On the Use of Algebraic Multigrid in Applications on High Performance Computers

Robert D. Falgout[0000−0003−4884−0087], Rui Peng Li[0000−0003−2802−5763], Wayne Mitchell[0000−0003−4397−481X], Daniel Osei-Kuffuor[0000−0002−6111−6205], Victor A. P. Magri[0000−0002−3389−523X], and Ulrike M. Yang[0000−0002−6957−0445]

Abstract The *hypre* software library provides a variety of parallel linear solvers designed for high performance computers. It focuses on algebraic multigrid methods (AMG), which provide excellent scalability. With the expanding use of accelerators in current and future high-performance computers, new programming models have been added to take advantage of the increased performance potential of graphics processing units (GPUs). This chapter will discuss porting strategies and present results for *hypre*'s GPU-enabled multigrid solvers within three application codes.

1 Introduction

The solution of large linear systems is an essential and often computationally expensive component of many large-scale simulations on high-performance computing (HPC) machines. Such simulations would not be feasible without linear solvers that are algorithmically scalable and tailored to particular HPC architectures. These demands have only increased in the new age of exascale computers, where performance is achieved through vast parallelism and highly specialized GPU architectures.

The *hypre* [12, 18] library of high-performance solvers and preconditioners is developed and maintained at Lawrence Livermore National Laboratory (LLNL) and provides scalable and portable algorithms that are relied upon by many large-scale application codes around the world. The goals of *hypre* are three-fold: first, to provide a variety of general and adaptable interfaces that can be easily used across many disparate application areas (Section 2); second, to provide scalable and effective algorithms for a wide set of applications and problem sizes (Section 2); and finally, to provide performant and portable implementations of these solvers on a wide

R.D. Falgout, R. Li, W. Mitchell, D. Osei-Kuffuor, V.A.P. Magri, U.M. Yang
Lawrence Livermore National Laboratory,
Center for Applied Scientific Computing, Livermore, CA, USA
e-mail: {falgout2, li50, mitchell82, oseikuffuor1, paludettomag1, yang11}@llnl.gov

variety of computing architectures (Section 3). Through pursuing these goals, *hypre* has seen widespread success in application codes, and three examples are given here (Section 4). The applications cover three scientific areas: combustion, subsurface flow, and computational fluid dynamics (CFD); they use different *hypre* interfaces and solvers and run on HPC systems, including the exascale computer Frontier at Oak Ridge National Laboratory (ORNL), with scalability demonstrated up to tens of thousands of GPUs. The paper concludes with a synthesis of the findings and future research directions in Section 5.

2 Interfaces and solvers

The *hypre* library provides three different interfaces [12] to allow users to represent their problems in a way they prefer: a structured interface that uses grids and stencils, a semi-structured interface for problems that are mostly structured, such as adaptive mesh refinement grids, or block-structured grids, defined with stencils or finite elements, and a linear-algebraic interface, where problems are expressed in terms of matrices and vectors.

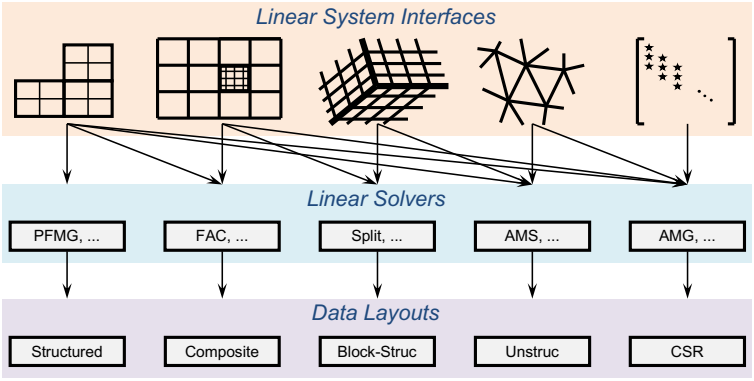

Fig. 1: Illustration of the idea behind *hypre*'s conceptual interfaces.

Figure 1 shows the notion of *hypre*'s conceptual interfaces and how they connect to solvers. The top row shows various concepts, such as structured grids, composite grids, block-structured grids, unstructured grids, or standard matrices. The second row lists various solvers and preconditioners. These require different information from the user provided through the interfaces. The bottom row lists different data layouts. The most efficient solvers, such as PFMG, can only be used through the structured interface on the left. In contrast, unstructured solvers such as (Boomer)AMG can be used with all interfaces, albeit at a cost, since they use less efficient data layouts but can solve more complex problems.

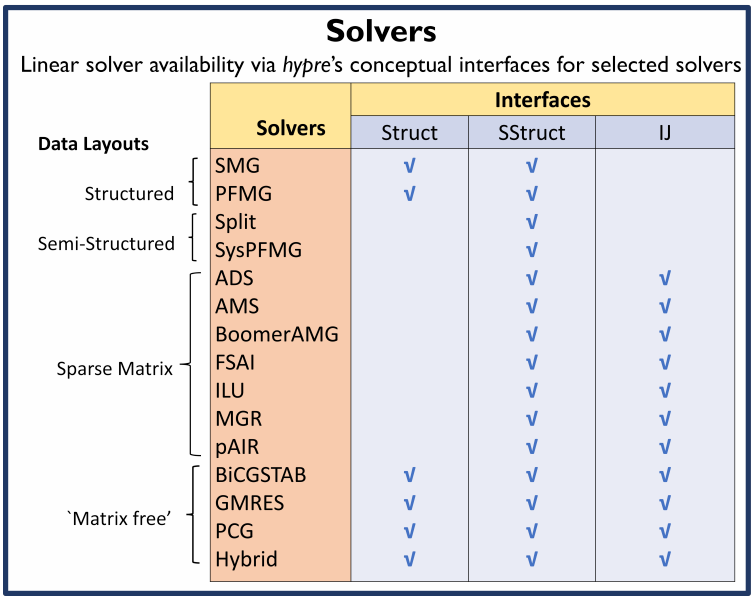

Fig. 2: Selected *hypre* solvers and their association with interfaces and data layouts.

