Modeling the Performance of an Algebraic Multigrid Cycle on HPC Platforms

Hormozd Gahvari[1], Allison H. Baker[2], Martin Schulz[2]
Ulrike Meier Yang[2], Kirk E. Jordan[3], William Gropp[1]

[1]Computer Science Department, University of Illinois at Urbana-Champaign, Urbana, IL 61801
[2]Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA 94551
[3]IBM TJ Watson Research Center, Cambridge, MA 02142

gahvari@illinois.edu, abaker@llnl.gov, schulzm@llnl.gov
umyang@llnl.gov, kjordan@us.ibm.com, wgropp@illinois.edu

ABSTRACT

Now that the performance of individual cores has plateaued, future supercomputers will depend upon increasing parallelism for performance. Processor counts are now in the hundreds of thousands for the largest machines and will soon be in the millions. There is an urgent need to model application performance at these scales and to understand what changes need to be made to ensure continued scalability. This paper considers algebraic multigrid (AMG), a popular and highly efficient iterative solver for large sparse linear systems that is used in many applications. We discuss the challenges for AMG on current parallel computers and future exascale architectures, and we present a performance model for an AMG solve cycle as well as performance measurements on several massively-parallel platforms.

Categories and Subject Descriptors

G.4 [**Mathematical Software**]: Algorithm design and analysis, Parallel and vector implementations; G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra—*Linear systems (direct and iterative methods)*

General Terms

Algorithms, Performance

Keywords

Algebraic Multigrid, Scaling, Performance Modeling, Massively Parallel Architectures

1. INTRODUCTION

Multigrid methods are popular for the solution of large sparse linear systems, which is a necessary and often time-consuming element of many large-scale scientific simulation codes. The parallel AMG solver BoomerAMG [11] in the hypre software library [12], for example, is an integral component in simulations in diverse areas such as groundwater flow, explosive materials modeling, electromagnetic applications, fusion energy simulations, and image-guided facial surgery. Multigrid methods have the "optimal" property that, when they work well, the amount of work per unknown stays constant. This property is especially attractive for parallel computing: as the size of supercomputers increases, we can solve increasingly larger problems in a roughly fixed amount of time.

Since the performance of AMG has a profound impact on a wide variety of applications across a wide range of disciplines, it is crucial to understand the challenges for future architectures with increased parallelism as well as to predict and locate performance bottlenecks that hinder performance. Therefore, given that currently no computers exist with several millions or billions of cores available, the development of performance models to evaluate algorithm performance has become very important to prepare application codes for exascale computing and beyond.

In this paper, we develop a novel performance model for the solve phase of the AMG algorithm. To our knowledge this is the first formal characterization of this important application. We start with the basic α-β model for communication combined with an analytical model of the computation. We then add penalties based on machine constraints, including distance effects, reduced per core bandwidth, and the number of cores per node. We validate the model on several parallel platforms and illustrate various challenges to the scalability of AMG, including the increasing communication complexity on coarser grids and the effects of increasing numbers of cores per node on the performance.

We make the following contributions:

- We present a performance model for the AMG solve cycle and validate it across various multicore architectures.
- We expose several bottlenecks on the various architectures using the AMG model.
- We discuss model-based predictions for the scalability of AMG on future machines.

This paper proceeds as follows. Section 2 discusses related work. Section 3 summarizes AMG and the performance challenges it faces on parallel machines. Section 4

ICS'11, May 31–June 4, 2011, Tuscon, Arizona USA.

describes our performance model, and Section 5 presents experiments done to validate it. Section 6 discusses lessons from the model results, followed by concluding remarks in Section 7.

2. RELATED WORK

A wide range of related projects target the modeling of numerical codes on large scale parallel systems. For example, [13] provides an analytical model for the application SAGE, [15] describes an approach to combine computation and communication profiles into a general performance model, and [3] targets the prediction of large scale performance behavior based on the extrapolation of small scale performance results.

While the issue of scaling AMG to higher and higher process counts has been a subject of much study recently, no performance models for AMG have been developed yet. In [2] and [1], changing the programming model is investigated as a means of better matching emerging multicore clusters and improving AMG performance. Performance models and their implications for geometric multigrid on exascale systems were considered in [8], but geometric multigrid is a less complex algorithm than AMG and does not suffer from the same performance degradation on the coarser grid levels. Also of interest is a brief analysis of when it would be preferable to use redundant computation to replace communication in [10], but again this analysis is for geometric multigrid, not AMG. Finally, the moving of data between main memory and cache can be a significant factor in the application performance, and this is examined for multigrid in [5]. We have not considered this at this time, but the underlying message of reducing data movement also motivates our work.

3. ALGEBRAIC MULTIGRID

Multigrid methods are well-suited for large-scale scientific applications because they are algorithmically scalable, i.e. they solve a sparse linear system $A^{(0)}u = f^0$ with n unknowns with $O(n)$ computations. They obtain this optimality by eliminating "smooth error", e, that is not removed by relaxation (or smoothing) by coarse-grid correction using successively smaller grids. Algebraic multigrid (AMG) does not require an explicit grid. Instead coarse grid selection and the generation of interpolation and restriction operators depend entirely on the matrix coefficients. For a detailed description of AMG, see [18], for example.

AMG consists of a setup phase and a solve phase, as illustrated in Figure 1. We describe and analyze the simplest multigrid cycle, the V-cycle, but our approach can be straightforwardly extended to analyze the more complicated W-cycle and full multigrid cycle. In the setup phase, the coarse grid variables, interpolation operators $P^{(m)}$, restriction operators $R^{(m)}$ and the coarse grid matrices $A^{(m+1)}$, are determined for $m = 0, ..., k-1$. The coarsest level $k-1$ is reached when $A^{(k)}$ is sufficiently small. In our experiments, $A^{(k)}$ has at most nine unknowns. Note from Figure 1 that the coarse grid matrices are determined via a triple matrix product, and, as a result, the "stencil size" on each grid level tends to increase as we coarsen. In other words, the coarse grid matrices are less sparse and require communication with more neighbor processes (to perform a matrix-vector multiplication, for example) than the fine grid matrices.

