Hierarchical and Nested Krylov Methods for Extreme-Scale Computing

Lois Curfman McInnes[a], Barry Smith[a], Hong Zhang[a,b], Richard Tran Mills[c,d]

[a] *Mathematics and Computer Science Division*
Argonne National Laboratory
9700 South Cass Avenue
Argonne, IL 60439, USA
[mcinnes,smith,hzhang]@mcs.anl.gov

corresponding author: Hong Zhang:
phone: 630-252-3978, fax: 630-252-5986

[b] *Department of Computer Science*
Illinois Institute of Technology
10 West 31st street
Chicago, IL 60616, USA

[c] *Computer Science and Mathematics Division*
Oak Ridge National Laboratory
1 Bethel Valley Road
Oak Ridge, TN 37831, USA
rmills@ornl.gov

[d] *Department of Earth and Planetary Sciences*
University of Tennessee
1412 Circle Drive
Knoxville, TN 37936, USA
rtm@utk.edu

Abstract

The solution of large, sparse linear systems is often a dominant phase of computation for simulations based on partial differential equations, which are ubiquitous in scientific and engineering applications. While preconditioned Krylov methods are widely used and offer many advantages for solving sparse linear systems that do not have highly convergent, geometric multigrid solvers or specialized fast solvers, Krylov methods encounter well-known scaling difficulties for over 10,000 processor cores because each iteration requires at least one vector inner product, which in turn requires a global synchronization that scales poorly because of internode latency. To help overcome these difficulties, we have developed *hierarchical* and *nested* Krylov methods in the PETSc library that reduce the number of global inner products required across the entire system (where they are expensive), though freely allow vector inner products across small subsets of the entire system (where they are inexpensive) or use inner iterations that do not invoke vector inner products at all. We introduce the *hierarchical FGMRES method*, or *h-FGMRES*, and we demonstrate the impact of two-level h-FGMRES with a nonlinear preconditioner on the PFLOTRAN subsurface flow application. We also demonstrate the impact of nested BiCGStab with a linear Chebyshev preconditioner. These hierarchical and nested Krylov methods significantly reduced overall PFLOTRAN simulation time on the Cray XK6 when using 10,000 through 224,000 cores through the combined effects of reduced global synchronization due to fewer global inner products and stronger inner hierarchical or nested preconditioners.

Keywords: hierarchical, nested, Krylov methods, iterative preconditioner

1. Introduction

Even on today's high-performance computer architectures, the solution of sparse linear systems is a substantial fraction of overall simulation time and presents challenges in scalability for applications based on partial differential equations (PDEs), which are ubiquitous in scientific and engineering modeling. Iterative solution with preconditioned Krylov methods [1] is the approach of choice for many large-scale applications. Krylov methods often highly accelerate the convergence of simple iterative schemes for

sparse linear systems that do not have highly convergent geometric multigrid solvers or specialized fast solvers. Thus, in general, Krylov methods are often desirable, but they do have a significant drawback on large core counts in that they require at least one global synchronization per iteration [2] for computing a vector inner product of the form $\mathbf{w}^T\mathbf{v} = \sum_{i=1}^{n} w_i v_i$, where $\mathbf{v} = [v_1, v_2, ...v_n],^T$ implemented as local summations followed by an "all-reduce" global collective operation. The other dominant phase of Krylov methods, matrix-vector products, do not require any global synchronization and generally require communication only between neighboring processors (and perhaps hierarchies of neighbors). The global synchronization, done with `MPI_Allreduce()` when using the Message Passing Interface (MPI) [3], is known to be a major bottleneck on the Cray XT5 and Cray XK6 because its cost grows relatively quickly with the number of cores involved in the reduction as a result of internode latency. Although much less of an issue on the IBM Blue Gene/P and Blue Gene/Q, `MPI_Allreduce()` is still strongly affected by load-balancing problems. Moreover, for applications to run efficiently as core counts continue to grow on emerging extreme-scale architectures, algorithmic approaches that reduce global synchronization are increasingly important.

To overcome the bottlenecks of Krylov solvers as applications scale to million-core simulations and beyond, researchers have developed new variants of Krylov methods that reduce and hide both synchronization and communication (see, e.g., [4]). For example, communication-avoiding GMRES (CA-GMRES) [5] reorganizes the sparse-matrix kernel of GMRES [6] to compute k matrix-vector products at once with reduced communication cost; s-step Krylov methods create s new Krylov vectors at once and orthogonalize them together [7–10]; a pipelined variation of GMRES, called $p(l)$-GMRES [11], hides the global reduction latency with l iterations; and an improved variant of BiCGStab [12], named IBiCGStab[13], reduces four global inner products per iteration to one through algorithmic reorganization and overlapping communication with computation. All these approaches have demonstrated important advances for single-level Krylov iterations. In fact, even further reduction of global synchronization can be achieved by incorporating multiple levels of hierarchical or nested inexact iterations as preconditioners. In this paper, we take this approach and name our methods *hierarchical and nested Krylov methods*.

Our work is motivated primarily by two large-scale applications, neutron transport with UNIC [14, 15] and subsurface flow with PFLOTRAN [16–21], both of whose dominant computational kernel is the solution of extremely large, sparse linear systems with over 2 billion degrees of freedom. Inspired by the UNIC strategy of exploiting modeling hierarchy, we have developed a general-purpose approach to hierarchical and nested Krylov solvers and demonstrated its effectiveness for PFLOTRAN. Moreover, because this hierarchical and nested algorithmic framework is encapsulated in the PETSc library [22–24], applications that interface to PETSc linear solvers can easily experiment with these techniques without any source code changes simply by activating the new solvers as runtime options.

The UNIC neutron transport application, which leverages natural hierarchies in modeling, was a 2009 Gordon Bell finalist [14] and demonstrated good weak scaling up to the largest available IBM Blue Gene/P and Cray XT5 at the time. UNIC solves the neutron transport problem in full reactor cores by using the S_n method. Neutron transport is solved in a six-dimensional space, the usual R^3 crossed with two angle variables and energy. Physically the code simulates the density of the stream neutrons at each point within the reactor moving in each direction with a given energy. Of interest to nuclear engineers is the smallest eigenvalue of the scatter operator. The S_n method discretizes the scattering operator in space by using the usual finite-element method, while in angle the S_n method uses the discrete ordinate method (essentially the neutron density is simulated for a finite number of predetermined angles). The discretization in energy is done by using a finite number of energy groups, binning the neutrons into collections with similar energy. The resulting sparse matrix has a natural hierarchical block structure with a large block for each energy, each containing a block for each angle. The block for each angle has a structure similar to that of the discretization of a scalar PDE over a 3D mesh. The energy groups are solved sequentially, but the other five dimensions are all solved in parallel with essentially two levels of flexible GMRES preconditioned with the conjugate gradient method. For example, suppose that we have divided the geometry into $2^{10} = 1,024$ pieces and have a total of $2^8 = 256$ angles (each of which requires 1,024 cores for its geometry). The "inner solve" consists of 256 independent CG solves each running on 1,024 cores using any appropriate preconditioner.