Figure 2 shows a list of available solvers in *hypre* and indicates the interfaces that can be accessed along with the data layouts used for their implementation. The term 'matrix free' means the solver can be run with any of the three data layouts, whereas all other solvers have been implemented using the data layout mentioned in Figure 2. Most of *hypre*'s solvers are multigrid methods. Exceptions are the Krylov solvers, the incomplete LU factorization preconditioner ILU, and the sparse approximate inverse preconditioner FSAI. These methods have been included for difficult systems that cannot be solved with multigrid methods and can also be used as smoothers within *hypre*'s unstructured AMG solver BoomerAMG for more robust convergence if needed. All methods in Figure 2 have been GPU-enabled for NVIDIA, AMD, and Intel GPUs via CUDA, HIP, and SYCL. However, not all options and configurations of these solvers are fully ported to GPUs. SMG, PFMG [3], SysPFMG, pAIR [23, 15], and BoomerAMG [16] can be described by the general AMG method described in Section 2.1, whereas ADS, AMS and MGR are more specialized solvers that internally use BoomerAMG. AMS [19] has been developed for systems derived from H-curl discretizations of the Maxwell equations, ADS [20] for H-div systems, and MGR [27], described in Section 2.2, for block linear systems. The solvers used in the applications considered in Section 4 are PFMG, BoomerAMG, and MGR, which are described in more detail below.

2.1 Algebraic multigrid

AMG methods consist of a setup phase and a solve phase. The setup phase consists of the coarsening algorithm that determines the variables to be used on the next coarser level, the generation of restriction and prolongation operators, $R^{(m)}$ and $P^{(m)}$, needed to move between different levels, m and $m+1$, and the generation of the coarse grid operator, which is generally accomplished by the Galerkin product, $A^{(m+1)} = R^{(m)} A^{(m)} P^{(m)}$, where $A^{(0)} = A$ is the original linear system to be solved. Often restriction is defined as the transpose of interpolation, $R^{(m)} = (P^{(m)})^T$, particularly when dealing with a symmetric matrix, A. Figure 3 shows two levels of an AMG V-cycle, beginning with a few steps of smoothing followed by a restriction of the residual to the next level. This process is then applied recursively until the coarsest level, where the system can be solved directly. The coarse-grid error correction, e^{m+1}, is then interpolated to the next finer level and added to the approximation, u^m, before performing some additional smoothing.

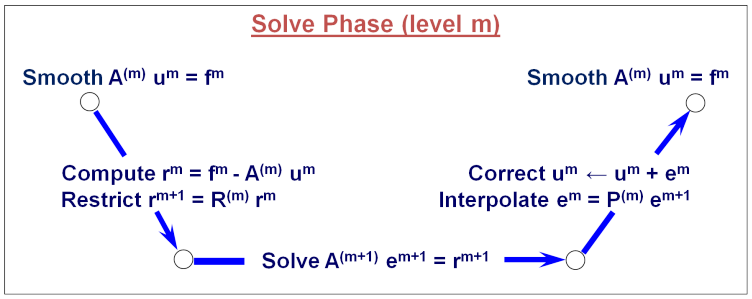

Fig. 3: Two levels of an AMG V-cycle.

PFMG, *hypre*'s most efficient solver, is based on the structured data layout, which is defined by grids and stencils and is well suited for GPUs. It uses semi-coarsening and a simple two-point interpolation operator. This strategy is highly efficient, since it limits stencil sizes on lower levels. The solve phase uses an efficient point smoother.

BoomerAMG, *hypre*'s most widely used preconditioner, is based on a parallel compressed sparse row (CSR) matrix data layout. It has a large variety of different options. Default settings for GPUs include PMIS coarsening [11] and interpolation operators based on matrix operations [21]. Smoothing is generally done with simple methods, such as L1-Jacobi [4], but more complex smoothers such as ILU or FSAI can also be employed.

