle coarsens the computational domain until a stopping criterion is reached. At each level down the V-cycle, the error on the solution is smoothed using, e.g., the Gauss-Seidel method with red/black ordering (GSRB), and a new right-hand side for the coarser level is constructed from the residual. For a simulation on a single uniform domain, a typical restriction termination criterion is met when the domain reaches 2-4 cells on a side.

When the domain is decomposed over a large number of processes, the coarsening would typically stop when each subdomain (box) reaches this small size. If there are a large number of subdomains then the problem size of the resultant bottom solve may be small relative to the finest level, but still large enough that its computational cost is substantial. One technique to alleviate this performance bottleneck is to agglomerate the collection of small boxes into one or more larger boxes, further coarsen those boxes, and solve on a subset of the processors. While this technique is very effective for uniform rectangular meshes, the geometry of non-rectangular domains or individual AMR levels with irregular coarse/fine boundaries may preclude the use of this technique. Thus we are left with the problem of computationally costly bottom solves, the focus of this paper.

Once the bottom problem has been solved, the coarse solutions are interpolated to update the solutions on the finer

levels whose errors are then smoothed again. The whole V-cycle process repeats until the norm of the residual on the finest grids reaches some specified tolerance.

II. Related Work

The s-step BiCGStab method used in this work is designed to minimize the communication bottlenecks found in today's supercomputers [5]. Research in s-step Krylov subspace methods is not new; over the past three decades, many recognized that global synchronizations (in both sequential and parallel implementations) could be blocked/fused to reduce the communication bottleneck. This has led to many s-step Krylov subspace methods (see, e.g., [6], [7], [8], [9], [10], [11], [12], [13], [14]). For a thorough overview, please review [15].

In addition to s-step formulations, there are other approaches to reducing synchronization cost in Krylov subspace methods, many of which involve overlapping communication and computation. In [16], Gropp presents an asynchronous variant of the conjugate gradient method (CG) with two global synchronization points per iteration that can be overlapped with the matrix-vector multiplication and application of the preconditioner, respectively. Overlapping techniques for solving least squares problems with Krylov subspace methods on massively parallel distributed memory machines are presented in [14].

In [17], a pipelined version of GMRES is presented, where the authors overlap nonblocking reductions (to compute dot products needed in later iterations) with matrix-vector multiplications. This resulted in speedups and improved scalability on distributed-memory machines. An analogous pipelined version of CG is presented in [18], and the pipelining approach is discussed further in [19]. Another pipelined algorithm, currently implemented in the SLEPc library [20], is the Arnoldi method with delayed reorthogonalization (ADR) [21]. This approach mixes work from the current, previous, and subsequent iterations to avoid extra synchronization points due to reorthogonalization.

In [22], the authors present a communication-avoiding Chebyshev iteration that uses a tiling approach to improve temporal locality when performing repeated matrix-vector multiplications (see also [23]). They apply this method as the smoother in a geometric multigrid solver for the Poisson equation on a regular 2D grid.

Unlike the overlapping or pipelining techniques which hide communication, the s-step formulation used here enables asymptotic reductions in global communication.

III. Experimental Setup

A. miniGMG

In order to provide a testbed for algorithmic exploration as well as a platform for detailed performance analysis, we leverage our miniGMG geometric multigrid benchmark detailed in Williams et al. [24]. The benchmark provides a controlled environment in which we may generate a variety of problems with different performance and numerical properties, analyze convergence, and provide detailed timing breakdowns by operation and by level within the V-cycle. miniGMG creates a global 3D domain and partitions it into subdomains of sizes similar to those found in real-world AMR multigrid applications. The subdomains are then distributed among processes. As in previous work, we choose to solve a finite-volume discretization of the variable-coefficient Helmholtz equation ($Lu = a\alpha u - b\nabla \cdot \beta \nabla u = f$) on a cube with periodic boundary conditions. We create only one 64^3 box per process to mimic the memory capacity challenges of real AMR MG combustion applications — multilevel AMR coupled with dozens of chemical species significantly limits the memory available to any one solve. For our synthetic problem, the right-hand side is a 3D triangle wave whose period spans the entire domain, and we set the nominally spatially-varying coefficients $\alpha = \beta = 1.0$, and $a = b = 0.9$. miniGMG uses piecewise constant interpolation between levels for all points and solves this problem using standard V-cycles terminated when each box is coarsened to 4^3. However, where previous work used multiple GSRB relaxations at this level [24], we now use a matrix-free BiCGStab bottom solver. We used the highly portable OpenMP version of miniGMG. Neither SIMD optimizations nor communication-avoiding smoothers are used in this paper as the focus is on the bottom solver.

B. Cray XE6 (Hopper)

All experiments presented in this paper are performed on Hopper, a Cray XE6 MPP at NERSC. Each compute node has four 6-core Opteron chips each with two DDR3-1333 memory controllers [25]. Each superscalar out-of-order core includes both a 64KB L1 and a 512KB L2 cache, while each chip includes a 6MB L3 cache. Pairs of compute nodes are directly connected via HyperTransport to a high-speed Gemini network chip. The Gemini network chips are connected to form a high-bandwidth, low-latency 3D torus. There is some asymmetry within the torus as peak MPI bandwidths are either 3GB/s or 6GB/s depending on direction. Programmers have virtually no control over job placement, and thus little control over this. Latencies can be as little as 1.5μs, but are higher in practice.

IV. Classical BiCGStab Performance

A. BiCGStab

Algorithm 1 presents the classical BiCGStab method as given by Saad [26], supplemented with convergence checks. We denote the dot product $a^T b$ as (a, b). BiCGStab's vector and matrix operations are grid and stencil operations in miniGMG. Thus, a dot product is implemented with pairwise multiplications between the corresponding cells of two grids followed by a global reduction, while a matrix-vector multiplication is a ghost zone exchange (point-to-point MPI communication) followed by the application of a stencil to a grid (e.g., Ap_j is a stencil applied to grid p_j).

In Algorithm 1, we observe that in each iteration, one nominally performs two matrix-vector multiplications, four dot products, and two norms. In miniGMG, BoxLib, and Chombo, we use the max norm ($||r||_\infty$) instead of the L^2 norm ($||r||_2$) to check convergence (the communication cost is the same).

While the matrix-vector multiplications simply require point-to-point (P2P) communication (`MPI_Isend/Irecv`) with neighboring processes, the dot products and norms require collective operations (`MPI_AllReduce`) — global reductions across the entire machine. In general, depending on the size of the problem, communication pattern, and parallel concurrency, either the local computation, P2P communication, or collective operations could be the performance bottleneck. As we are focused on a bottom solver in which each process owns only a 4^3 box after coarsening, we expect the bottom solve time to be dominated by either P2P communication or collectives.