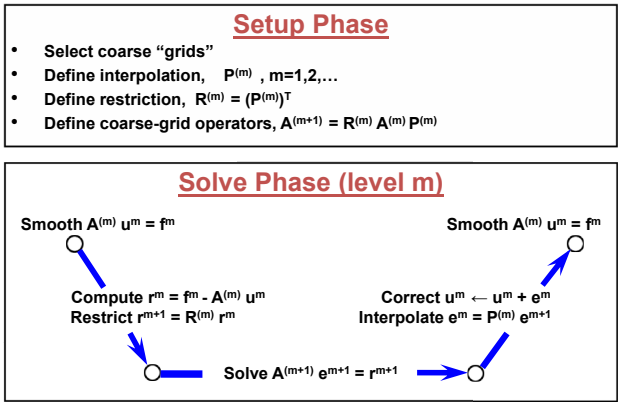


Figure 1: AMG building blocks.

In the solve phase, a smoother is applied on each level $m = 0, ..., k-1$, and then the residual r^m is transferred to the next coarser grid, where the process continues. On the coarsest level, the linear system $A^{(k)}e^k = r^k$ is solved by Gaussian elimination. The error e^k is then interpolated back up to the next finer grid, followed by relaxation. This is continued all the way up to the finest grid. The m-th level of the solve phase is described in Figure 1. The primary components of the solve phase are the matrix-vector multiplication (MatVec) and the smoother. The classical smoother used for algebraic multigrid is Gauss-Seidel, which is highly sequential. Therefore, we use a parallel variant, called hybrid Gauss-Seidel, which can be viewed as an inexact block-diagonal (Jacobi) smoother with Gauss-Seidel sweeps inside each process. In other words, we use a sequential Gauss-Seidel algorithm locally on each process, with delayed updates across processes. One sweep of hybrid Gauss-Seidel is very similar to a MatVec.

For our experiments, we use BoomerAMG, the parallel AMG code in the hypre software library. We use HMIS coarsening [17] with extended+i interpolation [16] truncated to at most 4 coefficients per row and aggressive coarsening with multipass interpolation [19] on the finest level.

3.1 Parallel Implementation

BoomerAMG uses the ParCSR matrix data structure, which is based on the sequential compressed sparse row (CSR) storage format. The ParCSR matrix A consists of p parts $A_k, k = 1, ..., p$, where A_k is stored locally on process k. Each A_k is split into two matrices D_k and O_k. D_k contains all coefficients of A_k, whose column indices point to rows that are stored locally on process k. O_k contains the remaining coefficients of A_k. Both matrices are stored in CSR format. Whereas D_k is a CSR matrix in the usual sense, for O_k, which in general is extremely sparse with many zero columns and rows, all non-zero columns are renumbered for greater efficiency, requiring an additional array that defines the mapping of local to global column indices.

In order to perform a parallel MatVec or smoothing step, process k needs to evaluate $A_k x = D_k x^D + O_k x^O$, where x^D is the local part of vector x and x^O the portion that needs to be received from other processes (receive processes). In order to receive the required information as well as to send information needed by other processes (send processes), each process has a communication package that contains the fol-

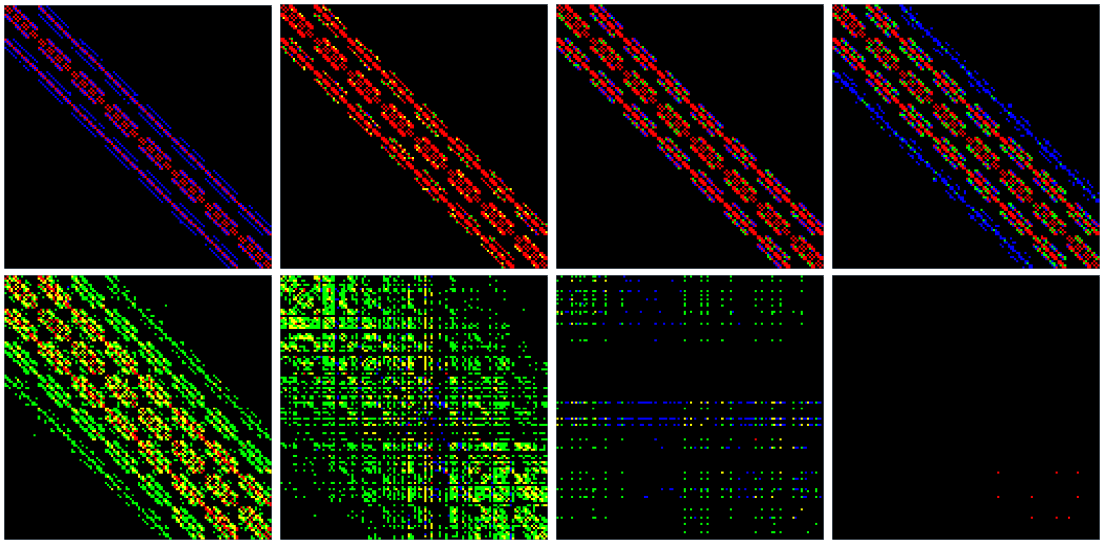


Figure 2: Level-by-level communication patterns for an AMG solve on a 7-point $200 \times 200 \times 200$ Laplace problem using 128 processes. Levels 0 (finest grid) through 3 (left to right) are on the top row, and levels 4 through 7 (left to right) are on the bottom row. Areas of black indicate zero messages between processes.

lowing information: the IDs of the receive processes, the size of the data to be received by each receive process, the IDs of the send processes, and the indices of the elements that need to be sent to each send process. The actual communication is then performed by posting non-blocking receives (MPI_Irecv) to each receive process followed by non-blocking sends (MPI_Isend) and finalized by an MPI_Waitall to all posted operations. See [6] for more discussion on the implementation details.

