LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-TR-533076

Preparing Algebraic Multigrid for Exascale

A. H. Baker, R. D. Falgout, H. Gahvari, T. Gamblin, W. Gropp, T. V. Kolev, K. E. Jordan, M. Schulz, U. M. Yang

February 27, 2012

Disclaimer

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

PREPARING ALGEBRAIC MULTIGRID FOR EXASCALE

ALLISON H. BAKER[†¶], ROBERT D. FALGOUT[†], HORMOZD GAHVARI[‡†], TODD GAMBLIN[†], WILLIAM GROPP[‡], KIRK E. JORDAN[§], TZANIO V. KOLEV[†], MARTIN SCHULZ[†], AND ULRIKE MEIER YANG[†]

Abstract. Algebraic Multigrid (AMG) solvers are an essential component of many large-scale scientific simulation codes. Their continued numerical scalability and efficient implementation is critical for preparing these codes for exascale. Our experiences on modern multi-core machines show that significant challenges must be addressed for AMG to perform well on such machines. We discuss our experiences and describe the techniques we have used to overcome scalability challenges for AMG on hybrid architectures in preparation for exascale.

1. Introduction. Sparse iterative linear solvers are critical for large-scale scientific simulations, many of which spend the majority of their run time in solvers. Algebraic Multigrid (AMG) is a popular solver because of its linear run-time complexity and its proven scalability in distributed-memory environments. However, changing supercomputer architectures present challenges to AMG's continued scalability.

Multi-core processors are now standard on commodity clusters and high-end supercomputers alike, and core counts are increasing rapidly. However, distributed-memory message passing implementations, such as MPI, are not expected to work efficiently with more than hundreds of thousands of tasks. With exascale machines expected to have hundreds of millions or billions of tasks and hundreds of tasks per node, programming models will necessarily be hierarchical, with local shared-memory nodes in a larger distributed-memory message-passing environment.

With exascale in mind, we have begun to focus on the performance of BoomerAMG [14], the AMG solver in the *hypre* [15] library, on multicore architectures. BoomerAMG has demonstrated good weak scalability in distributed-memory environments, such as on 125,000 processors of BG/L [8], or BG/P [5], but our preliminary study [7] has shown that non-uniform memory access (NUMA) latency between sockets, deep cache hierarchies, multiple memory controllers, and reduced on-node bandwidth can be detrimental to AMG's performance.

To achieve high performance on exascale machines, we will need to ensure numerical scalability and an efficient implementation as core counts increase, memory capacity per core decreases, and on-node cache architectures become more complex. Some components of AMG that lead to very good convergence do not parallelize well or depend on the number of processors. We examine the effect of high level parallelism involving large numbers of cores on one of AMG's most important components, smoothers, in Section 3. We also develop a performance model of the AMG solve cycle to better understand AMG's performance bottlenecks (Section 4), and use it to evaluate new AMG variants (Section 5). Since our investigations show that the increasing communication complexity on coarser grids combined with the effects of increasing numbers of cores lead to severe performance bottlenecks for AMG on various multicore architectures, we investigate two different approaches to reduce communication in AMG: an AMG variant, which we denote as the "redundant coarse grid approach" (Section 5), and the use of a hybrid MPI/OpenMP programming model (Section 6). We present results for AMG using MPI/OpenMP on three supercomputers with different node architectures: a cluster with four quad-core AMD Opteron processors, a Cray XT5 machine with two hex-core AMD Opteron processors, and a BlueGene/P system with a single quad-core PowerPC processor per node. Finally, Section 7 contains the conclusions.

2. The Algebraic Multigrid Solver. Multigrid (MG) linear solvers are particularly well-suited to parallel computing because their computational cost is linearly dependent on the problem size. This optimal property, also referred to as algorithmic scalability, means that proportionally increasing both the problem size and the number of processors (i.e., weak scaling), results in a roughly constant number of iterations to solution. Therefore, unsurprisingly, multigrid methods are currently quite popular for large-scale scientific computing and will play a critical role in enabling simulation codes to perform well at exascale.

[†]Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA 94551
[‡]Computer Science Department, University of Illinois at Urbana-Champaign, Urbana, IL 61801
[§]IBM TJ Watson Research Center, Cambridge, MA 02142
[¶]Now at Computational and Information Systems Laboratory, National Center for Atmospheric Research, Boulder, CO 80305.