Algorithm 1 Classical BiCGStab for solving $Ax = b$

1: Start with initial guess x_0
2: $p_0 := r_0 := b - Ax_0$
3: Set $\tilde{r}$ arbitrarily so that $(\tilde{r}, r_0) \neq 0$
4: **for** $j := 0, 1, \ldots$ until convergence or breakdown **do**
5: $\quad \alpha_j := (\tilde{r}, r_j)/(\tilde{r}, Ap_j)$
6: $\quad x_{j+1} := x_j + \alpha_j p_j$
7: $\quad q_j := r_j - \alpha_j Ap_j$
8: $\quad$ Check $||q_j||_2 = (q_j, q_j)^{1/2}$ for convergence
9: $\quad \omega_j := (q_j, Aq_j)/(Aq_j, Aq_j)$
10: $\quad x_{j+1} := x_{j+1} + \omega_j q_j$
11: $\quad r_{j+1} := q_j - \omega_j Aq_j$
12: $\quad$ Check $||r_{j+1}||_2 = (r_{j+1}, r_{j+1})^{1/2}$ for convergence
13: $\quad \beta_j := (\alpha_j/\omega_j)(\tilde{r}, r_{j+1})/(\tilde{r}, r_j)$
14: $\quad p_{j+1} := r_{j+1} + \beta_j(p_j - \omega_j Ap_j)$
15: **end for**

B. Performance and Scalability of miniGMG with BiCGStab

Figure 2 shows the time-to-solution for a multigrid solve as one weak-scales from 8 to 4096 processes on Hopper, where each process has 6 threads (one Opteron chip) and receives one subdomain (box) of 64^3 points at the finest grid. Thus, the problem scales from a domain with $N = 128^3$ points distributed over 48 cores to $N = 1024^3$ using 24,576 cores. The convergence criterion for miniGMG is to reduce the norm of the residual on the fine grid by a factor of at least 10^{-10}. Ideally, geometric multigrid performs $O(N)$ computations, so one might hope that runtime stays constant while weak-scaling — reality (solid red line) is far from this.

miniGMG allows us to tabulate time by level. Doing so allows us to separate the time spent in the traditional multigrid V-cycles (dashed green line) from the time spent in the BiCGStab bottom solver (dashed red line). Whereas the multigrid part of the solver scales perfectly, the time spent in the bottom solver grows rapidly. These observations reflect the general characteristics of weak-scaled applications dependent on multigrid solvers and motivate the need to address the performance and scalability of Krylov-based bottom solvers for geometric multigrid.

C. Breakdown of Bottom Solve Time

Our miniGMG benchmark also allows us to quantify the breakdown of time in the bottom solver by operation. Figure 3 (top) clearly shows the vast majority of time in the

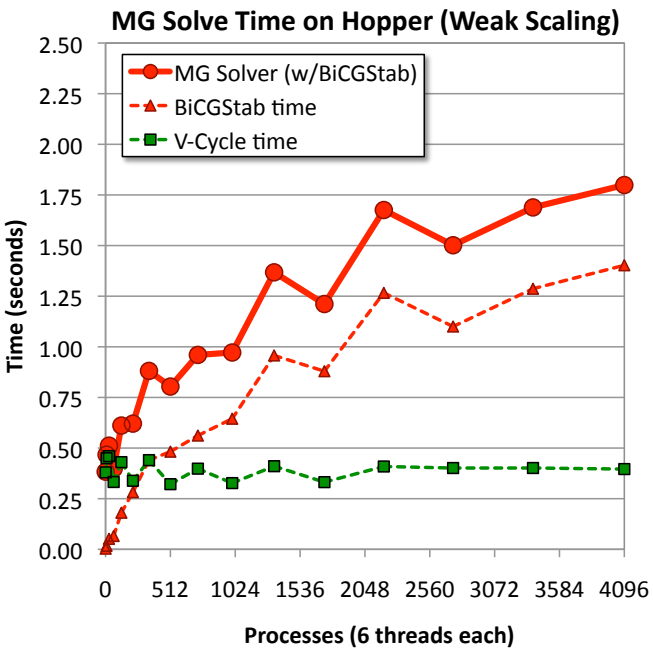

Fig. 2. Breakdown of miniGMG solver time as one weak-scales a problem with 64^3 points per process up to 4096 processes (24,576 cores) on Hopper for our synthetic problem.

bottom solve is spent in `MPI_AllReduce`. The time spent in P2P communication is an order of magnitude less and the time spent in computation is insignificant. Figure 3 (bottom) shows that the rapid increase in `MPI_AllReduce` time is attributable to two effects. First, the total number of BiCGStab iterations (summed across all V-cycles) increases quickly with problem size. This should come as no surprise when weak-scaling an algorithm with potential superlinear computational complexity. Second, the average time in `MPI_AllReduce` per iteration increases with machine scale. This is certainly plausible given that Hopper's network topology is a 3D torus and the PBS job scheduler has been optimized to maximize machine usage without guaranteeing each job is apportioned a compact subtorus. The result is an increasing number of increasingly slower BiCGStab iterations.

As it is difficult to reduce the time required for a collective operation like `MPI_AllReduce` without changing either the MPI implementation, the network architecture, or the job scheduler, we consider optimizing the bottom solver to reduce the number of times the collective is performed.

V. CABiCGStab

Krylov subspace methods are based on projection onto expanding subspaces, where, in each iteration m, the approximate solution is chosen from the expanding *Krylov subspace*

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

The main idea behind s-step Krylov subspace methods is to block the iteration space into groups of s, splitting the main loop of the Krylov subspace method into an outer loop

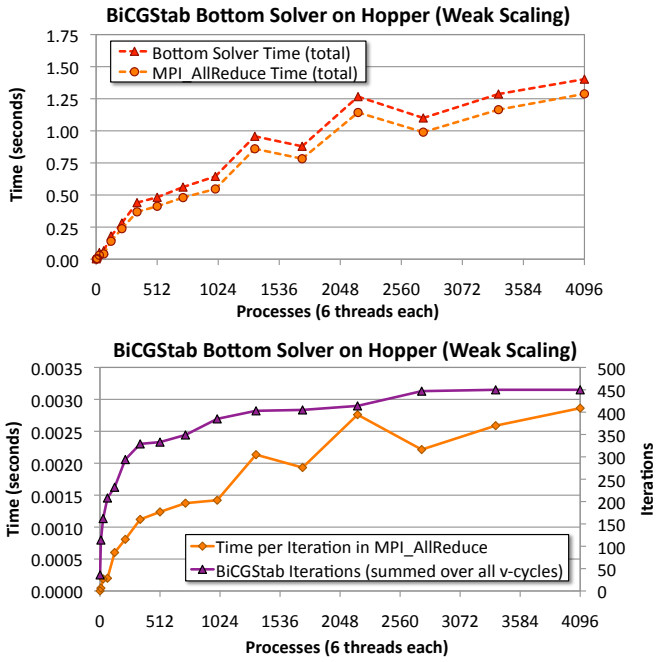

Fig. 3. Breakdown of the performance effects of the classical BiCGStab bottom solves as a function of scale. Top: dominant times. Bottom: requisite iterations vs. `MPI_AllReduce` time per iteration.