The second application, PFLOTRAN [17, 25], performs continuum-scale simulations of multiscale, multiphase, multicomponent flow and reactive transport in geologic media. The application has been designed for parallel scalability, using PETSc parallel constructs for representing PDEs on structured and unstructured grids. PFLOTRAN also uses PETSc Newton-Krylov solvers, with the inner, linear system solves usually preconditioned with a single-level domain decomposition approach when solving

time-dependent problems. PFLOTRAN has been used to run problems with billions of degrees of freedom on leadership-class supercomputer platforms such as the Cray XT series and the IBM Blue Gene/P. In such large-scale simulations on the Cray XT machines, we have found that one of the biggest barriers to parallel scalability is the high cost of global reduction operations at large core counts (beginning at approximately 10,000 cores). In previous work, we implemented in PETSc an improved variant of BiCGStab (IBiCGStab) [13] that requires only one synchronization point per iteration, which reduced overall runtime by around 20 percent but still left global reductions dominating above 30,000 cores [26].

The UNIC code organizes Krylov solves hierarchically, thereby minimizing the effect of the global synchronization with `MPI_Allreduce()` because it requires few calls over the entire system; instead, most occur in inner solves that each run on smaller numbers of processes and are extremely fast. In contrast, in conventional Krylov approaches, as used in PFLOTRAN and most other applications, all the `MPI_Allreduce()` calls occur across the entire system. The key factor is that although all these applications benefit in numerical efficiency from the orthogonalization resulting from the numerous inner products in the Krylov methods, UNIC is structured in such a way that most of the inner products need not be performed across the entire system.

This observation motivates our development of hierarchical and nested Krylov methods, which reduce the number of global inner products required across the entire system (where they are expensive) in the outer iterations through hierarchical or nested inexact inner iterations that either freely allow vector inner products across small subsets of the entire system (where they are inexpensive) or use inner iterations that do not invoke vector inner products at all.

The concept of leveraging the natural hierarchies that arise in both numerical models and high-performance architectures is intuitive; and indeed the terms *hierarchical iterative methods*, *inner-outer iterations*, *nested solvers*, and *multilevel solvers* have been used in various contexts for approaches that exploit these concepts [27–30]. Related numerical software includes pARMS [31] and the Teko package of Trilinos [32]. Note that the term *nested solver* is also sometimes used to refer to mesh or grid sequencing; we do not use that terminology in this paper. We emphasize that the presented hierarchical Krylov methods are not multigrid methods. Rather, they are hierarchical in the sense of a hierarchy of nested embedded domains, not a hierarchy of meshes.

The remainder of the paper explains the details of our approach, with definitions in Section 2 of multilevel hierarchical and nested Krylov methods, followed by the definitions of specific algorithms to be studied, namely, *hierarchical FGMRES* (*h-FGMRES)* and *nested BiCGStab*. For the former, we employ inner-level inexact GMRES iterations hierarchically for subsystems within subgroups of cores, which have less synchronization overhead; for the latter, we employ a nested inner-level Chebyshev method [33, 34], which does not require any global operations. Section 3 explains our software strategy that enables these new composable algorithms. Section 4 describes the mathematical model and test cases of the motivating application, PFLOTRAN. Section 5 presents experimental results on the Cray XK6 for PFLOTRAN; we compare the performance of hierarchical FGMRES and nested BiCGStab methods with conventional GMRES and BiCGStab methods. Our experiments demonstrate that the hierarchical FGMRES and nested BiCGStab methods significantly reduce overall simulation time on the Cray XK6 when using 10,000 through 224,000 cores through the combined effects of stronger inner hierarchical or nested preconditioners as well as reduced global synchronization due to fewer global inner products. We also implemented a nested variant of IBiCGStab, thereby demonstrating that these hierarchical and nested Krylov methods can be used in conjunction with most of the synchronization/communication hiding and reducing variants of single-level Krylov methods mentioned above, thereby further reducing runtimes and improving scalability.

2. Hierarchical and Nested Krylov Algorithms

The objective of this work is to reduce global vector inner products in Krylov methods through hierarchical or nested multilevel iterations and thereby to reduce overall simulation time and improve scalability for large-scale PDE-based applications. In this context, we begin by depicting in Figure 1 the hierarchical partitioning of a global vector x. A segment at an upper level is called the *parent* of the segments at lower levels that are inherited from it. At level m, a partitioned segment is denoted by $x_{(i_1,\cdots,i_{m-1},i_m)}$, in which the first *m-1* indices $(i_1,\cdots,i_{m-1})$ are inherited from its parents, x, $x_{(i_1)}$, $\cdots$, $x_{(i_1,\cdots,i_{m-1})}$, and i_m is its own index. Figure 2 provides a simple example of a two-level hierarchical vector partition. While the small dimension of the vector in this diagram enables us to provide a concrete illustration of the notation introduced in Figure 1, in practice, hierarchical and nested Krylov methods are

$$
\begin{array}{ll}
\textit{Level 0:} & \overbrace{\qquad\qquad\qquad\qquad x \qquad\qquad\qquad\qquad}\\
\textit{Level 1:} & \overbrace{x_{(1)},\ x_{(2)},\qquad \cdots,\qquad x_{(i_1)},\qquad \cdots,\qquad x_{(n_1)}}\\
\textit{Level 2:} & \cdots,\overbrace{x_{(i_1,1),}\ x_{(i_1,2),}\cdots,x_{(i_1,i_2),}\cdots,x_{(i_1,n_2)}},\cdots,\\
 & \cdots\\
\textit{Level m-1:} & \cdots,\overbrace{x_{(i_1,i_2\cdots,i_{m-2},1),}\cdots,x_{(i_1,i_2,\cdots,i_{m-2},i_{m-1}),}\cdots,x_{(i_1,i_2,\cdots,i_{m-2},n_{m-1})}},\cdots\\
\textit{Level m:} & \cdots,\overbrace{x_{(i_1,i_2,\cdots,i_{m-1},1),}\cdots,x_{(i_1,i_2,\cdots,i_{m-1},i_m),}\cdots,x_{(i_1,i_2,\cdots,i_{m-1},n_m)}},\cdots
\end{array}
$$

Figure 1: Hierarchical vector partition. The *parent* global vector x on level 0 has n_1 subvector *children* on level 1, denoted by $x_{(1)}, x_{(2)}, \cdots, x_{(i_1)}, \cdots, x_{(n_1)}$. We emphasize that parenthesized subscripts indicate subvectors, not individual vector elements. For notational simplicity, on level 2 we illustrate the subvector children, $x_{(i_1,1)}, x_{(i_1,2)}, \cdots, x_{(i_1,i_2)}, \cdots, x_{(i_1,n_2)}$, of a single level-1 parent, $x_{(i_1)}$. All other level-1 entries also have their own subvector children on level 2. Likewise, levels *m-1* and m illustrate the subvector children of a single parent subvector on the immediately preceding level.

$$
\begin{array}{ll}
\textit{Level 0:} & x = [1,2,3,4,5,6,7,8,9,10]\\
\textit{Level 1:} & \overbrace{x_{(1)} = [1,2,3],\qquad x_{(2)} = [4,5,6,7],\qquad x_{(3)} = [8,9,10]}\\
\textit{Level 2:} & \overbrace{x_{(1,1)} = [1,2,3]},\overbrace{x_{(2,1)} = [4], x_{(2,2)} = [5,6], x_{(2,3)} = [7]},\overbrace{x_{(3,1)} = [8], x_{(3,2)} = [9,10]}
\end{array}
$$

Figure 2: Simple example of hierarchical vector partition with 2 levels, which illustrates the partitioning notation introduced in Figure 1.

intended for large-scale problems of minimum dimension $O(10^6)$ using at least $O(10^4)$ cores. Although hierarchical and nested approaches do reduce the number of outer level Krylov iterations for smaller problem sizes and fewer cores, these approaches do not generally reduce overall simulation time in this regime because global synchronization effects are not as severe.