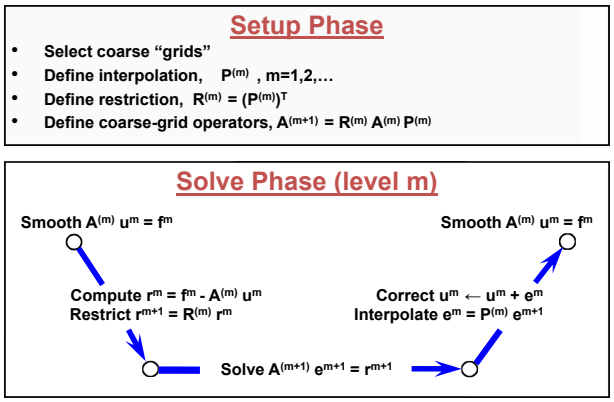

FIG. 2.1. *AMG building blocks.*

An MG method's low computational cost results from restricting the original linear system to increasingly coarser grids, which require fewer operations than the fine grid. An approximate solution is determined on the coarsest grid, typically with a direct solver, and is then interpolated back up to the finest grid. On each grid level an inexpensive smoother (e.g., a simple iterative method like Gauss-Seidel) is applied. The process of starting on the fine grid, restricting to the coarse grid, and interpolating back to fine grid again is called a "V-cycle", which corresponds to a single MG iteration.

MG has two phases: *setup* and *solve*, see Figure 2.1. The primary computational kernels in the setup phase are the selection of the coarse grids, creation of the interpolation operators, and the representation of the fine grid matrix operator on each coarse grid. The primary computational kernels in the solve phase are a matrix-vector multiply (MatVec) and the smoothing operator, which may closely resemble a MatVec.

AMG is a flexible and unique type of MG method because it does not require geometric grid information. In AMG, coarse "grids" are simply subsets of the fine grid variables, and the coarsening and interpolation algorithms make use of the matrix entries to select variables and determine weights. These algorithms can be quite complex, particularly in parallel. More detailed information on AMG may be found in [8] or [17].

A well-designed AMG method is algorithmically scalable in that the number of iterations should stay fixed with increasing problem size. However, an effective AMG code must also be computationally scalable: the run times should stay constant with weak scaling. Therefore, both the algorithmic details related to the underlying mathematics (which impact the convergence rate) and the implementation details of the algorithm are important. To prepare our code for exascale computing, we have begun to examine the primary components of AMG, beginning with the solve phase, to determine what issues will need to be addressed.

3. Smoothers. The smoothing process is at the heart of the AMG algorithm, and the quality of the smoother directly affects the design and the scalability of the multigrid solver. For a linear system with a symmetric and positive definite (SPD) matrix A, a smoother is another matrix M such that the iteration

$$e^0 = e\,, \qquad e^{n+1} = (I - M^{-1}A)e^n \quad \text{for } n = 1, 2, \ldots$$

reduces the high-frequency components of an initial error vector e. This makes the remaining error smooth, so it can be handled by the coarse grid corrections. The smoother should also be convergent, so that the above iteration does not increase the low-frequency components of the error. When M is not symmetric, one can consider the symmetrized smoother $\widetilde{M} = M^T(M^T + M - A)^{-1}M$, which corresponds to a smoothing iteration with M, followed by a pass with M^T. The symmetrized smoother is often used when preconditioning conjugate gradient (CG) with AMG, since CG requires a symmetric preconditioner.

One classical example of a convergent smoother is the Gauss-Seidel (GS) method, which is obtained by setting $M_{GS} = L + D$, where L is the lower triangular part and D is the diagonal part of A. Note that M_{GS} is not symmetric, so GS is frequently symmetrized in practice. Another class of general smoothers are the polynomial methods, where M is defined implicitly from $I - M^{-1}A = p(A)$ with p being a polynomial satisfying $p(0) = 1$. Both of these smoothing approaches have been essential in the development of serial algebraic multigrid and have excellent smoothing properties.

Parallel architectures, however, present serious challenges for these algorithms; GS is sequential in nature, while polynomial smoothers need knowledge of the spectrum of the matrix. Thus, a major concern for extending AMG for massively parallel machines has been the development of parallel smoothers that can maintain both scalability and good smoothing properties. In this section we summarize the theoretical and numerical results from [3] for several promising smoothers in the BoomerAMG code. Previous research in parallel smoothers can be found in [1, 19].