Regarding the distribution of data on the coarser levels, coarse points are kept on the same processes that they were located on at the finer levels. Because the number of coarse points will eventually be smaller than the total number of processes, on the coarser levels processes will start "dropping out" when they no longer own any rows in the matrix. Therefore, while a process's neighbors will be "close" in terms of process ranks at the fine levels, on the coarser levels neighbors will be farther away and messages will be smaller in size. On the coarsest level, where at most nine processes are still active, the remaining data (matrix and vectors) is distributed to all active processes, each of which then solves the coarsest system using Gaussian elimination.

3.2 Performance Challenges

The challenges to achieving good parallel AMG performance mainly center around performance degradation on coarse grids. As mentioned before, each process's communication partners will be farther away on the coarser grids than on the fine grids. There is also little computation on the coarser levels due to the smaller matrix sizes, so communication dominates the time spent here. Delays in sending messages to distant processes can cause scalability concerns, as an analysis in [8] found. This analysis, however, targeted only geometric multigrid for five-point and seven-point Laplace problems, where the communication pattern remains a simple, fixed stencil on all grids. When AMG is employed to solve the same problems, the communication pattern starts off as a simple stencil, but eventually becomes more irregular and involves far more communication partners, or in other words, increases the communication neighborhood for each process.

An initial performance experiment confirms this behavior: Figure 2 (obtained using the performance analysis tool TAU [14]) shows the communication between pairs of processes on each level of an AMG solve using 128 processes. Initially, on level 0, communication is regular and mostly focused on nearest neighbors. In subsequent levels, the communication neighborhood grows until, in level 5, it covers almost the complete process space. At the same time, we see several processes leave the communication pattern (black horizontal and vertical lines). In level 6, these dropped processes start to dominate, and in level 7, only a few processes remain. The communication and computation statistics from our subsequent experiments to validate our performance model, which are in Table 1, also highlight this trend. On the finer grids, computation time dominates the execution time, but then on the coarser grids, the amount of data per process gets small and communication time dominates instead.

Further challenges arise due to the trend of increasing numbers of cores on each node of a massively-parallel machine. The numerous cores on a single node contend for access to the interconnect, which slows down coarse grid performance even further because of the large number of messages that need to be sent on those coarse levels. Despite the fine grid matrix being many orders of magnitude larger than those matrices on the coarsest levels, the solve on the coarse grid can take as long as the solve on the fine grid. For example, this performance problem occurs when using 1024 processes on the Hera machine (machine specifications are given in Section 5). The computation on the fine grid problem, which has 64 million unknowns, took 25.9 ms, but the computation on a much coarser grid level with only 1224 unknowns took 42.3 ms. More generally, further examples of this unexpected and troubling phenomenon are seen in the results in Section 5. Note that two prior studies [2, 1] have

examined this issue to a limited extent and found that using OpenMP on the individual nodes and pinning threads to cores and processes to sockets can alleviate these problems to some degree, but not completely. We discuss multicore issues when presenting our performance models, but the use of OpenMP and pinning of threads and processes is beyond the scope of this paper.

4. MODELING PERFORMANCE

To understand the performance of AMG and predict its performance on future machines, we develop a performance model for the solve cycle that requires minimal machine-specific information. We first start with models for local computation and communication. Combined, these two models form our baseline model, which we consequently refine to reach a complete model that is able to cover the relevant system architecture properties.

For this we define the following terms:

- P – total number of processes
- C_i – number of unknowns on grid level i
- s_i, $\hat{s}_i$ – average number of nonzeros per row in the level i solve and interpolation operators, respectively
- p_i, $\hat{p}_i$ – maximum number of sends over all processes in the level i solve and interpolation operators, respectively
- n_i, $\hat{n}_i$ – maximum number of elements sent over all processes in the level i solve and interpolation operators, respectively
- t_i – time per flop on level i

We do not consider the overlap of communication and computation here, as on coarse grids there is hardly any computation available for this purpose. The use of maximum numbers of sends accounts for the use of nonblocking communication and MPI_Waitall, as the processes that are waiting are waiting for the one that is doing the most communication to finish. For all AMG solves, we assume one smoothing step before restricting and one smoothing step after interpolation (the default in BoomerAMG).

4.1 Modeling the AMG Steps

The computation time is modeled by multiplying the number of floating-point operations by 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 [7] 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.

We model the AMG solve cycle by modeling each level individually and write the total time of one AMG solve cycle as

$$T_{\text{solve}}^{\text{AMG}} = \sum_{i=0}^{G} T_{\text{solve}}^{i},$$

where G is the number of grid levels and T_{solve}^{i} is the time spent in the solve cycle at level i. We then split the time at each level, T_{solve}^{i}, into the time spent smoothing, restricting, and interpolating on that level:

$$T_{\text{solve}}^{i} = T_{\text{smooth}}^{i} + T_{\text{restrict}}^{i} + T_{\text{interp}}^{i}.$$

Here, T_{smooth}^{i} is the time spent smoothing on level i, T_{restrict}^{i} is the time spent restricting from level i to level $i+1$, and T_{interp}^{i} is the time spent interpolating from level i to level $i-1$.