Let x^j denote the computed approximation of the solution vector x at the jth iteration, and let $KSP^{(k)}$ represent a Krylov or other iterative method used at level k. We use the term *inexact KSP* for solving a linear system within a given convergence tolerance or with a (small) fixed number of iterations.

Algorithms 1 and 2 define multilevel hierarchical and nested Krylov methods for solving the linear system

$$Ax = b, \qquad \text{where} \qquad A \in C^{n\times n} \qquad x, b \in C^n. \tag{1}$$

Algorithm 1 Hierarchical Krylov approach for solving $Ax = b$

Partition the initial global vector x^0 into a hierarchy as shown in Figure 1; partition the total cores with same hierarchy.
Map the vector partition into the core partition.
for $j = 0, 1, \ldots$ until convergence (on all cores) **do**
  for level $k = 1, \ldots,$ m-1, recursively **do**
    level k-1: one $KSP^{(k-1)}$ iteration to either x^j (if k=1) or to $x^j_{(\cdots,i_{k-1})}$ (if k>1)
      with the following preconditioner:
    level k: for each child of x^j (or $x^j_{(\cdots,i_{k-1})}$), apply inexact $KSP^{(k)}$ iterations on the associated
      subgroup of cores.
  end for
end for

In this paper, we restrict our focus to two-level methods, which can be called *inner-outer* Krylov methods. For the hierarchical Krylov approach, the outer (top-level) solver iterates over all variables of a global problem, while inner (lower-level) solvers handle smaller subsets of physics on smaller physical

Algorithm 2 Nested Krylov approach for solving $Ax = b$

```
for j = 0, 1, ... until convergence (on all cores) do
    for level k = 1, ..., m-1, recursively do
        level k-1: Apply one KSP^(k-1) iteration to x^j with the following preconditioner:
        level k: Apply inexact KSP^(k) iterations to x^j.
    end for
end for
```

subdomains. Different types of Krylov methods $KSP^{(k)}$ can be chosen for the outer iteration and inner iterations on subgroups of cores. This approach greatly expands available preconditioners and adds tremendous flexibility to single-level Krylov methods. For example, problem partitioning and inner solver algorithms can be selected based on geometric and physics information for subdomains and subsets of variables.

For the algorithms described above, the lower-level (inner) iterations serve as preconditioners for the higher-level (outer) iterations. To facilitate discussion, we introduce the following definition.

Definition. When an iterative method is used as a preconditioner, if it employs only linear operations (and hence is a linear operator[1]), it is called a *linear iterative preconditioner*; otherwise it is a *nonlinear iterative preconditioner*. For example, Jacobi, Gauss-Seidel, SOR, SSOR, and Chebyshev methods (see, e.g., [35]) can be used as linear iterative preconditioners. Krylov methods, such as GMRES and CG, can be used only as nonlinear iterative preconditioners.

In order to maintain the convergence properties of outer Krylov iterations, the types of outer and inner iterative methods must be bound together in one of the following categories:

1. Outer: a *flexible* Krylov method that allows nonlinear iterative preconditioning;

   Inner: any iterative method, including Krylov methods such as GMRES or CG; or

2. Outer: any Krylov method;

   Inner: a linear iterative preconditioner.

GMRES [6] and its flexible variant FGMRES [29] are the most commonly used Krylov methods with well-established convergence properties. BiCGStab [12] with a block Jacobi or additive Schwarz (with small overlap) preconditioner, using one block per core and ILU(0) as the local solver, has often been the method of choice for PFLOTRAN, the application that has motivated this work. For highly time-dependent problems, this approach works well because the Jacobian systems are diagonally dominant, and at high processor core counts it is preferable to the multigrid approaches we have tried because of its lower communication costs. Since many Krylov iterations are often required for the Jacobian solves, BiCGStab is preferred over restarted GMRES because it uses short recurrences and avoids repeated orthogonalizations that have high communication costs at high core counts. The Chebyshev [34] method requires no inner products and has been used successfully in combination with other Krylov methods [33]. The spectrum of the Jacobian matrices for the PFLOTRAN cases studied in this paper (see Figure 6 in Section 5) suggest that the Chebyshev algorithm would be an optimal algorithm among the available linear iterative preconditioners.

Thus, in this paper we focus on two specific algorithms: *hierarchical FGMRES (h-FGMRES) with a GMRES nonlinear iterative preconditioner*, given by Algorithm 3, and *nested BiCGStab with a Chebyshev linear iterative preconditioner*, given by Algorithm 4. Because the inner-level solves serve as preconditioners, these can be inexact; in practice, we have found that just a few iterations, denoted by *inner_its* in Algorithms 3 and 4, suffice for the cases considered here. Alternatively, a (loose) convergence tolerance could be specified for the inner-level solves instead of a fixed number of iterations, though it is important not to waste resources in solving the inner iterations more accurately than needed in the context of the overall hierarchical or nested solver. Block Jacobi, using one block per core and ILU(0) as the local solver, is used as the innermost preconditioner for both algorithms.

Figure 3 shows the nonzero entries of the Jacobian matrix for PFLOTRAN case 1 (see Section 5) on a $4 \times 4 \times 4$ regular mesh partitioned across four cores. This diagram provides a complementary

[1] A linear operator L satisfies $L(ax) = aL(x)$ and $L(x + y) = L(x) + L(y)$.

Algorithm 3 Two-level h-FGMRES with a GMRES nonlinear iterative preconditioner

Partition the initial global vector x^0 into a hierarchy as shown in Figure 1.
Assign each initial level-2 subvector, $x^0_{(i_1,i_2)}$, to a core.
for $j = 0, 1, \ldots$ until convergence (on all cores) **do**
    one FGMRES iteration to x^j (over the global system) with the following preconditioner:
    for $k = 0, 1, \ldots, inner_its$ (concurrently on subgroups of cores) **do**
        for each child of x^j, one GMRES iteration (preconditioned with block Jacobi / ILU(0)).
    end for
end for

Algorithm 4 Nested BiCGStab with a Chebyshev linear iterative preconditioner

for $j = 0, 1, \ldots$ until convergence (on all cores) **do**
    one BiCGStab iteration over the global system with the following preconditioner:
    for $k = 0, 1, \ldots, inner_its$ (on all cores) **do**
        one Chebyshev iteration over the global system (preconditioned with block Jacobi / ILU(0)).
    end for
end for

perspective on the hierarchical FGMRES/GMRES approach given by Algorithm 3 and the nested BiCGStab/Chebyshev approach given by Algorithm 4. In each case the outer Krylov method addresses the entire global problem. Figure 4 uses the notation introduced in Figure 1 to illustrate the hierarchical vector partitioning that corresponds to the h-FGMRES matrix (on the left-hand side of Figure 3).