(the communication step) and an inner loop (computation steps). Each outer loop involves computing a basis for a $\Theta(s)$-dimensional Krylov subspace and computing a Gram matrix to encode dot products with the resulting basis vectors. This formulation, equivalent to a change of basis, then allows BiCGStab iterate updates to be computed without communication in the inner loop. For a thorough overview of s-step Krylov subspace methods see [15].

A. Algorithm

We briefly review the derivation of our communication-avoiding s-step BiCGStab (CABiCGStab for short) for the familiar reader; for details, see [5]. Assume we are at some iteration m and we wish to determine the dependencies for computing the next s iterations up to $m+s$. By induction on lines 6, 7, 10, 11, and 14 of Algorithm 1, we can write, for $0 \le j \le s-1$,

$$\begin{aligned} &p_{m+j+1}, r_{m+j+1}, x_{m+j+1} - x_m \in \mathcal{K}_{2s+1}(A, p_m) + \mathcal{K}_{2s}(A, r_m), \\ &q_{m+j} \in \mathcal{K}_{2s}(A, p_m) + \mathcal{K}_{2s-1}(A, r_m), \quad \text{and} \\ &p_{m+j} \in \mathcal{K}_{2s-1}(A, p_m) + \mathcal{K}_{2s-2}(A, r_m). \end{aligned}$$

Let P and R denote bases for the subspaces $\mathcal{K}_{2s+1}(A, p_m)$ and $\mathcal{K}_{2s}(A, r_m)$, respectively. We can then write, for $0 \le j \le s-1$,

$$\begin{aligned} q_{m+j} &= [P, R] d_j, \\ x_{m+j+1} - x_m &= [P, R] e_{j+1}, \\ r_{m+j+1} &= [P, R] c_{j+1}, \quad \text{and} \\ p_{m+j+1} &= [P, R] a_{j+1}, \end{aligned}$$

i.e., the length-$(4s+1)$ vectors $d_j, a_{j+1}, c_{j+1}, e_{j+1}$ are coordinates for the length-N vectors $q_{m+j}, p_{m+j+1}, r_{m+j+1}, x_{m+j+1} - x_m$, respectively, in terms of the columns of $[P, R]$. In our implementation, we order columns in P and R in the order they are computed; this gives the initial values $a_0 = [1, 0_{1,4s}]^T$, $c_0 = [0_{1,2s+1}, 1, 0_{1,2s-1}]^T$, and $e_0 = 0_{4s+1,1}$, where $0_{i,\ell}$ is a zero matrix with i rows and ℓ columns.

The bases P, R are generated by polynomials satisfying a three-term recurrence and represented by a $(4s+1)$-by-$(4s+1)$ tridiagonal matrix T', satisfying

$$A[\underline{P}, 0_{n,1}, \underline{R}, 0_{n,1}] = [P, R] T',$$

where $\underline{P}$ and $\underline{R}$ are P and R, respectively, with the last columns omitted (see [5] for details). Then the two matrix-vector multiplications needed in the iterate updates for $0 \le j \le s-1$ can be written as

$$\begin{aligned} A[q_{m+j}, p_{m+j}] &= A[P, R][d_j, a_j] \\ &= A[\underline{P}, 0_{n,1}, \underline{R}, 0_{n,1}][d_j, a_j] \\ &= [P, R] T' [d_j, a_j]. \end{aligned}$$

Since T' is small, each process stores a copy locally, and thus multiplication by T' incurs no communication.

Rather than update the length-N iterates in the inner loop, we update their respective length-$(4s+1)$ coordinate vectors, replacing multiplications with A by multiplications with T'. That is, lines 6, 7, 10, 11, and 14 in Algorithm 1 become

$$\begin{aligned} e_{j+1} &= e_j + \alpha_{m+j} a_j, \\ d_j &= c_j - \alpha_{m+j} T' a_j, \\ e_{j+1} &= e_{j+1} + \omega_{m+j} d_j, \\ c_{j+1} &= d_j - \omega_{m+j} T' d_j, \quad \text{and} \\ a_{j+1} &= c_{j+1} + \beta_{m+j} (a_j - \omega_{m+j} T' a_j). \end{aligned}$$

The length-N iterates can be recovered by premultiplication of the coordinate vectors by $[P, R]$.

The remaining step is to eliminate the length-N dot products in lines 5, 9, and 13 of Algorithm 1 that each incur one `MPI_AllReduce` per iteration. Let $[G, g] = [P, R]^T [P, R, \tilde{r}]$. Here, G is a $(4s+1)$-by-$(4s+1)$ matrix and g is a length-$(4s+1)$ vector. This matrix-matrix multiplication can be realized with a single `MPI_AllReduce`. Since G and g are both small quantities that can be duplicated locally, the dot products can be computed locally by the relations

$$\begin{aligned} (\tilde{r}, r_{m+j}) &= (g, c_j), \\ (\tilde{r}, r_{m+j+1}) &= (g, c_{j+1}), \\ (\tilde{r}, A p_{m+j}) &= (g, T' a_j), \\ (q_{m+j}, A q_{m+j}) &= (d_j, G T' d_j), \quad \text{and} \\ (A q_{m+j}, A q_{m+j}) &= (T' d_j, G T' d_j), \end{aligned}$$

for iterations $0 \le j \le s-1$. Note that since

$$\|q_{m+j}\|_2 = (q_{m+j}, q_{m+j})^{1/2} = (d_j, G d_j)^{1/2},$$

and similarly,

$$\|r_{m+j+1}\|_2 = (r_{m+j+1}, r_{m+j+1})^{1/2} = (c_{j+1}, Gc_{j+1})^{1/2},$$

the convergence checks in lines 8 and 12 can be performed locally (no communication) for $0 \leq j \leq s-1$.

Using these transformations, we can block BiCGStab iterates into groups of s, resulting in an outer loop that operates on blocks of iterates and an inner loop that computes s iterations of iterate updates. The resulting CABiCGStab method is shown in Algorithm 2. Observe that the same traditional breakdown criteria appear and are resolved in Algorithm 2 as in Algorithm 1.

B. Polynomial Bases

We have flexibility in selecting polynomials to use in construction of P and R (bases for $\mathcal{K}_{2s+1}(A, p_m)$ and $\mathcal{K}_{2s}(A, r_m)$, resp.). The condition number and norm of P and R have important implications for stability and convergence in finite precision; in the extreme case, an ill-conditioned basis can lead to divergence of the residual. The simplest basis for the Krylov subspace $\mathcal{K}_i(A, v)$ is the monomial basis, i.e., $[v, Av, \ldots, A^{i-1}v]$. It is well-known, however, that the monomial basis condition number grows exponentially with basis size ($\propto s$ in our case), which makes its use appropriate only for small values of s. We can use any basis of the form $[\rho_0(A)v, \rho_1(A)v, \ldots, \rho_{i-1}(A)v]$, where ρ_j is a polynomial of degree j. Typical choices resulting in well-conditioned matrices include Newton and Chebyshev polynomials, which are based on the spectrum of A (see, e.g., [27]).