The default smoother in BoomerAMG is a parallel version of GS known as hybrid Gauss-Seidel (hybrid-GS), which can be viewed as an inexact block-diagonal (Jacobi) smoother with GS sweeps inside each process. In other words, hybrid-GS corresponds to the block-diagonal matrix M_{HGS}, each block of which equals the process-owned $L + D$ part of A (BoomerAMG matrix storage is row-wise parallel). Even though hybrid-GS has been successful in many applications, its scalability is not guaranteed, since it approaches Jacobi when the number of processors is large, or when the problem size per processor is small. Our strategy for addressing this issue is to investigate different variants of hybrid-GS through a qualitative smoother analysis based on the two-grid theory from [9, 10]. As one application of the theory, we showed [3, 4] that hybrid-GS will be a good smoother when the off-processor part of the matrix rows is smaller than the diagonal in each processor. However, there are practical cases where the off-processor part of the matrix is significant, due to the problem being solved (e.g., definite Maxwell discretizations) or due to the parallel partitioning of A (e.g., due to the use of threading). In these cases, hybrid-GS will behave much worse than GS and can be divergent, even for large problem sizes per processor. To improve the robustness of hybrid-GS, we proposed [16] and analyzed [3] the ℓ_1 Gauss-Seidel smoother (ℓ_1-GS), which corresponds to $M_{\ell_1 GS} = M_{HGS} + D^{\ell_1}$, where D^{ℓ_1} is a diagonal matrix containing the ℓ_1 norms of the off-processor part of each matrix row. This smoother has the nice property that it is convergent for any SPD matrix A.

As a second part of our strategy for scalable multigrid smoothers, we also explore polynomial methods where the high end of the spectrum of A is approximated with several CG iterations and a fixed scaling for the lower bound. The theory from [9] can be applied also in this case to conclude that the best polynomial smoothers are given by shifted and scaled Chebyshev polynomials. To balance cost and performance, in practice we usually use the second order Chebyshev polynomial for $D^{-1/2}AD^{-1/2}$ (Cheby(2)). The cost of this method is comparable with the symmetrized hybrid-GS smoother. Polynomial smoothers have the major advantage that their iterations are independent of the ordering of the unknowns or the parallel partitioning of the matrix. They also need only a matrix-vector multiply routine, which is typically finely tuned on parallel machines. These advantages, however, need to be balanced with the cost of estimating the high end of the spectrum of A. In our experience so far, this cost has not affected the scalability of Cheby(2).

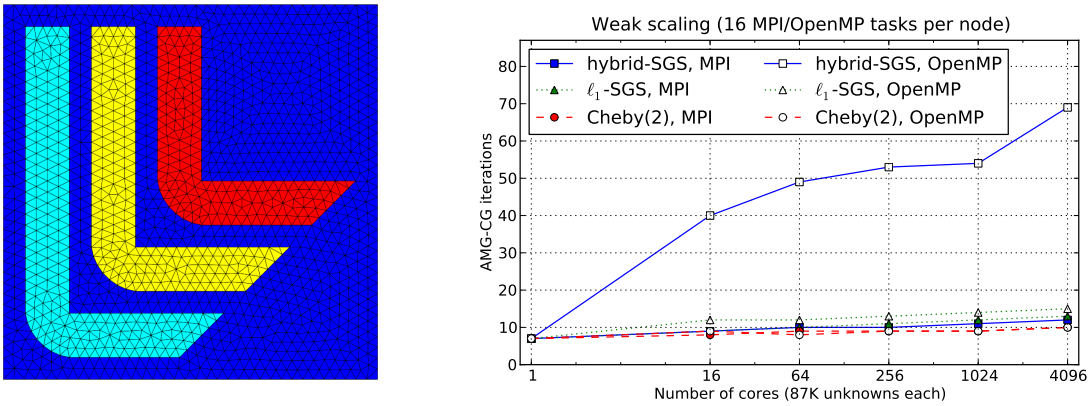

FIG. 3.1. *Coarse mesh for the model problem with indicated material subdomains (left). The diffusion coefficient has 3 orders of magnitude jumps between the interior/exterior materials. Comparison of the scalability of AMG-CG with the different smoothing options when using threading (right).*

We illustrate the numerical performance of Cheby(2) and the symmetrized version of hybrid-GS and ℓ_1-GS with several results from [3]. The test problem describes a variable coefficient diffusion which is posed on the unit square and discretized with unstructured linear triangular finite elements, see Figure 3.1. We report the iteration counts for BoomerAMG used as a two-grid solver (AMG) or a preconditioner in CG (AMG-CG) with a relative convergence tolerance of 10^{-6}.