4.2 Modeling Communication

For communication, we start 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, we then add penalties to the parameters to take into account machine-specific performance issues. In particular, we add a γ term to take into account communication distance and switching delays on the interconnect. We penalize β to account for limited bandwidth, and we penalize α and γ to account for performance degradation arising from multiple cores on a single node contending for available resources.

4.3 Baseline Model (α-β Model)

To reach our baseline model, we apply the communication model in the description of the three main steps and deduce formulas for each step based on the algorithmic requirements of the AMG implementation.

The complete time for the smoother at level i is given by

$$T_{\text{smooth}}^{i}(\alpha,\beta) = 6\frac{C_i}{P}s_i t_i + 3(p_i\alpha + n_i\beta).$$

This reflects one smoother application before restricting, one MatVec to form the residual, and one smoother application after interpolation, with two flops (one multiplication and one addition) per matrix entry.

The time for restricting on level i is given by

$$T_{\text{restrict}}^{i}(\alpha,\beta) = \begin{cases} 2\frac{C_{i+1}}{P}\hat{s}_i t_i + \hat{p}_i\alpha + \hat{n}_i\beta & \text{if } i < G \\ 0 & \text{if } i = G. \end{cases}$$

This reflects the cost of one MatVec that represents restriction from level i to level $i+1$.

The time for interpolation on level i given by

$$T_{\text{interp}}^{i}(\alpha,\beta) = \begin{cases} 0 & \text{if } i = 0 \\ 2\frac{C_{i-1}}{P}\hat{s}_{i-1} t_i + \hat{p}_{i-1}\alpha + \hat{n}_{i-1}\beta & \text{if } i > 0. \end{cases}$$

This reflects the cost of one MatVec that represents interpolation from level i to level $i-1$.

Therefore the complete baseline model is given by

$$T_{\text{solve}}^{\text{AMG}}(\alpha,\beta) = \sum_{i=0}^{G} T_{\text{solve}}^{i}(\alpha,\beta),$$

where

$$T_{\text{solve}}^{i}(\alpha,\beta) = T_{\text{smooth}}^{i}(\alpha,\beta) + T_{\text{restrict}}^{i}(\alpha,\beta) + T_{\text{interp}}^{i}(\alpha,\beta).$$

4.4 Distance Penalty (α-β-γ Model)

In modern interconnection networks it is assumed that distance does not have much effect on communication time. However, with many messages being sent at once, as is the case for coarse grids in AMG, this is no longer a safe assumption. On larger machines distance will be an even bigger factor. To take this into account, we replace the α in the baseline model by $\alpha(h) = \alpha(h_m) + (h - h_m)\gamma$, where h is the number of hops a message travels, h_m is the smallest possible number of hops a message can travel in the network, and γ is the delay per extra hop. This covers issues of switching delays and, to some extent, network contention. For machines with a mesh or torus interconnect, $h_m = 1$, and h is assumed to be the diameter of the network formed by the number of nodes being used, to take into account routing delays and possible "long hops" across a large machine room. For machines with a fat-tree interconnect, the shortest message travels one switch, or two links, so $h_m = 2$. The fat-tree machines we consider here have two-level trees, so h is 4, as each message passes through at most four links.

In terms of the baseline model, which was expressed as $T^{\text{AMG}}_{\text{solve}}(\alpha, \beta)$, a function of α and β, we get

$$\tilde{T}^{\text{AMG}}_{\text{solve}}(\alpha, \beta, \gamma) = T^{\text{AMG}}_{\text{solve}}(\alpha(h_m) + (h - h_m)\gamma, \beta).$$

4.5 Bandwidth Penalty (on β)

The peak hardware bandwidth is rarely achieved in message passing under ideal conditions using typical message sizes. This achievable bandwidth is in turn rarely achieved under non-ideal conditions. We take this into account by multiplying β by $\frac{B_{\max}}{B}$, where $B_{\max}$ is the peak hardware per-node bandwidth, and B is the bandwidth corresponding to β (if B is in bytes per second, and β is the time to send one double-precision floating point value, we would have $B = \frac{8}{\beta}$). The fraction $\frac{B_{\max}}{B}$ provides a measure of how much worse than ideal the available bandwidth actually is. The formula for the resulting model becomes

$$T^{\beta}_{\text{penalty}} = \tilde{T}^{\text{AMG}}_{\text{solve}}\left(\alpha, \frac{B_{\max}}{B}\beta, \gamma\right).$$

4.6 Multicore Penalty (on α and/or γ)

As mentioned previously, the increasing number of cores per node on parallel machines brings an additional set of challenges. Among other issues, there is increased contention between cores on a node to get onto the interconnect as well as additional noise caused by accesses to resources shared by multiple cores. While a precise accounting of all of these problems is essentially impossible, we model these effects with a focus on worst-case behavior, in particular on machines in which the aggregate bandwidth that could be generated by all cores communicating exceeds the per node bandwidth. We address this by multipling one or both of the terms $\alpha(h_m)$ and γ by $\left\lceil c\frac{P_i}{P}\right\rceil$. Parameter c is the number of cores per node, and P_i is the number of active processes on level i, meaning those processes that have not "dropped out" and have work to do on that level. The resulting models become

$$T^{\alpha}_{\text{penalty}} = \tilde{T}^{\text{AMG}}_{\text{solve}}\left(\left\lceil c\frac{P_i}{P}\right\rceil \alpha, \beta, \gamma\right)$$

and

$$T^{\gamma}_{\text{penalty}} = \tilde{T}^{\text{AMG}}_{\text{solve}}\left(\alpha, \beta, \left\lceil c\frac{P_i}{P}\right\rceil \gamma\right).$$

5. MODEL VALIDATION

In this section, we first describe the considered architectures and the experimental setup, including the test problem, and then present experimental results to validate the model.