C. Stability Improvements

We perform minor adjustments to the convergence checks in Algorithm 2 to handle finite precision roundoff error. In finite precision, the use of G in computing lines 14 and 18 can result in small negative numbers for estimates of $\|q_{m+j}\|_2$ and $\|r_{m+j+1}\|_2$. In this case, the result is flushed to 0, indicating convergence. As these convergence checks are entirely local (no communication) and operate on the tiny coordinate vectors, they are very fast and do not negate our communication-avoiding benefits attained elsewhere.

D. Implementation in miniGMG and Performance Potential

CABiCGStab (Algorithm 2) provides potential performance benefits in three areas — reducing the number of collective communications, reducing the number of P2P communications, and eliminating vertical (DRAM) data movement. One can tailor the implementation to optimize for whichever of these is the bottleneck for the problem and scale in question.

Inter-process communication only occurs in lines 6, 7, and 8 (there is no inter-process communication in the inner (j) loop). Moreover, the most computationally expensive routines occur in these three lines as well. All computations in the inner loop are operations on tiny length-$(4s+1)$ vectors. We will thus focus our analysis on the performance benefits of these lines.

Currently, miniGMG uses the monomial basis, i.e.,

$$[P, R] = [p_m, Ap_m, \ldots, A^{2s}p_m, r_m, Ar_m, \ldots, A^{2s-1}r_m].$$

Algorithm 2 CABiCGStab for solving $Ax = b$

1: Start with initial guess x_0
2: $p_0 := r_0 := b - Ax_0$
3: Set $\tilde{r}$ arbitrarily so that $(\tilde{r}, r_0) \neq 0$
4: Construct $(4s+1)$-by-$(4s+1)$ matrix T'
5: **for** $m := 0, s, 2s, \ldots$ until convergence or breakdown **do**
6: Compute P, a basis for $\mathcal{K}_{2s+1}(A, p_m)$
7: Compute R, a basis for $\mathcal{K}_{2s}(A, r_m)$
8: $[G, g] := [P, R]^T[P, R, \tilde{r}]$
9: Initialize length-$(4s+1)$ vectors a_0, c_0, d_0, e_0
10: **for** $j := 0$ to $s-1$ (or convergence/breakdown) **do**
11: $\alpha_{m+j} := (g, c_j)/(g, T'a_j)$
12: $e_{j+1} := e_j + \alpha_{m+j}a_j$
13: $d_j := c_j - \alpha_{m+j}T'a_j$
14: Check $||q_{m+j}||_2 = (d_j, Gd_j)^{1/2}$ for convergence
15: $\omega_{m+j} := (d_j, GT'd_j)/(T'd_j, GT'd_j)$
16: $e_{j+1} := e_{j+1} + \omega_{m+j}d_j$
17: $c_{j+1} := d_j - \omega_{m+j}T'd_j$
18: Check $||r_{m+j+1}||_2 = (c_{j+1}, Gc_{j+1})^{1/2}$ for convergence
19: $\beta_{m+j} := (\alpha_{m+j}/\omega_{m+j})(g, c_{j+1})/(g, c_j)$
20: $a_{j+1} := c_{j+1} - \beta_{m+j}(a_j - \omega_{m+j}T'a_j)$
21: **end for**
22: $p_{m+s} := [P, R]a_s$
23: $r_{m+s} := [P, R]c_s$
24: $x_{m+s} := [P, R]e_s + x_m$
25: **end for**

One can either perform these matrix-vector multiplications sequentially ($A^{k+1}p_m = A(A^k p_m)$), in pairs ($[A^{k+1}p_m, A^{k+1}r_m] = A[A^k p_m, A^k r_m]$), or in a communication-avoiding matrix powers [23] implementation that calculates several powers of A by aggregating MPI communication, reading A only once from DRAM, and performing some redundant work. The first approach will require $4s-1$ P2P communications every s steps — roughly twice the number of P2P communications for s iterations of classical BiCGStab (two per iteration). This is the approach currently used in miniGMG. The second approach can return this to parity with the classical algorithm with the caveat that the size of each message is doubled. The benefit of the third approach heavily depends on the value of s, the increased number of messages for the depth-$\Theta(s)$ ghost zones, the size of the matrix, and the characteristics of the machine. To date, the third approach has not shown benefits for miniGMG bottom solves with 7-point stencils and small s, but it is still an area of active research.

Line 8 of Algorithm 2 shows the construction of the Gram-like matrix $[G, g]$. As $[P, R, \tilde{r}]$ contains multiple grids as columns, the operation $[G, g] := [P, R]^T[P, R, \tilde{r}]$ is effectively a series of dot products on grids distributed across thousands of processes. If performed sequentially, these operations would require at least $4s^2$ calls to `MPI_AllReduce` — clearly a poor choice. It is thus much better to have each process aggregate the partial sums into a matrix and perform one

MPI_AllReduce on a matrix of size $(4s+1)$-by-$(4s+2)$. Moreover, one can exploit the symmetry of G for additional savings, although our implementation currently does not. Nevertheless, we have implemented the construction of $[G, g]$ as a high-performance distributed matrix-matrix multiplication, using multithreading to compute the local dot products and operating directly on the structured grid data structures. Moreover, where the classical algorithm required six calls to MPI_AllReduce per iteration, CABiCGStab asymptotically approaches $1/s$ calls to MPI_AllReduce per iteration.

The current infrastructure of miniGMG does not permit either exchanging ghost zones or applying the stencil A to pairs $[p_m, r_m]$. As such, we expect miniGMG's implementation of CABiCGStab to double the time spent in both the P2P communication and the application of the stencil (matrix-vector multiplication).

As some multigrid solves are "easy", requiring few (perhaps less than s) iterations in each bottom solve, we have implemented a "telescoping" approach to CABiCGStab in which the value of s increases over the course of the solve. The first iteration of the m loop uses $s = 1$, the second uses $s = 2$, and the n-th iteration uses $s = \min\{s_{\max}, 2^n\}$. This ensures that easy solves do not see high initial startup costs of computing the full $[P, R]$ and $[G, g]$ while "hard" solves see the asymptotic benefits.

Finally, we have also implemented a communication-avoiding version of conjugate gradient (CACG) based on [15]. Although its implementation and performance benefits are quite similar to CABiCGStab, its performance will not be presented in this paper because the solves within AMR MG applications are usually nonsymmetric.