2.2 Multigrid Reduction Framework

Multigrid reduction (MGR) is an approximation of the total reduction method to yield an efficient iterative solver. Originally introduced by Ries, Trottenberg and Winter in 1983 [24], various adaptations of the algorithm have been developed, including [22, 28, 2, 13, 27]. The MGR framework developed in *hypre* is designed to enable the efficient solution of systems of coupled equations, which are known to be problematic for scalar multigrid methods. The approach generalizes multi-stage and block preconditioners, typically used in multiphysics applications, in a standard multigrid framework [27]. Following standard multigrid convention, consider a splitting of a coupled system of discretized partial differential equations (PDEs) into two sets of C-points and F-points:

$$A = \begin{pmatrix} A_{FF} & A_{FC} \\ A_{CF} & A_{CC} \end{pmatrix} = \begin{pmatrix} I_{FF} & 0 \\ A_{CF}A_{FF}^{-1} & I_{CC} \end{pmatrix} \begin{pmatrix} A_{FF} & 0 \\ 0 & S_* \end{pmatrix} \begin{pmatrix} I_{FF} & A_{FF}^{-1}A_{FC} \\ 0 & I_{CC} \end{pmatrix}, \tag{1}$$

where I_{CC} and I_{FF} are identity operators and $S_* = A_{CC} - A_{CF}A_{FF}^{-1}A_{FC}$ is the Schur complement. The idea is to construct approximate restriction and interpolation operators, R and P, so that $RAP \approx S_*$. The resulting two-grid error propagation operator for the MGR solution can be expressed as

$$I - M_{MGR}^{-1}A = (I - PM_C^{-1}RA)(I - M_F^{-1}A), \tag{2}$$

which describes the complementary processes of F-relaxation, with smoother M_F^{-1}, followed by coarse grid correction, with $M_C^{-1} \approx (RAP)^{-1}$. For a detailed description and theory of the two-grid approach, as well as equivalences to multi-stage and block preconditioners used in the multiphysics community, see [27].

Algorithm 1 Multigrid Reduction Solve Phase.

1: $MGR(l)$
2: **if** l is coarsest level, L **then**
3: solve coarse grid system $A_L e_L = r_L$ by classical AMG
4: **else**
5: Let $r = b$ and $e_0 = 0$
6: **Global Relaxation:** $e_l \leftarrow e_l + M_l^{-1}r$, where $M_l \approx A_l$
7: **F-Relaxation:** $e_l \leftarrow e_l + M_{F_l}^{-1}(r_l - A_l e_l)$
8: **Restriction:** $r_{l+1} = R_l(r_l - A_l e_l)$
9: **Recursion:** $e_{l+1} = MGR(l+1)$
10: **Update solution:** $e_l \leftarrow e_l + P_l e_{l+1}$
11: **end if**

Algorithm 1 describes the multigrid reduction's solve phase, which extends the two-level method in (2) via recursion. The subscript l denotes the operator at level l of

the multilevel hierarchy. The algorithm presents a general approach, which includes pre-smoothing by a global smoother at each level. The global smoother may be used instead of F-relaxation (as a more potent smoother) or as an additional smoothing step. In general, block relaxation schemes such as Jacobi or hybrid Gauss-Seidel are sufficient as a global smoother. However, ILU smoothers are more effective for problems with complicated coupling of unknowns. The framework provides a flexible interface to prescribe the splitting of unknowns into C-points and F-points for various block linear systems. Assuming this splitting is done based on the different physical variables in the system, a typical approach reduces a physical variable at each level until a scalar problem is reached at the coarsest level. In practice, however, variables with similar physical properties can be prescribed together at the same level and treated as a scalar problem. The implementation admits various compositions of smoothers to handle different physical regimes across the levels. Typically, using cheaper yet more efficient smoothers for F-relaxation and a more robust solver for the coarse grid correction leads to a reasonable preconditioner. The framework supports standard smoothers such as Jacobi, Chebyshev, and ILU; multilevel options like AMG; and direct solvers at each level. At the coarsest level, the coarse grid correction uses classical AMG, which is important to maintain good scalability for large-scale simulations. Thus, it is favorable to have a reduction strategy that has good M-matrix properties on the coarsest grid. At each level in the hierarchy, the coarse grid matrix can be constructed via a Galerkin product (RAP) or a non-Galerkin approach, which provides additional flexibility in approximating the Schur Complement. The MGR framework offers GPU support for various solver options and linear algebra operations. It has been successfully applied to large-scale applications of subsurface flow with geomechanical processes [27, 8, 7]. Numerical examples in Section 4.2.2 will demonstrate the performance of MGR on a multiphase subsurface flow problem with thermal effects and poromechanics.

3 Porting strategies to exascale computers

Adapting *hypre* for GPU-based exascale machines necessitates the migration of operations within the algorithms from CPUs to GPUs, provided they remain suitable for the fine-grained parallelism inherent to GPUs. Otherwise, entirely new algorithms may need to be developed specifically for GPUs. For example, porting the original extended and extended+i interpolation algorithms [10] directly to GPUs poses various challenges. Consequently, new versions of the algorithms have been devised, formulated as sparse matrix-matrix multiplications (SpGEMM) [21]. This approach not only leverages existing high-performance SpGEMM kernels but also maintains the interpolation properties in the original forms.

Our porting strategies to develop high-performance GPU-accelerated AMG code adopt the following 3 approaches in sequence:

1. Utilize existing dense or sparse kernels from vendor libraries: cuBLAS, cuSPARSE, cuSOLVER (NVIDIA); rocBLAS, rocSPARSE, rocSOLVER (AMD); oneAPI, oneMKL (Intel).
2. Leverage high-level libraries that provide parallel primitives on GPUs of the algorithms in the C++ Standard library: thrust (CUDA); rocprim (HIP); oneDPL (SYCL). These primitives often times serve as building blocks of complicated AMG algorithms.
3. As necessary, develop custom device functions.

This sequence balances the extent of new code development while achieving optimal performance portability across GPUs and multicore CPUs. More details on the porting strategies of *hypre* to GPUs can be found in [14].