5.1 Machine Descriptions

To test our performance models, we run a series of experiments on the five architectures described in this section.

Intrepid is a large IBM BlueGene/P system at Argonne National Laboratory, consisting of 40 racks with 1024 compute nodes per rack. On each node is a quad-core 850 MHz PowerPC 450 processor. The nodes are connected by a proprietary 3D torus interconnect. The hardware bandwidth between nodes is 5.1 GB/s. On the software side, the compute nodes run a specialized small footprint compute node kernel. Further, for all experiments we use IBM's compiler and the BG/P derivative MPICH-2 version.

Jaguar is a hybrid Cray system at Oak Ridge National Laboratory consisting of both XT5 and XT4 nodes. These are organized into two partitions, one that is XT5 and one that is XT4. We run on the XT5 partition, which has 18,688 compute nodes in all. On each node are two hex-core 2.6 GHz AMD Opteron processors. However, we only use eight cores per node, as our test problem was created with power-of-two core counts in mind. The nodes are connected by a 3D torus interconnect. The hardware bandwidth between nodes is 6.4 GB/s. On the software side, the compute nodes run Compute Node Linux. Further, for all experiments we used PGI's compiler suite and Cray's native MPI implementation.

Hera is a Linux cluster at Lawrence Livermore National Laboratory consisting of 800 compute nodes, with four quad-core 2.3 GHz AMD Opteron processors per node. The nodes are connected by Infiniband, and organized as a two-level fat-tree topology. The first-stage switches have 24 ports, and the second-stage switches have 288 ports. The hardware bandwidth between nodes is 2.5 GB/s. On the software side, Hera runs CHAOS, a specialized version of RHEL5 adapted for HPC. Further, for all experiments we use gcc 4.1.2 and the MPI implementation is MVAPICH v0.99.

Zeus is a Linux cluster at Lawrence Livermore National Laboratory consisting of 260 compute nodes, with two four-core 2.5 GHz Intel Xeon processors per node. The nodes are connected by an Infiniband interconnect similar to Hera's. The software setup is identical to Hera.

Atlas is a Linux cluster at Lawrence Livermore National Laboratory consisting of 1,072 compute nodes, with four dual-core 2.4 GHz AMD Opteron processors per node. The nodes are connected by an Infiniband interconnect similar to Hera's. The software setup is identical to Hera.

5.2 Experimental Setup

On each of the architectures described above, we ran 10 AMG solve cycles and measured the amount of time spent in each level. We then divided the results by 10 to get a measurement of the time spent in each level for an average solve cycle. While each process takes its own time measurements, because some processes have no work on the coarser grid levels, we report times from the process that takes the most time on the coarsest grid. This strategy ensures that all measurements of time spent in each level are fair. We note that the maximum time spent in each level over all processes, which is the intuitive quantity to measure, is not

	Solve, 1024 Processes					Interpolation, 1024 Processes		
Level	No. Sends	Elems. Sent	Unknowns	NNZ/row	Active Procs.	No. Sends	Elems. Sent	NNZ/row
0	6	10000	64000000	7.0	1024	19	1290	2.1
1	25	3101	4865878	19.2	1024	21	493	3.4
2	26	1808	945465	53.5	1024	23	152	3.7
3	37	812	103412	81.5	1024	25	73	3.7
4	72	401	10442	86.8	1024	36	50	3.6
5	148	318	1201	69.8	709	97	113	3.3
6	93	159	140	45.7	131	48	48	2.2
7	18	18	19	17.7	19	2	2	0.16
8	0	0	1	1.0	1	–	–	–
	Solve, 65536 Processes					Interpolation, 65536 Processes		
Level	No. Sends	Elems. Sent	Unknowns	NNZ/row	Active Procs.	No. Sends	Elems. Sent	NNZ/row
0	6	10000	4096000000	7.0	65536	21	1357	2.1
1	26	3122	309040872	19.4	65536	24	536	3.4
2	26	1887	59587160	54.6	65536	25	178	3.7
3	40	826	6337442	85.5	65536	26	89	3.7
4	87	495	583594	99.6	65534	42	73	3.7
5	187	463	57923	97.0	39692	96	140	3.6
6	203	445	6746	86.0	6365	138	153	3.3
7	245	248	842	79.3	832	100	100	2.8
8	125	125	135	59.4	135	64	64	2.6
9	20	20	21	19.5	21	13	13	1.3
10	1	1	2	2.0	2	–	–	–

Table 1: AMG solve and interpolation operator statistics for Intrepid with 1024 and 65536 processes.

appropriate for a multigrid solve cycle. The reason is that once a process does not own any rows on a level (drops out), it quickly moves between levels, going down the grid hierarchy and then back up until the point when the interpolation results in rows on that process. Then, it sits idle until the the other processes catch up to it. This idle time, which is essentially the sum of the time spent on the level where the process drops out and all levels below it, is reported as being spent in just the level where the process drops out, and so the maximum time spent in each level gives times that are far too large on coarse grids.

Our test problem is a 3D 7-point Laplace problem on a cube, which was also considered in [2]. On Intrepid, Jaguar and Hera, the problem size per core is $50 \times 50 \times 25$. The problem is solved using 128, 1024, 8192 and 65,536 cores on Intrepid and Jaguar, and 128, 1024 and 3456 cores on Hera. On Zeus and Atlas, fewer cores were available to us, so we solve the same problem on 512 cores using $50 \times 50 \times 50$ variables per core. We additionally run the problem on 1728 cores on Atlas. Table 1 shows problem sizes and communication information for the runs on Intrepid. Statistics for the other architectures are similar and hence omitted. When viewing the data in Table 1, there are several things to note. First, one can observe that as the level number increases, the grid becomes coarser as indicated by the column labeled 'Unknowns'. At some coarser level, processes begin to drop out, as indicated by a decreasing number in the 'Active Procs.' column. After processes begin to drop, we also see a increase in the number of sends, though at this point the number of elements sent in each message is getting smaller ('Elems. Sent' column). In the middle levels we also see the increase in stencil size as indicated by 'NNZ/row', meaning the number of nonzero elements in each row on average. The restriction operator is not shown because, for all our experiments, the restriction operator is the transpose of the interpolation operator.