Although in practice one would apply hierarchical and nested Krylov approaches for much larger problems than this simple example, this matrix perspective facilitates consideration of the following, extreme, example of a hierarchical Krylov method based on FGMRES with a block Jacobi preconditioner on 2^n cores (for example, on $2^{17} = 131{,}072$ cores). We could employ flexible FGMRES using a two-block Jacobi preconditioner, with each block approximately solved on 2^{n-1} cores, and then recursively apply this algorithm to each of the two blocks, resulting in a hierarchy with n levels. A variant could be obtained by using an overlapping additive Schwarz preconditioner instead of block Jacobi. While one would generally expect to use fewer levels of hierarchy, this exercise introduces the basic concept extending the hierarchies beyond the two-level case illustrated in Figure 3.

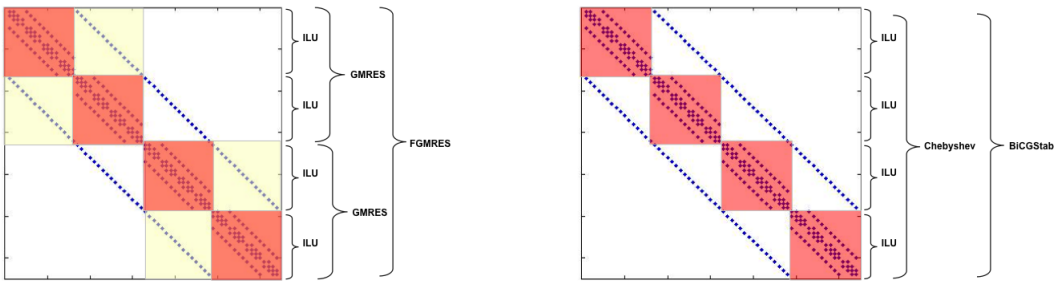

Figure 3: Illustration of hierarchical and nested Krylov approaches from the perspective of a sparse coefficient matrix A partitioned across four cores by contiguous blocks of rows; blue dots indicate nonzero entries of the matrix. *Left:* hierarchical FGMRES/GMRES (given by Algorithm 3). Two submatrices highlighted in yellow indicate level-1 partitioning, while four submatrices highlighted in red indicate level-2 partitioning; the corresponding vectors x and b are partitioned identically, as shown in Figure 4. *Right:* BiCGStab/Chebyshev given by Algorithm 4. In both cases the innermost preconditioner is block Jacobi with one block per processor and ILU(0) as the local solver.

3. Software for Composable Sparse Linear Solvers

Two design principles of the PETSc library that are essential for managing the software complexity of the hierarchical and nested approaches introduced in Section 2 are *object-oriented design* and *consistent*

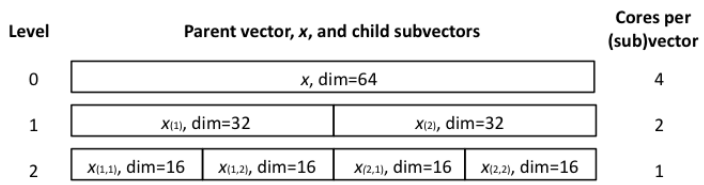

Figure 4: Illustration of hierarchical vector partitioning introduced in Figure 1 corresponding to the matrix illustrated in the left-hand side of Figure 3; (sub)vector dimensions are indicated at each level.

use of MPI communicators for all data and solver objects [22, 36]. The emerging era of extreme-scale computing provides new imperatives that reinforce the importance of good software engineering to enable the development of *new composable algorithms* from reusable, well-tested pieces. Such capabilities enable the creation of novel schemes tailored to exploit both hierarchical architectural features and increasingly complex modeling, including custom approaches for multiphysics problems [37].

A third principle that is important for managing numerical software complexity for emerging extreme-scale architectures is *separation of the control logic of the algorithms from the computational kernels of the solvers* [38]. This is crucial to allow injecting new hardware-specific computational kernels without having to rewrite the entire solver software library. In particular, though beyond the scope of this paper, these hierarchical and nested Krylov methods can readily employ new computational kernels for GPUs [39] and hybrid multicore programming models [40].

As further discussed in Section 4, PFLOTRAN employs the hierarchical and nested Krylov solvers as part of an implicit time advance, where a nonlinear system is solved at each timestep by a preconditioned Newton-Krylov method using PETSc's nonlinear solvers. The nonlinear solvers in turn can employ a broad range of Krylov solvers and preconditioners, all accessible through a common interface, so that particular algorithms and parameters can be selected at runtime. Section 5 indicates the runtime options used to activate the hierarchical and nested Krylov solvers in Algorithms 3 and 4.

4. Motivating Application: PFLOTRAN

We have tested these hierarchical and nested Krylov methods in PFLOTRAN, a state-of-the-art code for simulating multiscale, multiphase, multicomponent flow and reactive transport in geologic media on machines ranging from laptops to leadership-class supercomputers. PFLOTRAN solves a coupled system of mass and energy conservation equations for a number of phases, including air, water, and supercritical CO_2, and a number of chemical components. The code utilizes finite volume or mimetic finite difference spatial discretizations combined with backward-Euler (fully implicit) timestepping for the flow and reactive transport solves or, optionally, operator splitting for the reactive transport. PFLOTRAN, which is built on PETSc, makes extensive use of iterative nonlinear and linear solvers, distributed linear algebra data structures, parallel constructs for representing PDEs on structured and unstructured grids, performance logging, runtime control of solvers and other options, and parallel binary I/O.

Although PFLOTRAN can simulate complicated systems, in this study we restrict the benchmark problems to isothermal simulations of variably saturated porous media flow (no transport) as described by Richards' equation. In the mixed form (containing both pressure and saturation), the governing equation is

$$\frac{\partial}{\partial t}\big(\varphi s \rho\big) + \boldsymbol{\nabla} \cdot \rho \boldsymbol{u} = \mathcal{S},$$

where φ denotes the porosity of the geologic medium, s the saturation (fraction of pore volume filled with liquid water), ρ the fluid density, $\mathcal{S}$ a source/sink term representing water injection/extraction, and $\boldsymbol{u}$ the Darcy velocity defined as

$$\boldsymbol{u} = -\frac{\kappa \kappa_r}{\mu} \boldsymbol{\nabla}\big(P - \rho g z\big),$$

where P denotes fluid pressure, μ viscosity, κ the absolute permeability of the medium, κ_r the relative permeability of water to air, g the acceleration of gravity, and z the vertical distance from a datum. The relative permeability is a nonlinear function of saturation, and this nonlinearity is responsible for most of the numerical difficulties encountered in variably saturated flow problems. Several models for this relationship exist; we use van Genuchten [41] relative permeability functions in this study.

To numerically solve the governing equations, in this study we use a backward-Euler time discretization and a first-order finite volume spatial discretization on a uniformly spaced rectilinear grid. At cell interfaces, absolute permeabilities are calculated by harmonic averaging and one-point upstream weighting is used for relative permeability calculations. The resulting system of nonlinear algebraic equations is solved via Newton-Krylov methods using PETSc's Scalable Nonlinear Equations Solvers (SNES).