An example of the initial approach pertains to a critical operation, specifically sparse matrix-vector product (SpMV), which is essential in the AMG solve phase. Notably, sparse linear algebra libraries offered by vendors all include highly efficient algorithms for matrices in the CSR format.

As an illustrative example of the second approach, consider a fundamental operation used at many places in constructing AMG: transforming the row pointer array of the CSR matrix into the array of row indices of all nonzeros of a matrix. The following implementation, utilizing a double for-loop, can efficiently execute on CPUs, where `nrows` is the number of rows, `RowPtr` is the row pointer array of CSR, and `RowIdx` is the output array of row indices of all the nonzeros. To further improve performance on multi-core architectures, multi-threaded parallelism can be implemented for the outer for-loop.

```
#pragma omp parallel for
for (i = 0; i < nrows; i++)
  for (j = RowPtr[i]; j < RowPtr[i+1]; j++) { RowIdx[j] = i; }
```

Alternatively, for a GPU implementation aimed at maximizing parallelism, particularly in this case with respect to the number of nonzero coefficients, an efficient approach involves leveraging three parallel primitives: `fill`, `scatter_if`, and `inclusive_scan`. The first two primitives are in the C++ Standard Library, while the last one is available in the Boost C++ Libraries [25], and all of them are in the C++ parallel algorithms library Thrust [1]. Here, `empty_row` is a straightforward function that returns true if a row contains no nonzeros.

```
fill(RowIdx, RowIdx + nnz, 0);
scatter_if(counting_iterator<int>(0), counting_iterator<int>(nrows), RowPtr,
transform_iterator(zip_iterator(make_tuple(RowPtr, RowPtr+1)), empty_row()),
  RowIdx);
inclusive_scan(RowIdx, RowIdx + nnz, RowIdx, maximum<int>());
```

Compared to the CPU implementation, this approach offers several advantages when running on GPUs: 1) enhanced granularity of parallelism, particularly on a per-nonzero basis within the first and last primitives; 2) improved memory coalescing patterns from adjacent threads; 3) better workload balancing, especially when the distribution of nonzeros among rows is far from an even distribution. Seamless unification of primitive function calls across different languages (i.e., CUDA, HIP

and SYCL) can be achieved effortlessly through straightforward macro definitions within distinct namespaces.

The final approach of writing custom device functions is commonly employed when developing complicated AMG algorithms that are challenging to organize into standard linear algebra operations or even into standard parallel primitives. Implementations of these algorithms often diverge significantly from parallel CPU implementations for multi-core CPUs. Consequently, porting these algorithms presents nontrivial challenges, stemming from factors like thread granularity and the partitioning of computations among threads. We carefully implemented these device functions by exploiting low level hardware and language support, e.g., on-chip shared memory, warp-level primitives, thread synchronizations. Unifying device function implementations for CUDA and HIP is straightforward due to the similarity and compatibility of these two languages, requiring only the resolution of minor discrepancies such as the size of warps and maximum shared memory size. The syntax for launching device kernel functions remains identical across both languages. On the other hand, transitioning to SYCL presents remarkable challenges. While the mapping of functions between thrust and rocprim is straightforward, manual translation is often required for oneDPL. Additionally, while OneMKL provides sparse BLAS routines, caution must be exercised to navigate potential pitfalls, such as the matrix sorting requirement in the SpGEMM routine. Moreover, differences in kernel specification and launch syntax add further complexity to the process. To facilitate code reuse and efficiency, we defined macros as follows.

```
sycl::range<1> : gridsize, blocksize
void hypreGPUKernel_* (sycl::nd_item &item, ...) {
    get_grid_thread_id<1, 1>(item) }
#define HYPRE_GPU_LAUNCH {sycl::queue->submit([&](sycl::handler& cgh) {
cgh.parallel_for (sycl::nd_range<1>(gridsize*blocksize, blocksize),[=](sycl::
  nd_item<1> item)[[intel:: reqd_sub_group_size(HYPRE_WARP_SIZE)]] {(
  kernel_name)(item, __VA_ARGS__);});}).wait_and_throw();}
```

Finally, basic infrastructure elements, such as SYCL devices and warp/group-level functions, differ and thus necessitate re-implementation.

4 Use of *hypre* in applications

Hypre's recent advances, especially its GPU capabilities, have significantly enhanced the ability of application codes to leverage modern supercomputers. In this section, we demonstrate the use of *hypre* in three application codes, emphasizing its critical role in achieving scalability and reducing overall execution times.

4.1 Uintah[1]

The Uintah framework is utilized for simulating boilers and conducting combustion research at the University of Utah. The code employs *hypre*'s PFMG to solve pressure equations. Figure 4 shows a scaling study of the ARCHES turbulent combustion component that simulates a helium plume on a single-level structured grid, which is a cube divided into patches. The experiment was performed for three problem sizes using two different patch sizes on the Summit computer at ORNL and used both GPUs and CPUs for the Uintah components of the code through Kokkos enabling OpenMP threading for the CPU cores and CUDA for the GPUs. The *hypre* library was exclusively run on GPUs using CUDA, with PFMG and micro-kernel fusion for enhanced performance [17]. The results, shown in Figure 4, demonstrate excellent weak and strong scaling. Larger patch sizes yield better performance, while smaller patch sizes allow for a higher degree of fine-grained parallelism. Approximately 50 % of the total simulation time is attributed to *hypre*, indicating its significant contribution to the achieved scalability. Additionally, the Uintah team has successfully run their framework with *hypre* on two exascale computers: Frontier at ORNL (utilizing HIP for *hypre*) and Aurora at Argonne National Laboratory (utilizing SYCL for *hypre*).