E. Benefits to miniGMG

Figure 4 presents the resultant performance of miniGMG solver (solid blue line) using our CABiCGStab implementation with $s = 4$ on Hopper as we scale to 4096 processes. As each process is 6-way multithreaded, the final data point uses 24,576 cores and solves a 1024^3 problem. For reference, we also include the performance using the classical BiCGStab implementation (solid red line). Additionally, we break out the times spent in the respective bottom solves. At 24K cores, CABiCGStab's ability to asymptotically reduce the number of calls to MPI_AllReduce improves bottom solver performance by more than 4.2× and the overall multigrid solve time by nearly 2.5×. Moreover, this dramatically improves parallel efficiency and helps ensure any superlinear computational complexity of the bottom solver does not severely impede the linear computational complexity of geometric multigrid. We observe the aggregate performance (degrees of freedom solved per second) improves to nearly linear with CABiCGStab.

Figure 5 presents the reductions in time across all bottom solves using either classical BiCGStab or CABiCGStab (with $s = 4$) as the bottom solver. As expected, the sequential computation of $[P, R]$ resulted in a doubling of the time spent in P2P communication. It also doubled the time required to apply the linear operator — however, this time is negligible.

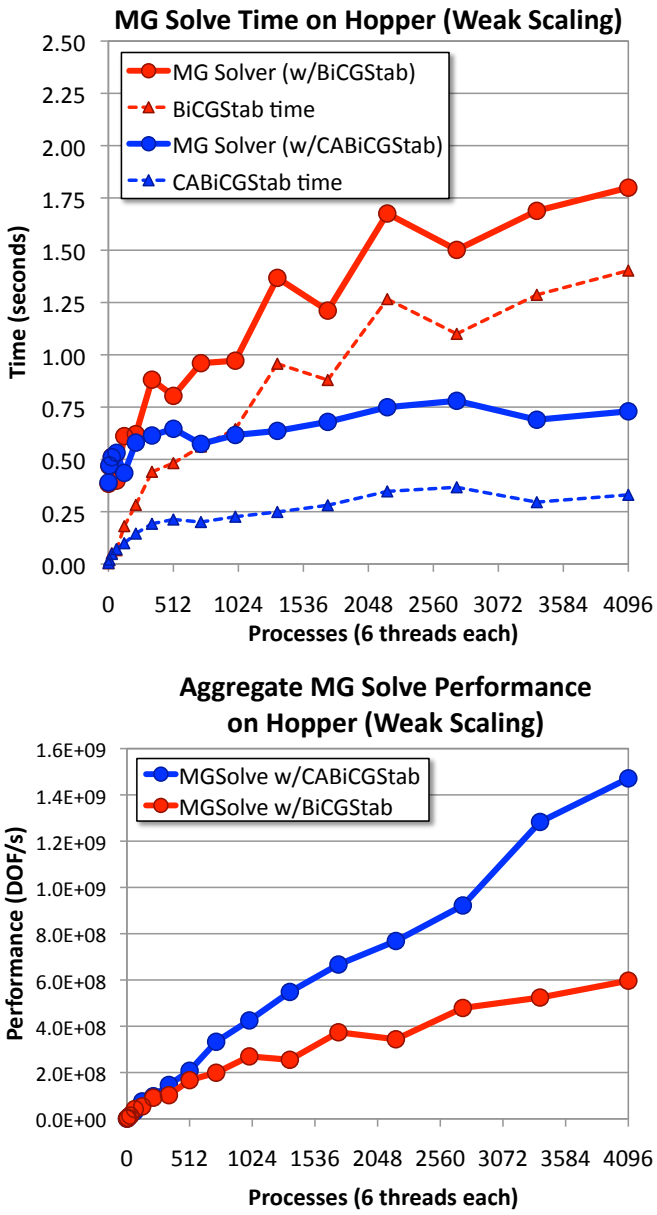

Fig. 4. miniGMG solve time (top) and performance (bottom) in degrees of freedom solved per second (DOF/s) using either the classical BiCGStab or our new CABiCGStab (with $s = 4$) bottom solver.

Although CABiCGStab now requires more than 150 local dot products when calculating each process' partial sum of the BLAS3-like operation $[G, g] = [P, R]^T[P, R, \tilde{r}]$, when properly optimized, this time is tiny compared to the MPI communication times. Nominally, in s steps, one expects BiCGStab to call MPI_AllReduce $6s$ times, but CABiCGStab to call it only once. Thus, if collective performance were always latency-limited (regardless of message size), then we would have expected CABiCGStab to reduce the MPI_AllReduce time by a factor of 24 for $s = 4$. Unfortunately, the size of the reductions has increased from an 8-byte double-precision value to 2448 bytes — the $(4s+1)$-by-$(4s+2)$ double-precision

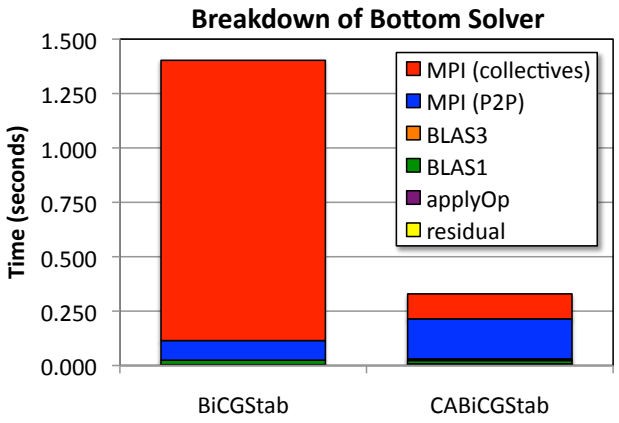

Fig. 5. Breakdown of the net time spent across all bottom solves at 24,576 cores with either the classical BiCGStab or CABiCGStab ($s = 4$) bottom solver.

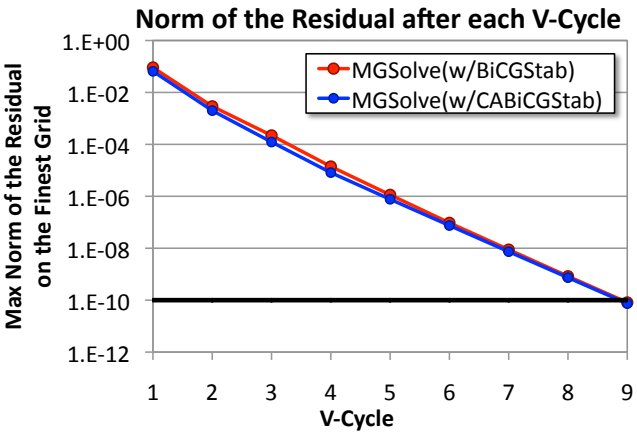

Fig. 6. Max norm of the residual on the finest grid after each V-cycle with either the classical BiCGStab or CABiCGStab ($s = 4$) bottom solver.

$[G, g]$ matrix. This has the effect of limiting the observed reduction in collective time to only 11.2×.