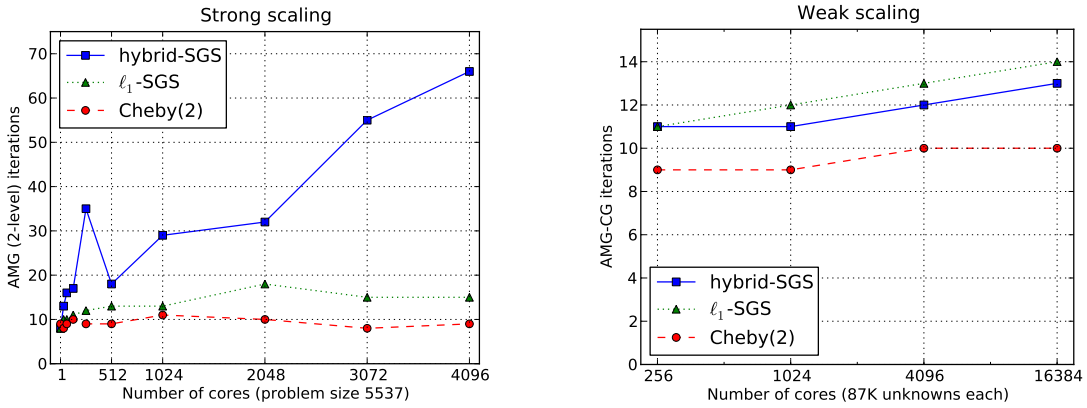

FIG. 3.2. *Strong scaling of a two-level AMG solver with very small problem sizes per processor (left), and weak scalability of AMG-CG with sufficiently large problem sizes per processor (right).*

We first investigate the impact of threading through several weak scaling runs that alternate between the use of MPI and OpenMP on compute nodes with four quad core processors each for a total of 16 cores per node. In this particular application, the numbering of the unknowns inside each processor is not guaranteed to have good locality, so the straightforward (not application-assisted) use of OpenMP introduces a bad partitioning onto the cores. As suggested by the theory, the performance of hybrid-SGS deteriorates significantly in this case, while ℓ_1-SGS and Cheby(2) remain robust. In contrast, in the MPI case the application provides parallel partitioning with a good constant θ, so all methods scale well. Note that the MPI weak scaling results on the right in Figure 3.2 indicate that with application-assisted parallel partitioning, all smoothers can lead to good weak scalability on very large number of processors.

Finally, we demonstrate the impact of small problem sizes per processor through the strong scaling runs presented on the left in Figure 3.2. This is an important test case, given the expected memory availability on future architectures. Since small problem sizes per processor are correlated with a large off-processor part of the matrix rows, the hybrid-SGS method deteriorates, as the smoothing analysis indicates is possible. In contrast, the effect on both ℓ_1-SGS and Cheby(2) is minimal, which is reassuring for their use on multi-core machines with small amounts of node memory.

4. Performance model. Since we strive to better understand the issues impeding the scalability of AMG on HPC platforms, we developed a performance model for the AMG solve cycle.

We modeled the computation time by multiplying the number of floating-point operations with the time per flop t_c. The flops in the AMG solve cycle are incurred as a result of a sparse matrix-vector multiplication (MatVec) for the interpolation and restriction steps and the similar operation of applying the smoother. Note that an in-depth study [11] found the floating-point rate for the MatVec operation to vary widely depending on the size of the matrix and vector. For this reason, we allow t_c to vary depending on the level, and denote the time per flop on level i with t_i. For communication, we started with the basic α-β model for interprocess communication, which breaks down the cost of communication into the start-up time α (latency) and the per-element send time β (inverse bandwidth). If a message has n elements in it, then the send cost is

$$T_{\text{send}} = \alpha + n\beta.$$

Note that α covers both the software overhead and the latency involved in message passing, and β is tied to the achievable bandwidth. To improve upon the basic model, penalties were added to the parameters to take into account machine-specific performance issues. In particular, a γ term was added to take into account communication distance and switching delays on the interconnect. We penalized β to account for limited bandwidth, and α and γ were penalized to account for performance degradation arising from multiple cores on a single node contending for available resources.

Figure 4.1 shows actual and modeled times per level of an AMG cycle on two different architectures, a BG/P with quad-core nodes and a 3D torus network (Intrepid), and a Linux cluster with nodes consisting of 4 quad-cores and a fat-tree QDR Infiniband network (Hera). On Intrepid most of the time is spent on the finest levels, which one would expect, since most of the floating operations occur there. The basic α-β