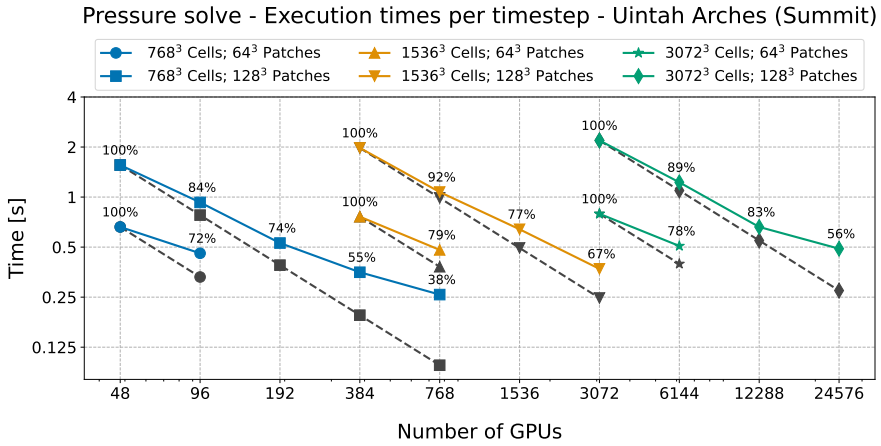

Fig. 4: Helium plume run to 24,576 NVIDIA V100 GPUs and 8,192 IBM POWER9 processors with 172,032 CPU threads using MPI+Kokkos for Uintah components and CUDA for PFMG.

[1] http://uintah.utah.edu

4.2 GEOS[2]

GEOS is an open-source and GPU-enabled simulation code written in C++ specifically designed to solve tightly-coupled multiphysics problems, with a primary focus on subsurface reservoir applications. It offers robust capabilities for modeling compositional flow, poromechanics, faults and fractures slip, and thermal effects in underground reservoirs, setting it apart from traditional simulators. The mathematical models in GEOS typically result in challenging linear systems of equations. These systems are primarily solved using *hypre*, employing its MGR preconditioner, or the BoomerAMG preconditioner for simpler single-phase flow scenarios. In this section, we present weak and strong scalability results for two simulations performed with GEOS, highlighting *hypre*'s performance and efficiency.

4.2.1 Weak scaling - Single-phase flow model

We used GEOS to simulate fluid flow in the vicinity of a wellbore, a scenario that is important for optimizing extraction processes and enhancing reservoir management. The model considers a single fluid type and solves for the pressure distribution in a radially structured mesh. The resulting linear systems generated in the transient simulation are solved with *hypre* 's GMRES preconditioned with BoomerAMG.

Figure 5 presents a weak scalability study in the Frontier supercomputer. We show average execution times per time step of the simulation and distinguish between *hypre* and GEOS times. The x-axis represents the number of GPUs[3] used followed by total number of degrees of freedom (DOFs) in millions (M) and billions (B). As the number of GPUs increases from 4 to 16384, corresponding to an increase in DOFs from 6.6 million to 27 billion, the execution times for both the overall GEOS simulation and the *hypre* solver components exhibit moderate increases. Specifically, the total GEOS execution time rises from 0.06 seconds to 0.08 seconds, demonstrating the framework's efficiency and scalability in handling larger problem sizes. *hypre*'s (BoomerAMG) setup time increases from 0.12 seconds to 0.47 seconds, reflecting the additional distributed communication overhead required to build the preconditioner with larger compute node counts. Lastly, *hypre*'s (BoomerAMG-GMRES) solve time shows a gradual increase from 0.18 seconds to 0.50 seconds, maintaining a relatively consistent performance despite the growing complexity. We highlight that the largest scale run used a little less than one-forth of the whole machine.

[2] https://www.geos.dev/

[3] In the context of the Frontier system, the term "GPU" refers to a Graphics Compute Die (GCD) and not a physical GPU card. Each AMD MI250X GPU contains 2 GCDs, and with 4 MI250X cards per node, this results in 8 GCDs per node, each treated as a separate GPU in system configurations and programming.

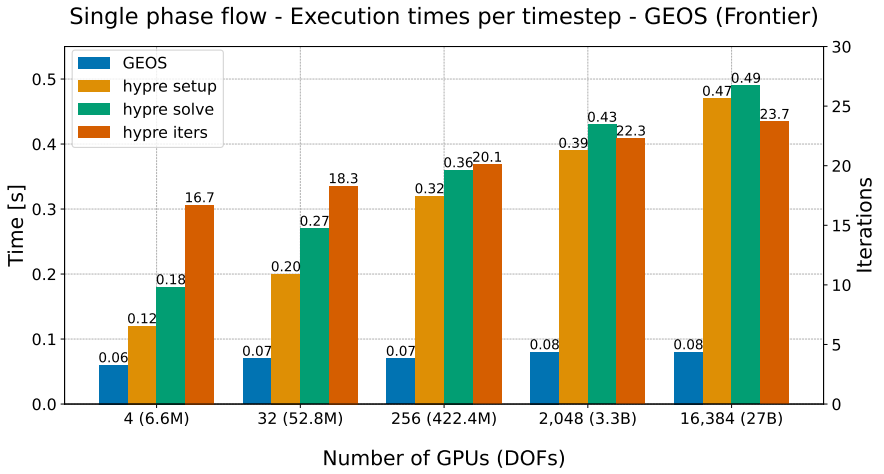

Fig. 5: Average execution time (left y-axis) and iteration count (right y-axis) per time step for a single-phase flow simulation in GEOS. As the number of GPUs and DOFs increase by a factor of 4096, both *hypre* setup and solve times increase slightly by a factor less than 4, demonstrating the excellent weak scalability of *hypre*'s algebraic multigrid solver (BoomerAMG).