We selected $s = 4$ as it provides a good speedup in the bottom solver without being so aggressive that it demands an alternative basis like the aforementioned Newton or Chebyshev bases lest converge be impeded. Figure 6 shows the max norm of the residual on the finest grid after each V-cycle using either the classical BiCGStab or our new CABiCGStab bottom solver. We observe that, as expected, using the communication-avoiding solver with a moderate choice of s does not significantly perturb the convergence of the multigrid solver. The difference in convergence is likely an artifact of using the L^2 norm checks rather than the max norm that miniGMG nominally uses.

F. Additional Memory Requirements

Our CABiCGStab implementation requires extra storage to hold $[P, R, \tilde{r}]$. Compared to our BiCGStab implementation, CABiCGStab requires $4s - 2$ additional grids. Although this overhead may be prohibitive for weak-scaled Krylov solves on fine grids, in the context of multigrid, it is negligible — roughly 24KB per process for $s = 4$.

VI. Application Results

Thus far, we have used a synthetic problem within the miniGMG benchmark framework in order to provide some initial insights into the benefits and challenges of bottom solvers and communication avoidance within geometric multigrid. In order to quantify how well these technologies translate to real applications, we implemented CABiCGStab within BoxLib [1], a framework for AMR on block-structured grids, and evaluated performance on both 2D and 3D applications. This has allowed us to quantify the performance challenges of multigrid solves found in real, large-scale science applications whose AMR strategies are driven by the needs of the entire simulation, rather than optimized for multigrid performance.

The synthetic problem is perfectly repeatable as neither the right-hand side nor the coefficients change. Time-dependent applications typically perform many multigrid solves with time-varying right-hand sides and possibly time-varying coefficients, thus requiring an ensemble of runs for benchmarking purposes. In addition, whereas miniGMG tests only periodic boundary conditions, real applications can have a variety of boundary conditions including Dirichlet, Neumann, and combinations of the two. Multigrid solves on a fine level within an AMR simulation, which rely on Dirichlet boundary conditions from a coarser level, can be particularly challenging because of the irregular geometry of the coarse/fine interface.

Although the application results presented here are based on BoxLib and BoxLib-based applications, the conclusions should generalize to other structured grid frameworks such as Chombo, PETSc, and hypre.

A. BoxLib CABiCGStab Implementation

The BoxLib AMR framework provides two versions of its BiCGStab iterative solver — one written in C++ and one in Fortran. We implemented both a C++ and a Fortran version of CABiCGStab (using miniGMG's version as a reference implementation) within BoxLib to provide drop-in replacements for the existing solvers. There are a number of features within BoxLib that are not present in miniGMG that can be used to realize certain optimizations in CABiCGStab.

As noted, when constructing $[P, R]$, one may construct the powers sequentially, in parallel, or in a communication-avoiding manner. Whereas miniGMG constructs them sequentially, the BoxLib implementation constructs the powers in pairs ($[A^{k+1}p_m, A^{k+1}r_m] = A[A^k p_m, A^k r_m]$). This has the benefit of reducing the number of times one must read A, or more precisely, the variable coefficients. Unfortunately, in itself, this provides no great performance boost to local computation as the solver is usually run as a bottom solver on tiny problems that fit in cache. However, this approach allows one to exploit the communication features in BoxLib that allow one to exchange the ghost zones for multiple variables with a minimal number of messages. Thus, where miniGMG requires a ghost zone exchange for every matrix-vector multiplication

with p_m and another ghost zone exchange for every matrix-vector multiplication with r_m, the BoxLib solvers pair up the ghost zone exchanges and send half as many messages of twice the size — a clear win for bottom solvers where the messages are tiny and thus latency-limited. Ultimately, this allows one to ensure the number of ghost zone exchanges per iteration in CABiCGStab is the same as the number in classical BiCGStab.

Unfortunately, the boundary conditions found in real applications coupled with the resultant coarse/fine boundary conditions in AMR preclude one from simply filling a four-element deep ghost zone and calculating four powers in a communication-avoiding manner as in our previous work [24]. Thus, we are currently restricted to calculating $[P, R]$ in pairs.

Like miniGMG, the BoxLib C++ solver uses a telescoping method, the monomial basis, and $s_{\max} = 4$, while the BoxLib Fortran solver hard-codes $s = 4$. Experiments on the bottom solves in real applications showed that using larger values of s with the monomial basis necessitated additional V-cycles and a loss of performance. These additional V-cycles tended to destroy the performance advantages of a communication-avoiding bottom solver.

On a final note, whereas miniGMG restricts each box down to 4^3 cells, when possible, most BoxLib applications restricts down to 2^3. Although this sounds like a small difference, one must remember that there are thousands of boxes distributed across thousands of cores. Thus, the bottom problem is still relatively large and a factor of 8 reduction in problem size can significantly reduce the number of BiCGStab iterations. When the number of iterations required in each bottom solve becomes comparable to the parameter s, the benefits of CABiCGStab are mitigated.

B. LMC

The Low Mach Number Combustion Code (LMC) simulates gas-phase combustion using a low Mach number model, coupled to detailed chemical reaction networks and differential diffusion [28]. Each timestep of LMC requires two types of cell-centered linear solves using multigrid on block-structured AMR grids.

The first type solves $(a\alpha - b\nabla \cdot \beta\nabla)u = f$, where u represents the concentration of a chemical species, a and b are constants, and α and β may vary spatially. As $a\alpha$ is non-zero, the matrix A is more diagonally dominant than the matrix resulting just from the discretization of $b\nabla \cdot \beta\nabla$. In 3D, the stencil uses a 7-point variable-coefficient stencil with a GSRB smoother. These diffusion solves are relatively easy in that they require only a few V-cycles, each with few iterations of the bottom solver. Diffusion solves will likely see no benefit from a communication-avoiding bottom solver, but could be accelerated with the communication-avoiding smoothers described in our previous work [24].

The second type, the mac_project solve, is similar to the first, but specialized to elliptic equations ($b\nabla \cdot \beta\nabla u = f$). The discretization of this operator is also a 7-point variable-coefficient stencil, and GSRB smoothers are also used for these solves. However the mac_project solves are often particularly challenging, requiring 10 or more V-cycles and potentially hundreds of bottom solver iterations. The mac_project solve is thus an ideal candidate for communication-avoiding bottom solvers and serves as the basis for our LMC experiments.