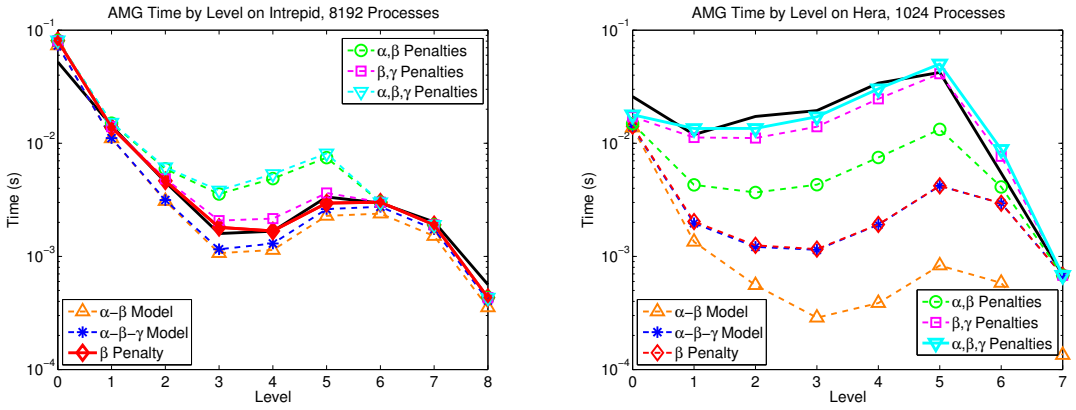

FIG. 4.1. *Model validation results on Intrepid (left), a Blue Gene/P at Argonne National Laboratory, and Hera (right) for a 3D 7pt Laplace problem with 62,500 grid points per process. The black line represents actual times. The last level, which is solved directly using Gaussian Elimination rather than relaxation, is not shown. On Intrepid, the only difficulties are distance of communication and lower effective bandwidth, but on Hera, every penalty applies.*

model is already a fairly good fit. The best fit is obtained using the α-β-γ model and adding the β-penalty. On Hera, most of the time is spent on the coarsest levels, caused by contention among multicore nodes and communication distance. All penalties needed to be applied here to get a good fit. Further details on the performance model as well as its validation on additional architectures can be found in [12]. Work is underway to include the effect of OpenMP threads on the performance.

5. Redundant coarse grid solve. Since our investigations clearly showed that communication on the coarser grids is a bottleneck, we considered algorithmic changes that will decrease communication.

The current default in BoomerAMG is to coarsen until the coarse grid is smaller than 10. At this point the coarsest grid matrix and right hand side is distributed to all processors that still contain coarsest grid points (at most 9), and the coarsest system is then solved redundantly on these processors using Gaussian elimination. If one were to use this strategy on a finer level with significantly more degrees of freedom than 9, which needs to be carefully chosen, one could potentially save a significant amount of communication, even though it requires an initial distribution of more data to a larger amount of processors. Since a direct solve at a finer level would be much too expensive, we use sequential AMG on all remaining processors. Similar approaches have been also considered in [13, 18] who found that this would likely result in improved performance.

We used the performance model to get preliminary estimates of speedups on Hera for the solve cycle of the new variant over the original AMG, depending on the level on which we apply the redundant coarse grid solve. The results are presented in Figure 5.1. While the modeled speedups for the solve phase are somewhat more optimistic than the actually achieved ones, the model predicted the levels on which one obtains the optimal improvements accurately.

The model only predicted solve cycle times, however, even larger speedups were obtained for the AMG setup phase, which contains more complex communication patterns. We present the actually achieved speedups for the total times, including both setup and solve times for AMG-CG, on 3 different multicore architectures for two different problems in Table 5.1, showing that one can achieve significant speedups using the redundant coarse grid approach.

Work is underway on a more sophisticated version, which is expected to lead to additional performance improvements.

6. Use of a Hybrid Programming Model. In this section we investigate the use of a hybrid MPI/OpenMP programming model for BoomerAMG on three different multi-core architectures: the quad-core/quad-socket Opteron/Infiniband cluster Hera at Lawrence Livermore National Laboratory (up to 11,664 cores), the dual hex-core Cray XT-5 Jaguar at Oak Ridge National Laboratory (up to 196,608 cores), and the quad-core Blue Gene/P system Intrepid at Argonne National Laboratory (up to 128,000 cores). On each machine, we investigate an MPI-only version of AMG, as well as hybrid versions that use a mix of MPI and