4.2.2 Strong scaling - Thermo-Hydro-Mechanical (THM) model

This test problem uses *hypre*'s MGR as a preconditioner to GMRES to solve a real-world simulation of underground carbon storage and sequestration (CSS) [9, Section 4.4] including thermal effects. The primary variable types of this multi-physics problem are seven: three displacement degrees of freedom, two component densities (water and gas), one pressure, and one temperature. The MGR strategy consists of four levels. In the first level, the displacement degrees of freedom are eliminated, followed by the water component density, and then gas component density. At the last level, pressure and temperature are treated together since, in this test problem, they share elliptic properties that are amenable to AMG. To address the pressure-temperature coarse grid problem, we apply a single V(1,1)-cycle of an unknown-based BoomerAMG [5]. This approach improves setup and solution times by creating an algebraic multigrid hierarchy with interpolation operators that do not take into account inter-variable couplings, in this case, pressure and temperature degrees of freedom. For more information on the isothermal version (without temperature) of this MGR preconditioning technique, see [9, Section 3].

Figure 6 shows strong scaling results obtained using Ruby, an HPC cluster at LLNL equipped with 2x 28-core Intel® Xeon® Platinum 8276 CPUs. The results indicate that as the number of MPI ranks increases, there is a significant reduction in the total execution time for the application and *hypre* times (setup and solve), demonstrating efficient parallel scalability. Notably, the execution time decreases al-

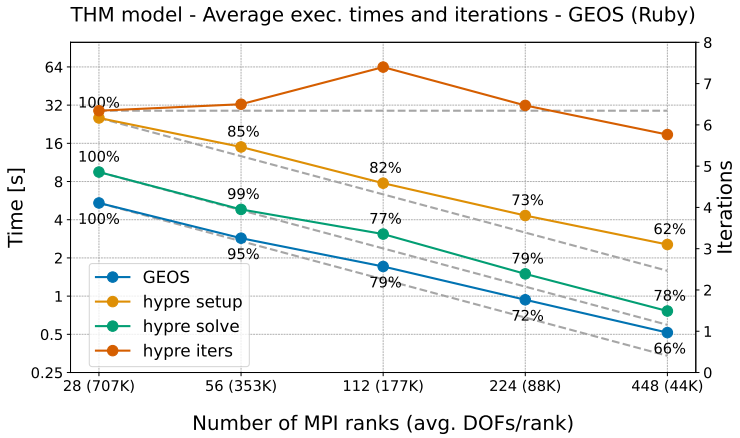

Fig. 6: Strong scaling results for a THM simulation in GEOS, showing average execution times (left y-axis) and iterations (right y-axis) for different numbers of MPI ranks on Ruby. Both *hypre* phases and GEOS exhibit parallel efficiencies of up to nearly 70% after a 16-fold increase in computational resources.

most linearly with the increase in MPI ranks, maintaining high efficiency percentages across different configurations. For instance, the *hypre* setup time drops from approximately 32 seconds with 28 ranks to around 2 seconds with 448 ranks, reflecting a 62% efficiency at the highest rank count. Similarly, the *hypre* solve and GEOS times exhibit substantial reductions, achieving around 66% and 78% parallel efficiency, respectively, at the maximum rank count (448). The average number of iterations required for convergence remains relatively stable, with a slight decrease, highlighting the robustness of the MGR preconditioner. These results demonstrate *hypre*'s scalability and effectiveness in leveraging high-performance computing resources for large-scale multiphysics simulations.

4.3 Chombo[4]

Chombo is a software framework designed for CFD simulations on block-structured and adaptively refined grids. Chombo is capable of handling complex geometries and dynamic mesh refinement, making it suitable for high-resolution simulations at various Reynolds numbers capturing both laminar and turbulent flow regimes. BoomerAMG is used by Chombo through PETSc [6] as a preconditioner to GMRES in order to solve its pressure projection equation.