We evaluate the proposed hierarchical and nested Krylov algorithms using two PFLOTRAN test cases. We consider only flow problems here. In coupled flow and reactive transport simulations, PFLOTRAN sequentially couples flow and reactive transport and performs separate flow and reactive transport solves. Both solves can be computationally expensive; but flow solves tend to require many iterations, and involve proportionally less local work than does the reactive transport solve, and therefore suffer more from communication bottlenecks—hence our focus on flow problems. We describe the two test cases below. In our numerical experiments, we run each test case for the minimum number of time steps required to obtain a reasonable picture of the basic physics seen in that case.

Case 1: Cubic domain with a central injection well; number of time steps: 6.

This case models a 100 m $\times$ 100 m $\times$ 100 m domain with a uniform effective permeability of 1 darcy and an injection well at the exact center. The domain is fully saturated, and the initial pressure distribution follows a hydrostatic gradient; hydrostatic equilibrium is enforced at the domain boundaries. At time zero, injection begins at a rate of 10 m^3 per hour; the sharp change in pressure (see Figure 5) results in difficult-to-solve algebraic systems during the first several time steps. We note that although we scale up the grid resolution as we increase processor core counts in our experiments, the presence of the well renders this case not useful for assessing weak scalability: injection at the well (represented as a point in the conceptual model) will be spread across a smaller volume as the grid resolution is increased, resulting in somewhat different physical behavior at different grid spacings.

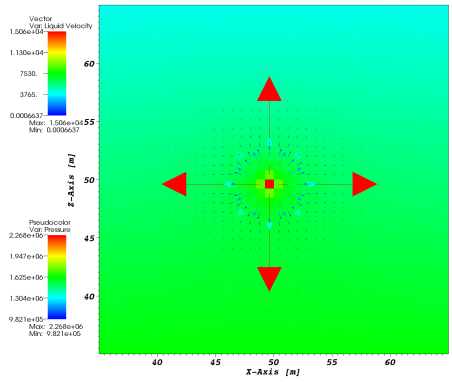

Figure 5: A portion of a 2D slice through the middle the y-axis (z is vertical) for a version of the case 1 problem run using a 128 $\times$ 128 $\times$ 128 grid. The plotted vector field shows that the velocities (in units of m/yr) are large near the center (and essentially zero far away), and a pseudocolor plot of the pressure field (in Pascals) shows that pressure mostly varies in accordance with the hydrostatic gradient but is high in the center cell where the injection well is located). As the grid resolution is refined, the center of the domain will display the same basic pattern but the high velocity/high pressure region around the well will occupy a smaller portion of the domain.

Case 2: Regional flow without well near river; number of time steps: 2.

This case models a 5000 m $\times$ 2500 m $\times$ 100 m region with a river at the eastern boundary. The subsurface is divided into four horizontal stratigraphic units, with homogeneous material properties throughout each unit. The domain is variably saturated with the water table initially located 10 m below the surface at the western boundary and sloping gradually toward the river. No flow conditions are imposed at the north, south, and bottom boundaries; hydrostatic equilibrium is imposed at the western boundary. At the eastern boundary, the river is represented by imposing a seepage face condition: a hydrostatic pressure gradient is imposed below the river stage, and atmospheric pressure is imposed above it. A recharge condition of 25 cm/yr is imposed across the top boundary. Because no wells are included in the domain, one can draw some conclusions about weak scalability using this benchmark.

5. Numerical Experiments

In this section we present numerical results of Algorithms 3 and 4 described in Section 2 on PFLOTRAN cases 1 and 2. Experiments were conducted on a Cray XK6 system located at the Oak Ridge Leadership Computing Facility [42], with 18,688 compute nodes, each consisting of one AMD 16-core Opteron 6274 processor running at 2.2 GHz, 32 gigabytes of DDR3 memory, giving a total of 299,008 cores. The new high-performance Gemini network provides high bandwidth, lower latency, faster collectives, and greater reliability than did previous generations of Cray machines.

For a given number of cores, np, the vectors x, b, and matrix A in Equation (1) are partitioned contiguously into np row blocks, each of which is assigned to a core. In what follows, we refer to this mapping between the Equation (1) and the np total cores as a *nonhierarchical system distribution*. The diagonal blocks of the distributed matrix A, shown by the red submatrices in Figure 3, form the innermost preconditioning submatrices. In all experiments the innermost preconditioner is block Jacobi, using one block per core and ILU(0) as the solver on each block.

Comparison of GMRES and hierarchical FGMRES (h-FGMRES). Table 1 compares GMRES (indicated by $ngp = 1$ in column 2) and h-FGMRES (indicated by $ngp > 1$ in column 2) for PFLOTRAN case 1. For both cases we employed 30 as the GMRES restart parameter. To implement the 2-level h-FGMRES hierarchy, we group the nonhierarchical system distribution on np cores described above into ngp subgroups, each containing np/ngp cores and holding a corresponding subsystem of Equation (1). For each outer FGMRES iteration, 6 GMRES inner iterations with preconditioner block Jacobi / ILU(0) were concurrently applied to each subsystem using the level-1 subgroups of cores. At a given timestep, if the outer linear iterations fail to reach the given convergence criterion after a maximum of 10,000 iterations, the timestep is cut, and the computation restarts. Column 3 indicates the number of timestep cuts. For GMRES, the number of blocks used in the block Jacobi preconditioner equals the number of cores, np. As the number of cores increases, the preconditioner becomes weaker, causing more failures and thus more timestep cuts. Two-level h-FGMRES is equivalent to using ngp diagonal blocks of matrix A as a preconditioning matrix. Because $ngp \ll np$, the blocks are much larger and retain more Jacobian data, resulting in more robust preconditioning than with the nonhierarchical approach and thereby avoiding timestep cuts in all test cases. Column 4 displays the percentage of total execution time consumed by vector inner products in both outer and inner iterations.

Table 1: Comparison of GMRES and 2-level h-FGMRES for PFLOTRAN Case 1.

Number of Cores (np) (mesh size)	Groups of Cores (ngp)	Timestep Cuts	% Time for Inner Products	Outer Iterations	Execution Time (sec)
512	1	0	28	3,853	43.9
(256x256x256)	16	0	17	903	45.8
4,096	1	0	39	11,810	146.5
(512x512x512)	64	0	23	2,405	126.7
32,768	1	1	48	35,177	640.5
(1024x1024x1024)	128	0	28	5,244	297.4
98,304	1	7	77	59,250	1346.0
(1024x1024x1024)	128	0	47	6,965	166.2
160,000	1	9	72	59,988	1384.1
(1600x1600x640)	128	0	51	8,810	232.2

As shown by column 5, h-FGMRES significantly reduced the number of outer Krylov iterations (and thus global `MPI_Allreduce()` calls), thereby reducing the fraction of overall runtime for inner products on 98,304 cores from 77% for conventional GMRES to 47% for h-FGMRES, and on 160,000 cores from 72% for conventional GMRES to 51% for h-FGMRES. Moreover, the inner h-FGMRES iterations served as a stronger preconditioner than block Jacobi / ILU(0) alone, leading to a more robust solver that overcame the timestep cuts needed by conventional GMRES. The impact of h-FGMRES on overall PFLOTRAN simulation time is profound at high core counts: on 98,304 and 160,000 cores, h-FGMRES reduced overall runtime by factors of 8 and 6, respectively, compared with conventional GMRES.