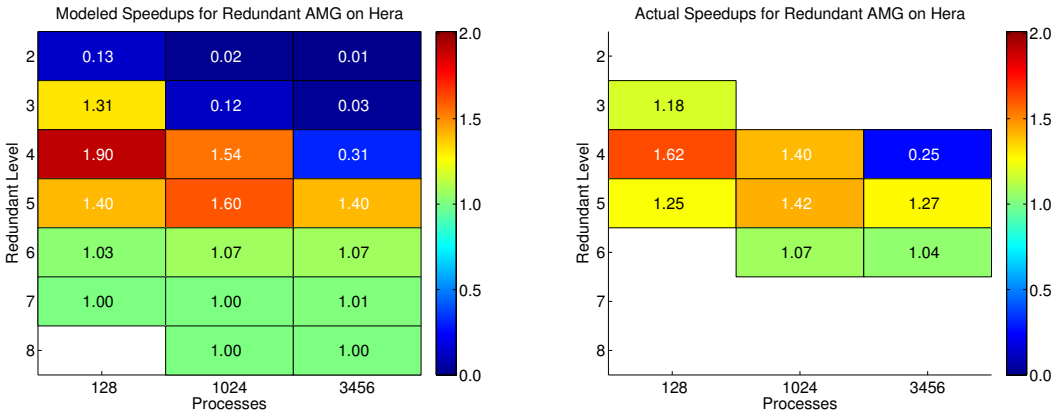

FIG. 5.1. *Modeled and actual speedups for the solve phase when using redundant solve on Hera applied to a 3D 7pt Laplace problem with 62,500 grid points per process.*

	7pt 3D Laplace problem			27pt stencil		
No. of cores	Hera	Atlas	Coastal	Hera	Atlas	Coastal
128	1.77	1.11	1.62	1.40	1.03	1.31
432	2.00	1.11	2.07	1.39	1.20	1.64
1024	1.88	1.25	2.09	1.57	1.25	1.93
2000	1.79	1.18	2.04	1.62	1.25	1.66

TABLE 5.1

Speedup of the redundant coarse grid solve (total run times for AMG-CG) on 3 different multicore architectures for two different problems with 62,500 grid points per core. One node on Hera has 4 quad-core 2.3 GHz AMD Opteron processors per node, Atlas has 4 dual-core 2.4 GHz AMD Opteron processors per node, and Coastal has 2 four-core 2.5 GHz Intel Xeon processors per node. All nodes are connected by an Infiniband interconnect.

OpenMP on node. For each experiment, we utilize all available cores per node on the respective machine. We investigate the performance of AMG-GMRES(10) applied to a Laplace problem on a domain of size $N \times N \times \alpha N$, where $\alpha = 1$ on Hera and Intrepid, and $\alpha = 0.9$ on Jaguar, to allow more optimal partitioning when using 6 or 12 threads per node. The domain is decomposed in such a way that each processor has $50 \times 50 \times 25$ unknowns on Hera and Intrepid and $50 \times 50 \times 30$ on Jaguar. We consider both MPI-only and hybrid MPI/OpenMP runs and use the notation described in Figure 6.1. In addition, for Hera, we include a version, labeled "H*mxn*MC", that uses the MCSup (Multi-Core Support) library, which provides the user with an API to allocate memory in a distributed manner across all processors in a way that matches the implicit thread distribution of OpenMP and to pro-actively pin the threads to the processors and so eliminates most remote memory accesses and reduces contention [7, 6, 2].

We use hybrid-GS as a smoother. The number of iterations to convergence varies across experimental setups from 17 to 44. Note that, since both the coarsening algorithm and the smoother are dependent on the number of tasks and the domain partitioning among MPI tasks and OpenMP threads, the number of iterations can vary for different combination of MPI tasks and OpenMP threads, even when using the same problem size and number of cores. We present total times in Figure 6.1. Separate setup and cycle times for this problem on the three architectures are described in the original paper [6].

It is immediately apparent that on the two NUMA architectures, Hera and Jaguar, the MPI-only versions as well as H12x1 on Jaguar, perform significantly worse than the other versions, whereas on Intrepid the MPI-only version generally performs best, with the exception of the less optimal processor geometries of 27,648 and 128,000 cores, where H2x2 is somewhat better. The worst performance on Intrepid is observed when using 4 threads per MPI task. Note that not all of the setup phase is threaded, leading to less parallelism when OpenMP is used, causing the lower performance of H1x4 on Intrepid, which has significantly slower cores than Jaguar or Hera. Interestingly enough this effect is not notable on Hera and Jaguar, which are, however, severely effected by the fact that the algorithms in the setup phase are complex and contain a large amount of non-collective communication leading to a large communication overhead and network contention. This