[4] https://commons.lbl.gov/display/chombo

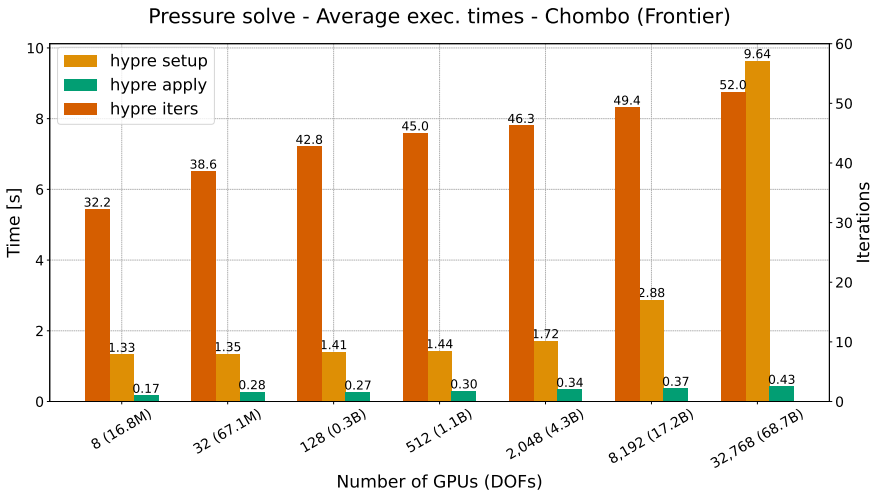

Fig. 7: Average execution time (left y-axis) and iteration count (right y-axis) per linear system for the pressure projection solve in Chombo. As the problem size scales up, the execution times for both setup and solve phases show moderate increase, highlighting *hypre*'s performance and scalability in handling large-scale elliptic systems arising from computational fluid dynamics simulations.

Figure 7 illustrates the weak scaling results obtained on Frontier (ORNL) for simulating flow inside a cylinder packed with spheres [26]. The chart shows BoomerAMG's construction time (*hypre* setup), which is performed only once at the start of the simulation; average application time (*hypre* apply); and average iteration counts (*hypre* iters) per linear system across different numbers of GPUs, ranging from 8 to 32,768 (almost half of the entire machine). As the number of GPUs increases, the preconditioner application times and iterations scales very well, increasing by only a factor of 2.5 and 1.6, respectively, despite a 4,096-fold increase in the number of GPUs. BoomerAMG setup times shows excellent scaling up to 2,048 GPUs, although performance degrades beyond this point, which requires further investigation.

5 Conclusions

We analyzed the impact of three different *hypre* solvers on three HPC systems: Ruby at LLNL, Summit, and Frontier at ORNL. For the applications presented here, solving linear systems consistently took up more than half of the total simulation time, emphasizing the crucial role of efficient solvers for sparse linear systems in HPC simulation codes. Additionally, *hypre* achieves good strong and weak scaling for these applications, effectively leveraging tens of thousands of GPUs.

As we enter the era of exascale computing, optimizing solver performance becomes increasingly important. The insights gained from these applications and many others will inform future developments in *hypre* to reduce computational costs and enhance scalability at large node counts. Continued efforts will focus on refining *hypre* solvers to fully utilize the potential of exascale systems, particularly for the El Capitan system at LLNL, which is based on AMD accelerated processing units. This will ultimately enable more sophisticated and accurate scientific simulations at faster execution times and finer resolutions.

Acknowledgements This work was performed in part under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. This material is based upon work supported by the Exascale Computing Project, a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. Support for this work was provided in part by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) Program through the Frameworks, Algorithms, and Scalable Technologies for Mathematics (FASTMath) Institute.

Competing Interests The authors have no conflicts of interest to declare that are relevant to the content of this chapter.

References

1. Thrust: The C++ parallel algorithms library. https://nvidia.github.io/cccl/thrust/
2. Ali, A., Brannick, J.J., Kahl, K., Krzysik, O.A., Schroder, J.B., Southworth, B.S.: Constrained local approximate ideal restriction for advection-diffusion problems. SIAM Journal on Scientific Computing **0**(0), S96–S122 (0). DOI 10.1137/23M1583442. URL https://doi.org/10.1137/23M1583442
3. Ashby, S., Falgout, R.: A parallel multigrid preconditioned conjugate gradient algorithm for groundwater flow simulations. Nuclear Science and Engineering **124**, 145–159 (1996). DOI 10.13182/NSE96-A24230
4. Baker, A., Falgout, R., Kolev, T., Yang, U.M.: Multigrid smoothers for ultraparallel computing. SIAM Journal on Scientific Computing **33**, 2864–2887 (2011). DOI 10.1137/100798806
5. Baker, A.H., Kolev, T.V., Yang, U.M.: Improving algebraic multigrid interpolation operators for linear elasticity problems. Numerical Linear Algebra with Applications **17**(2-3), 495–517 (2010). DOI 10.1002/nla.688. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nla.688
6. Balay, S., Abhyankar, S., Adams, M.F., Benson, S., Brown, J., Brune, P., Buschelman, K., Constantinescu, E.M., Dalcin, L., Dener, A., Eijkhout, V., Faibussowitsch, J., Gropp, W.D., Hapla, V., Isaac, T., Jolivet, P., Karpeev, D., Kaushik, D., Knepley, M.G., Kong, F., Kruger, S., May, D.A., McInnes, L.C., Mills, R.T., Mitchell, L., Munson, T., Roman, J.E., Rupp, K., Sanan, P., Sarich, J., Smith, B.F., Zampini, S., Zhang, H., Zhang, H., Zhang, J.: PETSc Web page. https://petsc.org/ (2024). URL https://petsc.org/
7. Bui, Q., Hamon, F., Castelletto, N., Osei-Kuffuor, D., Settgast, R., White, J.: Multigrid reduction preconditioning framework for coupled processes in porous and fractured media. Computer Methods in Applied Mechanics and Engineering **387**, 114111 (2021). DOI 10.1016/j.cma.2021.114111
8. Bui, Q.M., Osei-Kuffuor, D., Castelletto, N., White, J.A.: A scalable multigrid reduction framework for multiphase poromechanics of heterogeneous media. SIAM Journal