The mappings of MPI processes to nodes used were the defaults on each machine. On Intrepid, this is a block mapping, where each node is filled with successive MPI ranks before assigning processes to the next one. Each job is also guaranteed a contiguous piece of the interconnect. On Hera, Zeus, and Atlas, the positions of the nodes on the interconnect varies from job to job, and the mapping is either block or cyclic, with the choice left to the scheduler. In a cyclic mapping, successive MPI ranks are assigned to different nodes, until each node has one. Then the next task is assigned to the first node, and the process repeated until all the processes are assigned. On Jaguar, information about which nodes the scheduler allocates and how MPI processes are mapped to them is proprietary and thus unavailable to us.

5.3 Machine Parameters

We use benchmark measurements to obtain values for machine parameters. Parameters α and β are determined from best-case latency and bandwidth measurements taken by the latency-bandwidth benchmark in the HPC Challenge suite [4]. Parameter γ is determined as follows. We start with the formulation of α in the distance penalty model, written as a function of the number of hops h:

$$\alpha(h) = \alpha(h_m) + \gamma(h - h_m).$$

Here, $\alpha(h_m)$ is the latency for the shortest possible message distance, corresponding with the minimum latency number reported in the benchmark results. The maximum latency possible is

$$\alpha(D) = \alpha(h_m) + \gamma(D - h_m),$$

where D is the diameter of the network. Taking the

	Intrepid	Jaguar	Hera	Zeus	Atlas
α	3.42 μs	6.05 μs	1.31 μs	0.583 μs	4.62 μs
β	19.3 ns	4.47 ns	6.08 ns	5.80 ns	7.29 ns
γ	28.5 ns	39.9 ns	2.68 μs	3.04 μs	0.88 μs
t_0	27.4 ns	3.27 ns	5.12 ns	4.67 ns	6.22 ns
t_1	12.8 ns	1.18 ns	1.39 ns	1.45 ns	3.22 ns
t_2	7.66 ns	0.935 ns	1.09 ns	1.45 ns	2.23 ns

Table 2: Machine parameters for the architectures evaluated.

maximum latency reported in the benchmark results to be $\alpha(D)$, we have

$$\gamma = \frac{\alpha(D) - \alpha(h_m)}{D - h_m}.$$

The computation rates t_i are measured using a serial sparse MatVec benchmark [9] run on one node, simultaneously on the number of cores per node used in our experiments to properly stress the memory system. Specific values are obtained for the first three levels (t_0, t_1, and t_2), and the value obtained for t_2 is used to approximate the computation rate on all coarser levels. The values are determined from the observed computation rate for sparse MatVec problems matching the dimension and number of nonzero entries per row of the solve operators for the respective levels. Values for all parameters appear in Table 2.

5.4 Validations Results

We apply the possible penalties outlined in the previous section to the basic performance model and show results for the following six combinations. Recall that the penalties in options 2 through 6 are all applied to the baseline model and note that in parentheses are the corresponding legend entries for the plots that follow.

1. Baseline model *(α-β Model)*
2. Baseline plus distance penalty *(α-β-γ Model)*
3. Baseline plus distance penalty and bandwidth penalty on β *(β Penalty)*
4. Baseline plus distance penalty, bandwidth penalty on β, and multicore penalty on α *(α, β Penalties)*
5. Baseline plus distance penalty, bandwidth penalty on β, and multicore penalty on γ *(β, γ Penalties)*
6. Baseline plus distance penalty, bandwidth penalty on β, and multicore penalty on α and γ *(α, β, γ Penalties)*

We model AMG using the models above and contrast their performance. In the following graphs, the best fit option is shown as a solid line in the plots, while the others are shown as lighter weight dotted lines. The actual measured performance is shown as a black line. The coarsest grid, which is solved using Gaussian Elimination instead of smoothing, is not shown. The results are shown in Figure 3 (Intrepid), Figure 4 (Jaguar), Figure 5 (Hera, Zeus), and Figure 6 (Atlas).

On Intrepid, once there is little computation, the performance generally tracks the communication counts, with little overall degradation. The best fit is obtained by the β penalty model, so distance effects and the bandwidth penalty play a role as well. The total cycle time is on average predicted with 71% accuracy. Most of this error is due to the MatVec benchmark mispredicting the computation rate on the finest level. The predicted cycle time without this level has an average accuracy of 94%.

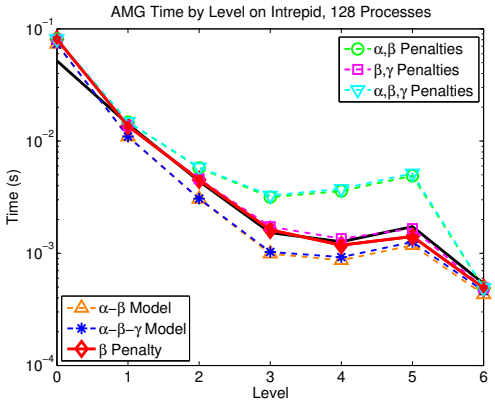
On Jaguar, the other machine with a torus topology, our results are different; the models with penalties to α do the best job of tracking the actual performance, with the α, β, γ penalty model doing the best job overall, though there is some deviation. This deviation can be partially attributed to the MatVec benchmark underestimating the time per flop on the machine. However, the fit that the models provide, with an average cycle time prediction accuracy of 80%, is still sufficient to suggest that endpoint contention is primarily responsible for the performance problems here.