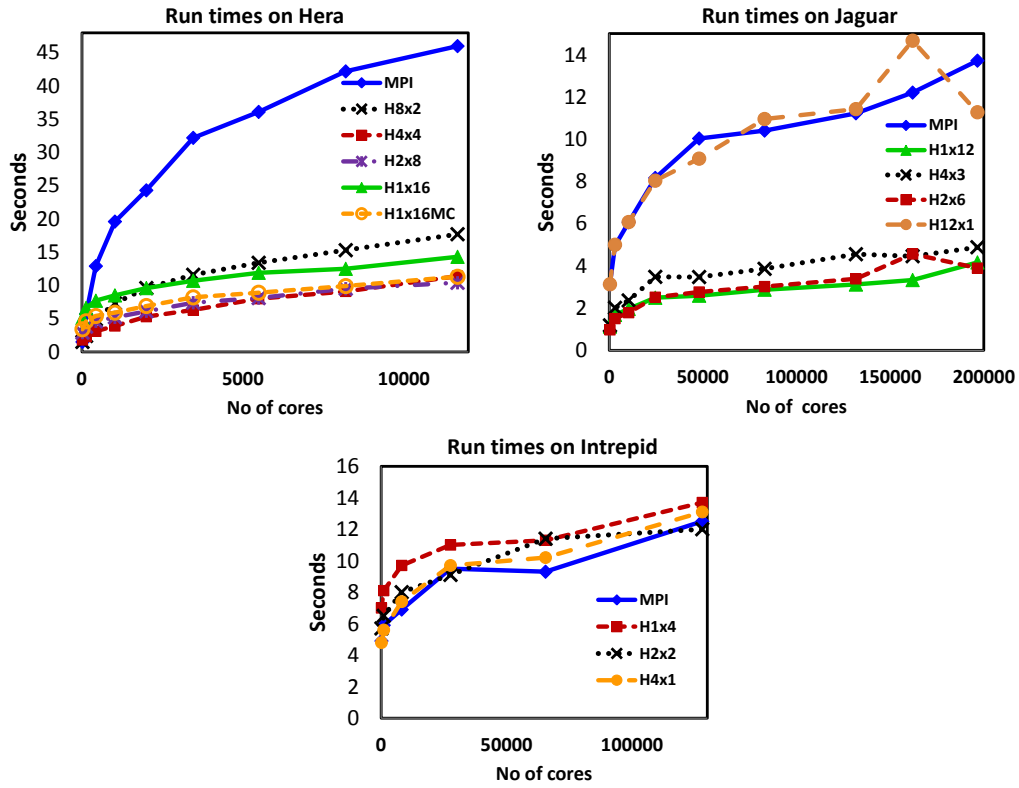

FIG. 6.1. *Total times for AMG-GMRES(10) applied to a 7-point 3D Laplace problem on three different multi-core architectures; Hm × n denotes runs performed with m MPI tasks per cluster and n OpenMP threads per MPI task, "MPI" denotes runs performed with the MPI-only version, Hm × nMC denotes the use of MCSup.*

effect is still visible, but less pronounced in the solve phase, which has a smaller amount of communication. On Hera, the worst performance for the solve phase is obtained for H1x16 (see [6, 7]), and is caused by the NUMA architecture. Using the *MCSup* library, see H1x16MC, performance is significantly improved. In the setup phase there is no NUMA effect for H1x16, since it mainly uses temporary storage, which is allocated within an OpenMP thread and therefore implicitly placed into the right memory module. For Hera and Jaguar initially the best times are obtained for the version that maps best to the architecture (H4x4 for Hera and H2x6 for Jaguar), to be then surpassed by H2x8 for Hera and H1x12 for Jaguar, versions with smaller network contention. Note that for the largest run on Jaguar, H1x12 takes more iterations than H2x6, causing H2x6 to be faster.

7. Conclusions. We investigated algebraic multigrid for exascale machines and considered both mathematical as well as computer science aspects to achieving scalability. Our investigation of smoothers showed that hybrid-GS promises to work well for certain problems even when we are dealing with millions or billions of cores. For more complicated problems ℓ_1-GS and polynomial smoothers are a viable fully parallel alternative to hybrid-GS because their convergence is not affected by the high level of parallelism needed for efficient implementations on exascale machines. We developed a performance model for the AMG solve cycle and showed that distance of communication and contention caused by the communication complexity on the coarsest levels are performance bottlenecks on multicore architectures with large communication latencies. We used the model to evaluate the redundant coarse grid variant of AMG and validated that this variant can perform significantly better on such architectures. An alternative approach, the use of a hybrid MPI/OpenMP programming model, also led to significant performance improvements on two of the architectures considered. Our tests also demonstrated that a general solution is not possible without taking into account the specific target architecture. All techniques presented here have been implemented in the hypre library and are available in the latest release. We have also added a 64bit integer option, which allows the solution of systems that are larger than 2 billion unknowns.

Acknowledgments: Most of the basic research presented here was funded by LDRD 10-SI-014. The investigation of the hybrid MPI/OpenMP programming model was leveraged with ASCR SciDAC2-TOPS funding. All implementations into the hypre library were performed using ASC FRIC funding. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. It also used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357, as well as resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. These resources were made available via the Performance Evaluation and Analysis Consortium End Station, a Department of Energy INCITE project. Neither Contractor, DOE, or the U.S. Government, nor any person acting on their behalf: (a) makes any warranty or representation, express or implied, with respect to the information contained in this document; or (b) assumes any liabilities with respect to the use of, or damages resulting from the use of any information contained in the document. LLNL-TR-533076.