on Scientific Computing **42**(2), B379–B396 (2020). DOI 10.1137/19M1256117. URL https://doi.org/10.1137/19M1256117
9. Cremon, M.A., Franc, J., Hamon, F.P.: Constrained pressure-temperature residual (CPTR) preconditioner performance for large-scale thermal CO_2 injection simulation. Computational Geosciences (2024). DOI 10.1007/s10596-024-10292-z. URL https://doi.org/10.1007/s10596-024-10292-z
10. De Sterck, H., Falgout, R.D., Nolting, J.W., Yang, U.M.: Distance-two interpolation for parallel algebraic multigrid. Numerical Linear Algebra with Applications **15**(2-3), 115–139 (2008). DOI https://doi.org/10.1002/nla.559
11. De Sterck, H., Yang, U.M., Heys, J.J.: Reducing complexity in parallel algebraic multigrid preconditioners. SIAM Journal on Matrix Analysis and Applications **27**(4), 1019–1039 (2006). DOI 10.1137/040615729
12. Falgout, R., Jones, J., Yang, U.: The design and implementation of hypre, a library of parallel high performance preconditioners. In: A. Bruaset, A. Tveito (eds.) Numerical Solution of Partial Differential Equations on Parallel Computers, pp. 267–294. Springer (2006)
13. Falgout, R.D., Friedhoff, S., Kolev, T.V., MacLachlan, S.P., Schroder, J.B.: Parallel time integration with multigrid. SIAM Journal on Scientific Computing **36**(6), C635–C661 (2014). DOI 10.1137/130944230
14. Falgout, R.D., Li, R., Sjögreen, B., Wang, L., Yang, U.M.: Porting hypre to heterogeneous computer architectures: Strategies and experiences. Parallel Computing **108**, 102840 (2021). DOI https://doi.org/10.1016/j.parco.2021.102840. URL https://www.sciencedirect.com/science/article/pii/S0167819121000867
15. Hanophy, J., Southworth, B.S., Ruipeng Li, T.M., Morel, J.: Parallel approximate ideal restriction multigrid for solving the SN transport equations. Nuclear Science and Engineering **194**(11), 989–1008 (2020). DOI 10.1080/00295639.2020.1747263. URL https://doi.org/10.1080/00295639.2020.1747263
16. Henson, V.E., Yang, U.M.: BoomerAMG: A parallel algebraic multigrid solver and preconditioner. Applied Numerical Mathematics **41**(1), 155 – 177 (2002). DOI 10.1016/S0168-9274(01)00115-5
17. Holmen, J.K., Sahasrabudhe, D., Berzins, M.: Porting uintah to heterogeneous systems. In: PASC22: Proceedings of the Platform for Advanced Scientific Computing Conference, pp. 1–10 (2022). DOI 10.1145/3539781.3539794
18. *hypre*: High performance preconditioners. https://llnl.gov/casc/hypre, https://github.com/hypre-space/hypre
19. Kolev, T.V., Vassilevski, P.S.: Parallel auxiliary space AMG for $H(curl)$ problems. Journal of Computational Mathematics **27**, 604–623 (2009). DOI 10.4208/jcm.2009.27.5.013
20. Kolev, T.V., Vassilevski, P.S.: Parallel auxiliary space AMG solver for $H(div)$ problems. SIAM Journal on Scientific Computing **34**(6), A3079–A3098 (2012). DOI 10.1137/110859361
21. Li, R., Sjögreen, B., Yang, U.M.: A new class of AMG interpolation methods based on matrix-matrix multiplications. SIAM Journal on Scientific Computing **43**(5), S540–S564 (2021). DOI 10.1137/20M134931X. URL https://doi.org/10.1137/20M134931X
22. MacLachlan, S., Manteuffel, T., McCormick, S.: Adaptive reduction-based AMG. Numerical Linear Algebra with Applications **13**(8), 599–620 (2006). DOI https://doi.org/10.1002/nla.486. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nla.486
23. Manteuffel, T.A., Ruge, J., Southworth, B.S.: Nonsymmetric algebraic multigrid based on local approximate ideal restriction (ℓAIR). SIAM Journal on Scientific Computing **40**(6), A4105–A4130 (2018). DOI 10.1137/17M1144350. URL https://doi.org/10.1137/17M1144350
24. Ries, M., Trottenberg, U., Winter, G.: A note on MGR methods. Linear Algebra and its Applications **49**, 1 – 26 (1983). DOI 10.1016/0024-3795(83)90091-5
25. Schäling, B.: The Boost C++ libraries. XML Press (2014)
26. Trebotich, D.: Exascale Computational Fluid Dynamics in Heterogeneous Systems. Journal of Fluids Engineering **146**(4), 041104 (2024). DOI 10.1115/1.4064534. URL https://doi.org/10.1115/1.4064534

27. Wang, L., Osei-Kuffuor, D., Falgout, R.D., Mishev, I.D., Li, J.: Multigrid reduction for coupled flow problems with application to reservoir simulation. In: SPE Reservoir Simulation Conference. Society of Petroleum Engineers, SPE-182723-MS (2017). DOI 10.2118/182723-MS
28. Zaman, T., Nytko, N., Taghibakhshi, A., MacLachlan, S., Olson, L., West, M.: Generalizing reduction-based algebraic multigrid. Numerical Linear Algebra with Applications **31**(3), e2543 (2024). DOI https://doi.org/10.1002/nla.2543. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nla.2543