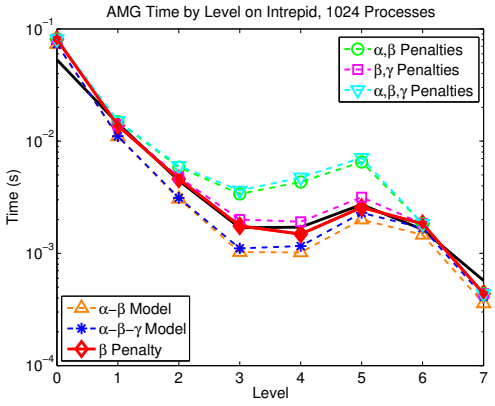
The performance on Hera and Zeus is best tracked by models with a γ penalty, all of which are very close to each other. The best fit comes from the α, β, γ penalty model, with average cycle time prediction accuracies of 82% and 98% for Hera and Zeus, respectively. On Atlas, the best fit is the β, γ penalty model, with an average cycle time prediction accuracy of 87%, though the model does not do an ideal job of tracking the level-by-level performance. Note that Atlas has a much higher latency than Hera and Zeus, and the models with an α penalty substantially underpredict performance on Atlas. The common theme for these three machines, though, is that the best-fit models all have penalties to γ. As these machines have fat-tree interconnects with large switching costs, this indicates that the problem lies with messages contending for the switches.

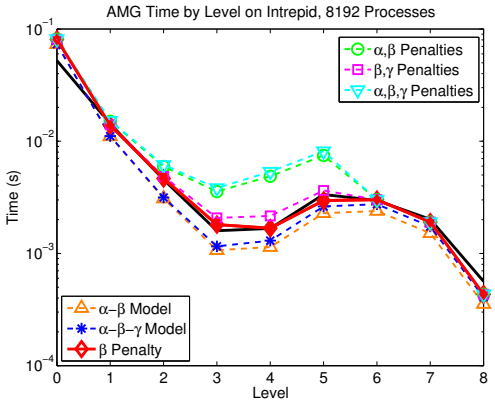
6. LESSONS ON SCALABILITY

Our performance model has significantly contributed to our understanding of the scalability of AMG for large parallel machines. In particular, both distance of communication and contention among multicore nodes are major factors in observed performance and including their effects into our AMG model was essential to achieve a good fit. In this section, we discuss the important lessons learned for ensuring the scalability of AMG (and other HPC applications), both for the design of HPC architectures and for AMG algorithms themselves.

On the architectural side, interconnects must be able to handle both distance and contention effectively. This problem was most noticeable on fat-tree machines where communication distance was a big factor in performance, with γ larger than α or not far from it in magnitude on each platform. Most of this cost came from the top-level switch, as the value of α reflected the cost of communication through one of the lower-level switches. In addition, there was contention in the switches, reflected by the models with penalties to γ giving the best approximation to the performance on these machines. Future machines with the fat-tree topology will need switches that can handle growing amounts of traffic with smaller and smaller delays. Furthermore, machines with a torus interconnect are not immune from penalties to distance or contention either, as the results on Jaguar showed. The main factor was contention among the cores, as shown in the effectiveness of the penalty to α. As is the case with the fat-tree machines, future machines with a mesh or torus topology will also need to be able to handle increasing communication distances and contention among the cores on each node.







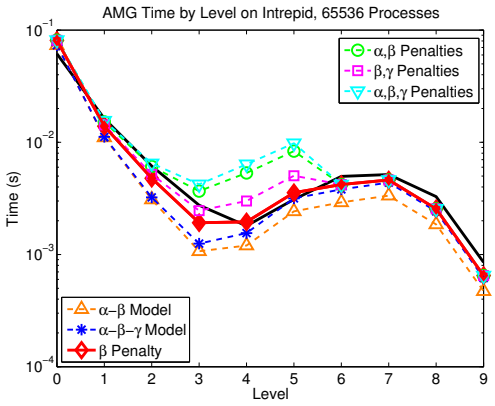
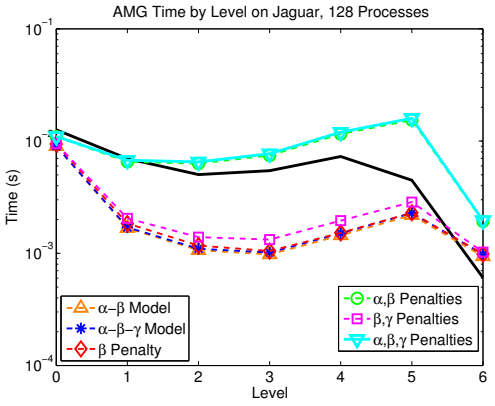
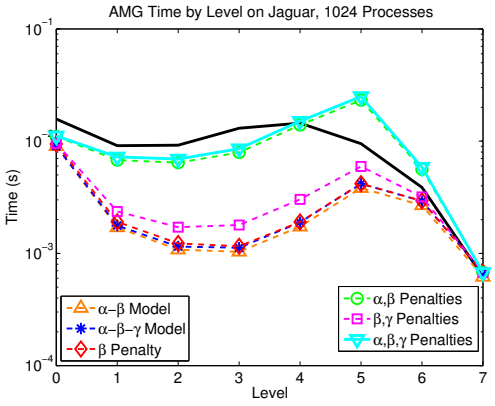
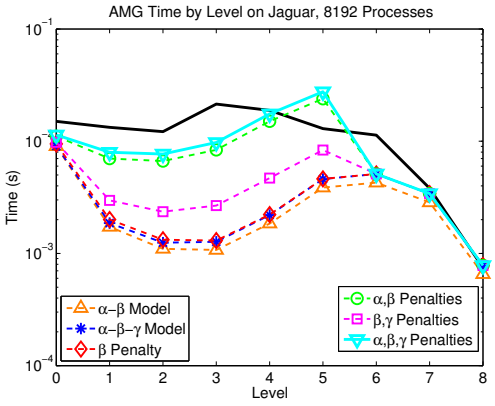


Figure 3: Performance model results on Intrepid.