In our tests, the total number of cores is divided equally into ngp subgroups, chosen as a multiple of 16, the number of cores in a processor node of the Cray XK6. As expected, during experiments we observed that the algorithmic performance is sensitive to problem partitioning and that respecting

the physics of a problem (for example, not partitioning in the midst of the well region for case 1), is important. Because the h-FGMRES approach provides enormous flexibility in possible choices of inner-level solvers for subproblems, it is feasible to apply custom partitionings and inner-level solvers with varying degrees of cost and robustness based on underlying physics and geometry as well as machine topology; such considerations are aspects of future research.

We emphasize that all algorithmic experiments required no source code changes to PFLOTRAN; rather, the application simply activated the new hierarchical and nested Krylov solvers by using PETSc runtime options. For example, the runtime options for conventional GMRES are given below, where the prefix `-flow` denotes PFLOTRAN's flow solver (the focus of this work).

```
./pfltran -pflotranin <pflotran_input>
-flow_ksp_type gmres -flow_ksp_pc_side right -flow_pc_type bjacobi -flow_pc_bjacobi_blocks np
-flow_sub_pc_type ilu
```

Likewise, the following are runtime options for two-level h-FGMRES, given by Algorithm 3, where the prefix `-sub` (or `-flow_sub` for PFLOTRAN's flow solver) indicates algorithmic choices for the inner level of the hierarchy. Here we use 6 iterations of GMRES, preconditioned by block Jacobi / ILU(0), although more generally, any appropriate linear solver could be employed.

```
./pfltran -pflotranin <pflotran_input>
-flow_ksp_type fgmres -flow_ksp_pc_side right -flow_pc_type bjacobi -flow_pc_bjacobi_blocks ngp
-flow_sub_ksp_type gmres -flow_sub_ksp_max_it 6 -flow_sub_pc_type bjacobi
-flow_sub_sub_pc_type ilu
```

Comparison of BiCGStab and nested BiCGStab/Chebyshev. The linear solvers of choice for PFLOTRAN have been BiCGStab and IBiCGStab, both with a block Jacobi / ILU(0) preconditioner. One of our goals in this work is to further improve their performance by adding a nested linear iterative preconditioner, as introduced generally in Algorithm 2. Among all available linear iterative preconditioners, the Chebyshev method is likely optimal if (1) the spectrum of the Jacobian matrix is located inside an enveloping ellipse that does not include the origin and (2) the approximate extreme eigenvalues can be computed efficiently during implementation of the Chebyshev algorithm. We investigated the spectra of PFLOTRAN Jacobian matrices for test cases 1 and 2 and confirmed that property (1) is satisfied. The Chebyshev implementation provided by the PETSc library is a variant of the *hybrid Chebyshev Krylov subspace algorithm* without purification [33]. For a newly updated matrix, this algorithm applies a small number of GMRES iterations to approximate extreme eigenvalues, which are then fed to the Chebyshev iterations. The cost of eigenvalue estimate is negligible in our tests.

Table 2 compares conventional BiCGStab (denoted by 0 inner iterations in column 2) with nested BiCGStab (denoted by 4 inner iterations in column 2) for PFLOTRAN's test case 1. The BiCGStab algorithm employs the preconditioner twice during each iteration; for the nested BiCGStab approach, each application of the preconditioner employs two Chebyshev iterations, which themselves are preconditioned by block Jacobi / ILU(0). Column 4 displays the percentage of overall execution time for global vector inner products. On large numbers of cores (e.g., $np = 98{,}304$ and 160,000) the ratios of time savings in these global synchronizations (column 4) are almost the same as the savings in overall execution time (column 5).

Table 2: Comparison of BiCGStab and nested BiCGStab for PFLOTRAN case 1.

Number of Cores (np) (mesh size)	Inner Iterations	Outer Iterations	% Time for Inner Products	Execution Time (sec)
512	0	1237	10	18.8
(256x256x256)	4	565	6	24.3
4,096	0	2649	28	46.5
(512x512x512)	4	1049	12	45.8
32,768	0	4875	36	95.7
(1024x1024x1024)	4	1998	19	94.8
98,304	0	5074	64	66.7
(1024x1024x1024)	4	1965	40	43.0
160,000	0	6240	64	90.6
(1600x1600x640)	4	2641	50	69.2

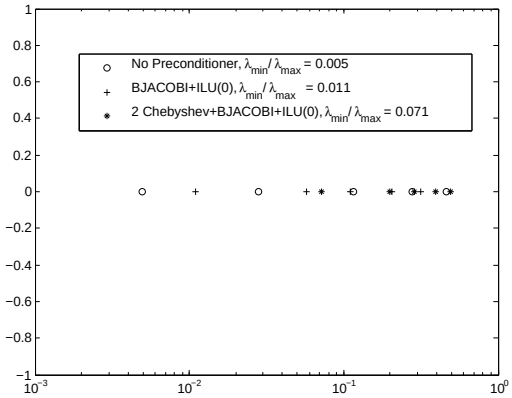

Figure 6: Smallest few eigenvalues of Jacobian matrix, scaled by largest eigenvalue (i.e., $1/\lambda_{max}$) with and without preconditioner.

Again, no changes to PFLOTRAN were needed for experimenting with these algorithms. The PETSc runtime options to activate conventional BiCGStab for PFLOTRAN's flow solver are as follows:

```
./pfltran -pflotranin <pflotran_input>
-flow_ksp_type bcgs -flow_ksp_pc_side right -flow_pc_type bjacobi -flow_pc_bjacobi_blocks np
-flow_sub_pc_type ilu
```

Likewise, the runtime options for nested BiCGStab with the Chebyshev preconditioner, given by Algorithm 4, are as follows:

```
./pfltran -pflotranin <pflotran_input>
-flow_ksp_type bcgs -flow_ksp_pc_side right -flow_pc_type ksp
-flow_ksp_ksp_type chebyshev -flow_ksp_ksp_chebychev_estimate_eigenvalues 0.1,1.1
-flow_ksp_ksp_max_it 2 -flow_ksp_ksp_norm_type none -flow_ksp_pc_type bjacobi
-flow_ksp_sub_pc_type ilu
```

For further insight, we plot in Figure 6 the estimated smallest few eigenvalues scaled by $1/\lambda_{max}$ for a Jacobian matrix using a mesh of dimension $32 \times 32 \times 32$, distributed among 64 cores. We consider three cases: (1) no preconditioner, (2) a preconditioner of block Jacobi / ILU(0), and (3) a preconditioner of two Chebyshev iterations, which themselves are preconditioned by block Jacobi / ILU(0). The eigenvalues are computed through GMRES or FGMRES outer iterations with restart 300. As indicated by the estimated values $\lambda_{min}/\lambda_{max}$, two Chebyshev iterations with block Jacobi / ILU(0) is a stronger preconditioner than just block Jacobi / ILU(0), thus reducing the outer iterations of the flow solver, as reported in Table 2.

Comparison of BiCGStab, IBiCGStab, and nested IBiCGStab/Chebyshev. The hierarchical and nested Krylov approaches presented in this paper can be used in conjunction with most of the recently developed Krylov methods mentioned in Section 1 that reduce or hide communication and

Table 3: Comparison of BiCGStab, IBiCGStab, and nested IBiCGStab for PFLOTRAN case 1.