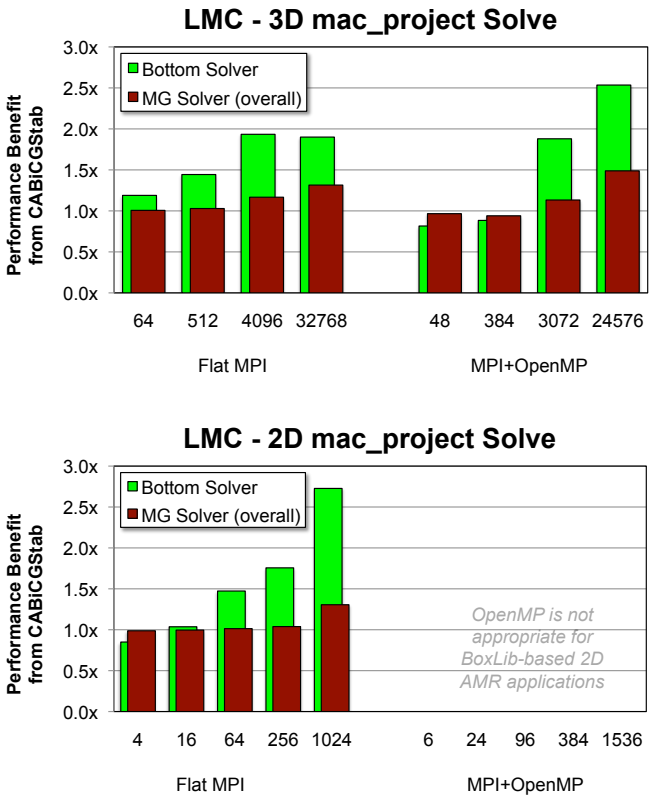

Fig. 7. mac_project solver speedup arising from the use of CABiCGStab in the 3D (top) and 2D (bottom) versions of LMC as a function of the number of cores and programming model.

Figure 7 shows the performance benefit to the mac_project solves in LMC from replacing the classical BiCGStab bottom solver with CABiCGStab when weak-scaling the number of cores on the Cray XE6 (Hopper). We record the total time spent in the mac_project solves across 5 time steps and show speedup for both flat MPI as well as MPI+OpenMP (6 threads per process) for both 2D (square domain with one 64^2 box per process) and 3D (cubic domain with one 64^3 box per process) runs. At the maximum concurrency, we see up to a 2.5× speedup in the bottom solver and up to a 1.5× speedup in the 3D mac_project multigrid solve. The bottom solver speedup is mitigated by the fact that the classical algorithm constitutes less than 43% and 54% of the 3D mac_project solve time for the flat MPI and hybrid MPI+OpenMP versions, respectively. In 2D, the performance benefits of CABiCGStab are more immediate (1.5× speedup at 64 cores), but the time spent in the classical bottom solver was less than 35%.

Unlike miniGMG, where CABiCGStab performs almost exactly the same number of iterations as the classical algorithm, the CABiCGStab bottom solves in each V-cycle of the mac_project solves in LMC perform a similar (within 10%)

number of iterations as the classical algorithm. Nevertheless, the number of V-cycles remains the same.

C. Nyx

Nyx is a 3D N-body and gas dynamics code that uses AMR for large-scale cosmological simulations [29]. Nyx tracks the time evolution of a system of discrete dark matter particles gravitationally coupled to an inviscid ideal fluid in an expanding universe. The mass of the dark matter particles is deposited on the AMR grid hierarchy using a cloud-in-cell scheme and converted to a density field, which is added to the gas density. This total density defines the right-hand side for a constant-coefficient Poisson solve ($b\nabla^2 u = f$) for the gravitational potential. The gradient of the gravitational potential ($-\nabla u$) is the gravitational force vector which is used to accelerate both the dark matter particles and the gas. Both the right-hand side and the gravitational potential are defined on cell centers, and a standard 7-point discretization of the Laplacian operator is used, except at coarse/fine interfaces where the stencil coefficients are modified.

Figure 8 presents the speedup in Nyx's 3D gravity solve from using CABiCGStab, as a function of concurrency (number of cores) and programming model. Like LMC, we conduct weak-scaling experiments with one 64^3 box per process and record the time spent in the solver over the course of 6 time steps. In the flat MPI programming model, we realized a $2\times$ speedup in the bottom solver over BoxLib's BiCGStab baseline. With pure MPI at 4K cores, the bottom solver constitutes only 26% of the total time spent in the multigrid solve. Thus, the overall benefit was limited to a 15% speedup. In a hybrid MPI+OpenMP environment, in which the bottom solver took as much as 41% of the total multigrid solve time, the overall benefit was still 14% even though the speedup of the bottom solver was roughly a factor of $1.6\times$. The convergence behavior when using CABICGStab within Nyx was similar to LMC — a similar number of bottom solver iterations and exactly the same number of V-cycles.

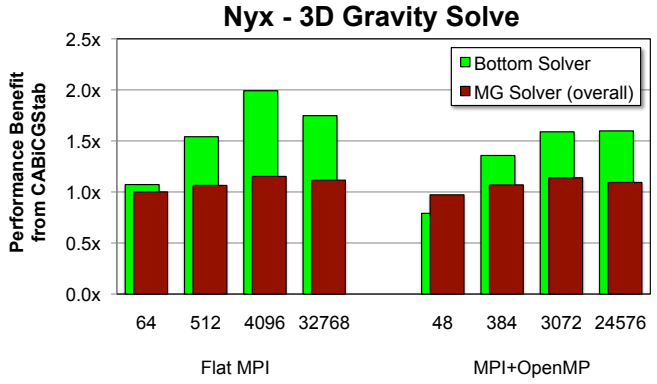

Fig. 8. Gravity solve speedup arising from the use of CABiCGStab in Nyx as a function of the number of cores and programming model.

VII. Summary and Conclusions

At scale, geometric multigrid solvers can be bottlenecked by the performance of their coarse-grid (bottom) solvers. In this paper, we implemented, evaluated, and analyzed the performance benefits of using an s-step formulation of BiCGStab as a replacement for the existing classical BiCGStab bottom solvers used in various multigrid solvers. Using CABiCGStab, we observed as much as a $4.2\times$ increase in performance in the bottom solver for synthetic problems and up to $2.7\times$ in real applications. Overall, replacing the classical algorithm with our new communication-avoiding variant improved solver performance by as much as $1.5\times$ on 24,576 cores. We observe that although the communication-avoiding s-step methods can asymptotically reduce the number of collective operations, their performance benefit is limited (even in the absence of rounding error) by the nontrivial time spent in P2P communication, but also the quadratic increase in the size of the collective communications and vector-vector operations. Moreover, we found that roundoff error due to finite precision effectively limited s to 4 when using the monomial basis on the solves found in real applications. Larger values of s with the monomial basis required more bottom solve iterations and eventually extra v-cycles to reach the same convergence criterion. Future work will explore the performance benefits of alternative polynomials such as Newton or Chebyshev.

Although in this paper we evaluated CABiCGStab as a bottom solver for geometric multigrid on weak-scaled applications, it should provide similar benefits for strong-scaled applications with implicit solvers even if they are not they are preconditioned with multigrid. That is, as one strong-scales a solver, the problem size per process is ultimately reduced to the point where communication (likely collective operations) becomes the bottleneck. This is analogous to the bottom solver challenges in a geometric multigrid V-cycle. The use of a communication-avoiding Krylov variant should allow increased performance and/or increased scalability for strong-scaled applications.