REFERENCES

[1] M. Adams, M. Brezina, J. Hu, and R. Tuminaro. Parallel multigrid smoothing: Polynomial versus Gauss-Seidel. *J. Comput. Phys.*, 188:593–610, 2003.

[2] A. H. Baker, R. D. Falgout, T. Gamblin, T. V. Kolev, M. Schulz, and U. M. Yang. Scaling Algebraic Multigrid Solvers: On the Road to Exascale. In C. Bischof, H.-G. Hegering, W. Nagel, and G. Wittum, editors, *Competence in High Performance Computing 2010*. Springer, 2012.

[3] A. H. Baker, R. D. Falgout, T. V. Kolev, and U. M. Yang. Multigrid smoothers for ultra-parallel computing. *SIAM Journal on Scientific Computing*, 33:2864–2887, 2011.

[4] A. H. Baker, R. D. Falgout, T. V. Kolev, and U. M. Yang. Multigrid smoothers for ultra-parallel computing - additional theory and discussion. Technical Report LLNL-TR-489114, Lawrence Livermore National Laboratory, 2011.

[5] A. H. Baker, R. D. Falgout, T. V. Kolev, and U. M. Yang. Scaling hypre's Multigrid Solvers to 100,000 Cores. In M. Berry, K. Gallivan, E. Gallopoulos, A. Grama, B. Philippe, Y. Saad, and F. Saied, editors, *High Performance Scientific Computing: Algorithms and Applications*, pages 261–279. Springer, 2012.

[6] A. H. Baker, T. Gamblin, M. Schulz, and U. M. Yang. Challenges of scaling algebraic multigrid across modern multicore architectures. In *Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2011)*, pages 275–286, 2011.

[7] A. H. Baker, M. Schulz, and U. M. Yang. On the performance of an algebraic multigrid solver on multicore clusters. In J.M.L.M. Palma et al, editor, *VECPAR 2010, Lecture Notes in Computer Science 6449*, pages 102–115. Springer, 2010. Berkeley, CA, June 2010. http://vecpar.fe.up.pt/2010/papers/24.php.

[8] R. D. Falgout. An introduction to algebraic multigrid. *Computing in Science and Eng.*, 8(6):24–33, 2006.

[9] R. D. Falgout and P. S. Vassilevski. On generalizing the algebraic multigrid framework. *SIAM J. Numer. Anal.*, 42(4):1669–1693, 2004. UCRL-JC-150807.

[10] R. D. Falgout, P. S. Vassilevski, and L. T. Zikatanov. On two-grid convergence estimates. *Numer. Linear Algebra Appl.*, 12(5–6):471–494, 2005. UCRL-JRNL-203843.

[11] H. Gahvari. Benchmarking Sparse Matrix-Vector Multiply. Master's thesis, University of California, Berkeley, December 2006.

[12] H. Gahvari, A. H. Baker, M. Schulz, U. M. Yang, K. E. Jordan, and W. Gropp. Modeling the Performance of an Algebraic Multigrid Cycle on HPC Platforms. In *25th ACM International Conference on Supercomputing*, Tucson, AZ, June 2011.

[13] W. Gropp. Parallel Computing and Domain Decomposition. In T. Chan, D. Keyes, G. Meurant, J. Scroggs, and R. Voigt, editors, *Fifth Conference on Domain Decomposition Methods for Partial Differential Equations*, pages 349–361. SIAM, 1992.

[14] V. E. Henson and U. M. Yang. BoomerAMG: a parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics*, 41:155–177, 2002.

[15] *hypre*. High performance preconditioners. http://www.llnl.gov/CASC/linear_solvers/.

[16] T. Kolev and P. Vassilevski. Parallel auxiliary space AMG for H(curl) problems. *J. Comput. Math.*, 27:604–623, 2009.

[17] K. Stüben. An introduction to algebraic multigrid. In U. Trottenberg, C. Oosterlee, and A. Schüller, editors, *Multigrid*, pages 413–532. Academic Press, London, 2001.

[18] D. E. Womble and B. C. Young. A model and implementation of multigrid for massively parallel computers. *International Journal of High Speed Computing*, 2:239–255, 1990.

[19] U. M. Yang. On the use of relaxation parameters in hybrid smoothers. *Numer. Linear Algebra Appl.*, 11:155–172, 2004. UCRL-JC-151575.