Number of Cores	BiCGS		IBiCGS		Nested IBiCGS	
grids	Outer Iters	Time (sec)	Outer Iters	Time (sec)	Outer Iters	Time (sec)
512						
256x256x256	1237	18.8	1364	21.3	546	24.2
4,096						
512x512x512	2649	46.5	2668	41.5	1033	44.1
32,768						
1024x1024x1024	4875	95.7	5219	85.0	2073	89.0
98,304						
1024x1024x1024	5074	66.7	5219	37.8	2039	33.4
160,000						
1600x1600x640	6240	90.6	6904	66.6	2407	52.0

Table 4: BiCGStab, IBiCGStab and nested IBiCGStab for PFLOTRAN case 2.

Number of Cores	BiCGS		IBiCGS		Nested IBiCGS	
grids	Outer Iters	Time (sec)	Outer Iters	Time (sec)	Outer Iters	Time (sec)
1,600 800x408x160	1003	16.6	1046	16.4	411	18.0
16,000 1600x816x320	2378	38.7	2265	32.7	829	29.6
80,000 1600x1632x640	4304	81.4	4397	60.2	1578	53.1
160,000 1600x1632x640	4456	71.3	4891	38.7	1753	31.9
192,000 1600x1632x640	3207	41.5	4427	29.8	1508	22.2
224,000 1600x1632x640	4606	55.2	4668	29.3	1501	20.9

synchronization, for example, IBiCGStab [13]. To demonstrate this capability, we implemented nested IBiCGStab by replacing the outer BiCGStab iteration with the IBiCGStab iteration in Algorithm 4; that is, we used identical runtime options of the nested BiCGStab but replaced "`-flow_ksp_type bcgs`" with "`-flow_ksp_type ibcgs`."

Tables 3 and 4 compare the numbers of outer iterations and execution time of the BiCGStab, IBiCGStab, and nested IBiCGStab methods for PFLOTRAN cases 1 and 2, respectively. Figure 7 shows the percentage of overall execution time of IBiCGStab and nested IBiCGStab compared with BiCGStab for test case 2. The impact of the nested IBiCGStab approach is profound at very high core counts. For example, for test case 2 on 80,000 cores, the nested IBiCGStab approach reduced overall runtime to 65% of BiCGStab and 88% of IBiCGStab, while on 224,000 cores, the nested IBiCGStab approach reduced overall runtime to only 37% of BiCGStab and 71% of IBiCGStab.

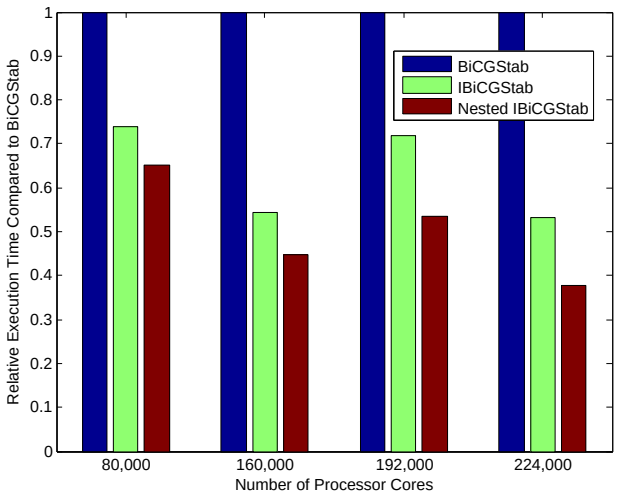

Figure 7: Comparison of BiCGStab, IBiCGStab, and nested IBiCGStab for PFLOTRAN case 2.

6. Conclusions

This paper introduced hierarchical and nested Krylov variants to overcome well-known scaling difficulties caused by global reductions in standard Krylov approaches for extreme numbers of processor cores. We introduced the *hierarchical FGMRES method*, or *h-FGMRES*, and we demonstrated the impact at high core counts of two-level h-FGMRES, using GMRES as an inner-level preconditioner, on the PFLOTRAN subsurface flow application. We also demonstrated the impact of a nested BiCGStab/Chebyshev

algorithm. Because these algorithms can be activated at runtime, with no changes to application source code, other application codes that employ the PETSc library can easily experiment with such techniques. These principles of hierarchal and nested Krylov methods can be applied to complementary advances in synchronization/communication-avoiding Krylov methods, thereby offering potentially multiplicative benefits of combined approaches.

Future work includes exploring additional inner/outer algorithmic pairs and extending the hierarchies and nesting to additional levels, especially as we consider emerging extreme-scale architectures. Also important is incorporating physics insight and machine topology into hierarchical partitioning and algorithmic selection, potentially through automated strategies over the longer term.

Acknowledgments

We thank Satish Balay and Jed Brown for insightful discussions and assistance. We gratefully acknowledge use of the Cray XE6 (Hopper) computer at NERSC for preliminary studies on hierarchical Krylov methods. This research used resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the Department of Energy under Contract DE-AC05-00OR22725. L.C. McInnes, B. Smith, and H. Zhang were supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357. R. Mills was partially supported by the Climate and Environmental Sciences Division of the Office of Biological and Environmental Research (BER) and the Computational Science Research and Partnerships Division of the Office of Advanced Scientific Computing Research within the U.S. Department of Energy's Office of Science.

References

[1] Y. Saad, Iterative Methods for Sparse Linear Systems, 2nd edition, SIAM, Philadelpha, PA, 2003.

[2] D. Brown, P. Messina (Chairs), Scientific Grand Challenges: Crosscutting Technologies for Computing at the Exascale (2010).
URL `http://extremecomputing.labworks.org/SumReps/Crosscutting-SumRep-PNNL_20168.pdf`

[3] MPI: A message-passing interface standard, International J. Supercomputing Applications 8 (3/4).

[4] J. Ang, K. Evans, A. Geist, M. Heroux, P. Hovland, O. Marques, L. McInnes, E. Ng, S. Wild, Report on the workshop on extreme-scale solvers: Transitions to future architectures, Office of Advanced Scientific Computing Research, U.S. Department of Energy, Washington, DC, March 8-9, 2012 (2012).
URL `http://science.energy.gov/~/media/ascr/pdf/program-documents/docs/reportExtremeScaleSolvers2012.pdf`

[5] M. Mohiyuddin, M. Hoemmen, J. Demmel, K. Yelick, Minimizing communication in sparse matrix solvers, in: Proceedings of SC09, ACM, 2009. doi:10.1145/1654059.1654096.

[6] Y. Saad, M. H. Schultz, GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 7 (1986) 856–869.

[7] J. van Rosendale, Minimizing inner product data dependencies in conjugate gradient iteration, in: Proceedings of the IEEE International Conference on Parallel Processing, IEEE Computer Society, 1983.

[8] A. Chronopoulos, C. W. Gear, S-step iterative methods for symmetric linear systems, Journal of Computational and Applied Mathematics 25 (1989) 153–168.

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

[10] R. Vuduc, Quantitative performance modeling of scientific computations and creating locality in numerical algorithms, Ph.D. thesis, Massachusetts Institute of Technology (1995).