Although not explored in this paper, an implementation of the s-step formulation of BiCGStab could be tailored to provide performance even when bound by local computation. For example, weak-scaled Krylov solves without multigrid may be dominated by local matrix-vector multiplications, which are in turn usually bound by the time required to read the matrix from DRAM. The s-step formulation allows one to optimize the construction of $[P, R, \tilde{r}]$ to (asymptotically) read the matrix once ever s steps.

Broadly speaking, the communication-avoiding s-step Krylov subspace methods expand the co-design space by allowing hardware and software designers to trade collective latency for bandwidth. Moreover, one can trade s fine-grained operations for one large coarse-grained operation that expresses far more parallelism and may be more appropriate given the manycore trends in processor architecture. Along those lines, we plan on expanding this work to GPU and Xeon Phi processors to evaluate whether communication-avoiding

Krylov subspace methods can ensure that those architectures can be efficiently utilized.

ACKNOWLEDGMENTS

Authors from Lawrence Berkeley National Laboratory were supported by the U.S. Department of Energy's Advanced Scientific Computing Research Program under contract DE-AC02-05CH11231. Authors from UC Berkeley were funded by ASCR (award DE-SC0005136), Microsoft (#024263), Intel (#024894), and U.C. Discovery (#DIG07-10227). Further support comes from Par Lab affiliates National Instruments, NEC, Nokia, NVIDIA, Samsung, and Sun Microsystems. This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-05CH11231.

REFERENCES

[1] "BoxLib website," https://ccse.lbl.gov/BoxLib.
[2] "Chombo website," http://seesar.lbl.gov/ANAG/software.html.
[3] "PETSc website," http://www.mcs.anl.gov/petsc/.
[4] "hypre website," http://computation.llnl.gov/casc/hypre/software.html.
[5] E. Carson, N. Knight, and J. Demmel, "Avoiding communication in nonsymmetric Lanczos-based Krylov subspace methods," *SIAM J. Sci. Comp.*, vol. 35, no. 5, 2013, to appear.
[6] Z. Bai, D. Hu, and L. Reichel, "A Newton basis GMRES implementation," *IMA J. Numer. Anal.*, vol. 14, no. 4, pp. 563–581, 1994.
[7] A. Chronopoulos and C. Gear, "*s*-step iterative methods for symmetric linear systems," *J. Comput. Appl. Math*, vol. 25, no. 2, pp. 153–168, 1989.
[8] E. De Sturler and H. Van Der Vorst, "Reducing the effect of global communication in GMRES(m) and CG on parallel distributed memory computers," *Appl. Numer. Math.*, vol. 18, no. 4, pp. 441–459, 1995.
[9] J. Erhel, "A parallel GMRES version for general sparse matrices," *Electron. Trans. Numer. Anal.*, vol. 3, no. 12, pp. 160–176, 1995.
[10] W. Joubert and G. Carey, "Parallelizable restarted iterative methods for nonsymmetric linear systems. Part I: theory," *Int. J. Comput. Math.*, vol. 44, no. 1-4, pp. 243–267, 1992.
[11] C. Leiserson, S. Rao, and S. Toledo, "Efficient out-of-core algorithms for linear relaxation using blocking covers," *J. Comput. Syst. Sci. Int.*, vol. 54, no. 2, pp. 332–344, 1997.
[12] J. Van Rosendale, "Minimizing inner product data dependencies in conjugate gradient iteration," ICASE-NASA, Tech. Rep. 172178, 1983.
[13] H. Walker, "Implementation of the GMRES method using Householder transformations," *SIAM J. Sci. Stat. Comput.*, vol. 9, pp. 152–163, 1988.
[14] T. Yang, "Solving sparse least squares problems on massively distributed memory computers," in *Advances in Parallel and Distributed Computing*. IEEE, 1997, pp. 170–177.
[15] M. Hoemmen, "Communication-avoiding Krylov subspace methods," Ph.D. dissertation, EECS Dept., U.C. Berkeley, 2010.
[16] W. Gropp, "Update on libraries for Blue Waters," http://jointlab.ncsa.illinois.edu/events/workshop3/pdf/presentations/Gropp-Update-on-Libraries.pdf, 2010, bordeaux, France.
[17] P. Ghysels, T. Ashby, K. Meerbergen, and W. Vanroose, "Hiding global communication latency in the GMRES algorithm on massively parallel machines," *SIAM J. Sci. Comput.*, vol. 35, no. 1, pp. C48–C71, 2013.
[18] P. Ghysels and W. Vanroose, "Hiding global synchronization latency in the preconditioned conjugate gradient algorithm," *Parallel Computing*, 2013.
[19] T. Ashby, P. Ghysels, W. Heirman, and W. Vanroose, "The impact of global communication latency at extreme scales on Krylov methods," in *Algorithms and Architectures for Parallel Processing*. Springer, 2012, pp. 428–442.
[20] V. Hernandez, J. Roman, and V. Vidal, "SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems," *ACM Trans. Math. Software*, vol. 31, no. 3, pp. 351–362, 2005.
[21] V. Hernández, J. Román, and A. Tomás, "Parallel Arnoldi eigensolvers with enhanced scalability via global communications rearrangement," *Parallel Comput.*, vol. 33, no. 7, pp. 521–540, 2007.
[22] P. Ghysels, P. Kłosiewicz, and W. Vanroose, "Improving the arithmetic intensity of multigrid with the help of polynomial smoothers," *Numer. Linear Algebra with Appl.*, vol. 19, no. 2, pp. 253–267, 2012.
[23] J. Demmel, M. Hoemmen, M. Mohiyuddin, and K. Yelick, "Avoiding communication in computing Krylov subspaces," EECS Dept., U.C. Berkeley, Tech. Rep. UCB/EECS-2007-123, Oct 2007.
[24] S. Williams, D. Kalamkar, A. Singh, A. Deshpande, B. Van Straalen, M. Smelyanskiy, A. Almgren, P. Dubey, J. Shalf, and L. Oliker, "Optimization of geometric multigrid for emerging multi- and manycore processors," in *Supercomputing*, 2012.
[25] "Hopper website," http://www.nersc.gov/users/computational-systems/hopper.
[26] Y. Saad, *Iterative Methods for Sparse Linear Systems, 2nd edition*. Philadelpha, PA: SIAM, 2003.
[27] B. Philippe and L. Reichel, "On the generation of Krylov subspace bases," *Appl. Numer. Math*, vol. 62, no. 9, pp. 1171–1186, 2012.
[28] M. S. Day and J. B. Bell, "Numerical simulation of laminar reacting flows with complex chemistry," *Combust. Theory Modelling*, vol. 4, no. 4, pp. 535–556, 2000.
[29] A. S. Almgren, J. B. Bell, M. J. Lijewski, Z. Lukic, and E. Van Andel, "Nyx: A massively parallel AMR code for computational cosmology," *Astrophysical Journal*, vol. 765, no. 39, 2013.