[11] P. Ghysels, T. Ashby, K. Meerbergen, W. Vanroose, Hiding global communication latency in the GMRES algorithm on massively parallel machines.
URL `http://www.exascience.com/wp-content/uploads/2012/02/Ghysels_latency_gmres.pdf`

[12] H. van der Vorst, BiCGSTAB: A fast and smoothly converging variant of BiCG for the solution of nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 13 (1992) 631–644.

[13] L. T. Yang, R. Brent, The improved BiCGStab method for large and sparse unsymmetric linear systems on parallel distributed memory architectures, in: Proceedings of the Fifth International Conference on Algorithms and Architectures for Parallel Processing, IEEE, 2002.

[14] D. Kaushik, M. Smith, A. Wollaber, B. Smith, A. Siegel, W. S. Yang, Enabling high fidelity neutron transport simulations on petascale architectures, SC'09 Gordon Bell Prize Finalist (2009).

[15] G. Palmiotti, M. A. Smith, C. Rabiti, M. Leclere, D. Kaushik, A. Siegel, B. Smith, E. E. Lewis, UNIC: Ultimate neutronic investigation code, in: Joint International Topical Meeting on Mathematics and Computation and Supercomputing in Nuclear Applications, 2007.

[16] P. Lichtner et al., PFLOTRAN project, `http://ees.lanl.gov/pflotran/`.

[17] C. Lu, P. C. Lichtner, PFLOTRAN: Massively parallel 3-D simulator for CO2 sequestration in geologic media, in: Fourth Annual Conference on Carbon Capture and Sequestration DOE/NETL, 2005.

[18] R. T. Mills, V. Sripathi, G. Mahinthakumar, G. Hammond, P. C. Lichtner, B. F. Smith, Engineering PFLOTRAN for scalable performance on Cray XT and IBM BlueGene architectures, in: Proceedings of SciDAC 2010 Annual Meeting, 2010.

[19] R. Mills, C. Lu, P. Lichtner, G. Hammond, Simulating subsurface flow and transport on ultrascale computers using PFLOTRAN, in: Journal of Physics: Conference Series, Vol. 78, IOP Publishing, 2007, p. 012051.

[20] R. Mills, G. Hammond, P. Lichtner, V. Sripathi, G. Mahinthakumar, B. Smith, Modeling subsurface reactive flows using leadership-class computing, Journal of Physics: Conference Series 180 (2009) 012062.

[21] R. T. Mills, F. M. Hoffman, P. H. Worley, K. S. Pwerumalla, A. Mirin, G. E. Hammond, B. Smith, Coping at the user-level with resource limitations in the Cray message passing toolkit MPI at scale, in: Cray Users Group Conference, 2009.

[22] S. Balay, W. D. Gropp, L. C. McInnes, B. F. Smith, Efficient management of parallelism in object oriented numerical software libraries, in: E. Arge, A. M. Bruaset, H. P. Langtangen (Eds.), Modern Software Tools in Scientific Computing, Birkhauser Press, 1997, pp. 163–202.
URL `ftp://info.mcs.anl.gov/pub/tech_reports/reports/P634.ps.Z`

[23] S. Balay, J. Brown, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, H. Zhang, PETSc users manual, Tech. Rep. ANL-95/11 - Revision 3.3, Argonne National Laboratory (2012).
URL `http://www.mcs.anl.gov/petsc`

[24] B. F. Smith, W. D. Gropp, The design of data-structure-neutral libraries for the iterative solution of sparse linear systems, Scientific Programming 5 (1996) 329–336.

[25] P. Lichtner (PI), Modeling Multiscale-Multiphase-Multicomponent Subsurface Reactive Flows Using Advanced Computing, `http://ees.lanl.gov/pflotran/`.

[26] R. Mills, G. Hammond, P. Lichtner, V. Sripathi, G. Mahinthakmuar, B. Smith, Modeling subsurface reactive flows using leadership-class computing, in: Journal of Physics: Conference Series, Vol. 180, IOP Publishing, 2009, p. 102062.

[27] E. Giladi, G. H. Golub, J. B. Keller, Inner and outer iterations for the Chebyshev algorithm, SIAM J. Numer. Anal 35 (1995) 300–319.

[28] A. El maliki, R. Guenette, M. Fortin, An efficient hierarchical preconditioner for quadratic discretizations of finite element problems, Numerical Linear Algebra with Applications 18 (5) (2011) 789–803. doi:10.1002/nla.757.

[29] Y. Saad, A flexible inner-outer preconditioned GMRES algorithm, SIAM Journal on Scientific Computing 14 (2) (1993) 461–469. doi:10.1137/0914028.
URL `http://dx.doi.org/10.1137/0914028`

[30] V. Simoncini, D. Szyld, Flexible inner-outer Krylov subspace methods, SIAM Journal on Numerical Analysis (2003) 2219–2239.

[31] Y. Saad, M. Sosonkina, pARMS: A package for the parallel iterative solution of general large sparse linear systems user's guide, Tech. Rep. UMSI2004-8, Minnesota Supercomputer Institute, University of Minnesota, `http://www-users.cs.umn.edu/~saad/software/pARMS` (2004).

[32] M. A. Heroux, J. M. Willenbring, Trilinos users guide, Tech. Rep. SAND2003-2952, Sandia National Laboratories (2003).
URL `http://trilinos.sandia.gov/`

[33] H. Elman, Y. Saad, P. E. Saylor, A hybrid Chebyshev Krylov-subspace algorithm for solving nonsymmetric systems of linear equations, SIAM Journal on Scientific and Statistical Computing 7 (3) (1986) 840–855.

[34] G. Golub, R. Varga, Chebyshev semi-iterative methods, successive overrelaxation iterative methods, and second-order richardson iterative methods, parts I and II, Numer. Math. 3 (1961) 147–168.

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

[36] B. Smith, Encyclopedia of Parallel Computing, Springer, 2011, Ch. PETSc, the Portable, Extensible Toolkit for Scientific computing.

[37] J. Brown, M. G. Knepley, D. A. May, L. C. McInnes, B. F. Smith, Composable linear solvers for multiphysics, Preprint ANL/MCS-P2017-0112, Argonne National Laboratory, to appear in 11th International Symposium on Parallel and Distributed Computing (ISPDC 2012) (2012).

[38] B. Smith, L. C. McInnes, E. Constantinescu, M. Adams, S. Balay, J. Brown, M. Knepley, H. Zhang, PETSc's software strategy for the design space of composable extreme-scale solvers, Preprint ANL/MCS-P2059-0312, Argonne National Laboratory, DOE Exascale Research Conference, April 16-18, 2012, Portland, OR (2012).

[39] V. Minden, B. F. Smith, M. G. Knepley, Preliminary implementation of PETSc using GPUs, in: Proceedings of the 2010 Workshop of GPU Solutions to Multiscale Problems in Science and Engineering, 2012.

[40] S. Abhyankar, B. Smith, K. Stevens, Preliminary implementation of hybrid MPI/pthread programming model in PETSc, Preprint ANL/MCS-P2011-0112, Argonne National Laboratory (2012).

[41] M. T. van Genuchten, A closed-form equation for predicting the hydraulic conductivity of unsaturated soils, Soil Science Society of America Journal 44 (1980) 892–898.

[42] OLCF JAGUAR Supercomputer, `https://www.olcf.ornl.gov/computing-resources/jaguar/`.

Government License. The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory ("Argonne"). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.