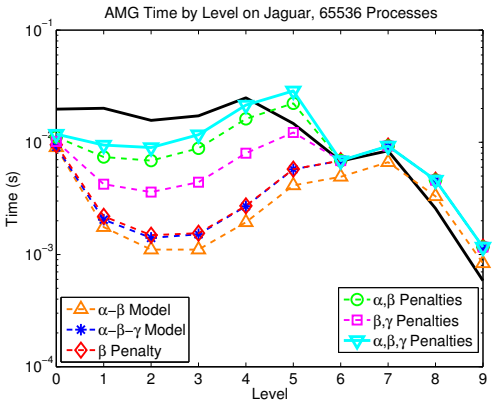


Figure 4: Performance model results on Jaguar.

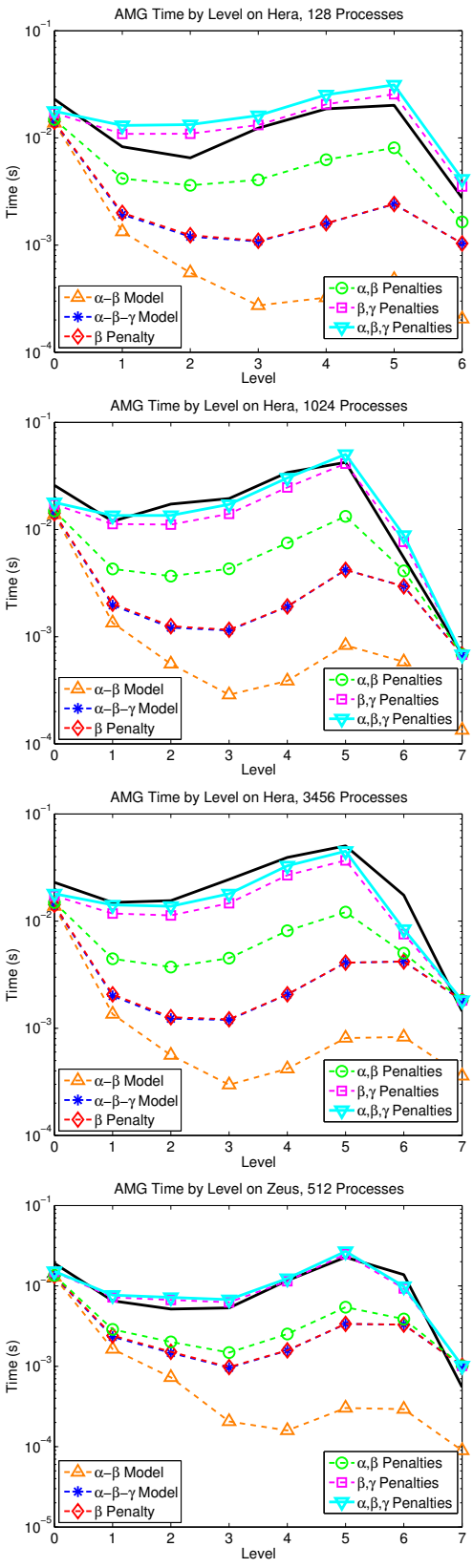


Figure 5: Performance model results on Hera and Zeus.

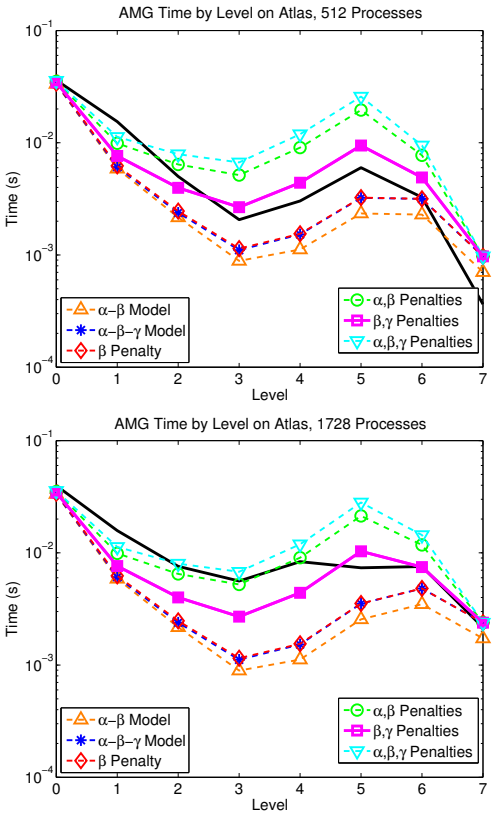


Figure 6: Performance model results on Atlas.

It is likely, though, that improvements in interconnection networks will not be enough to ensure the scalability of AMG in the future. Increasing numbers of cores and cores per node in future parallel machines will make the problems of distance and contention even greater. AMG itself will have to adapt to the changing landscape through algorithmic changes to better handle its performance problems on coarse grids. These changes will have to reduce the amount of communication, whether by trading it for redundant computation, or by using new coarsening and interpolation schemes to create operators that involve less communication.

7. CONCLUSIONS

Motivated by a desire to understand the issues impeding the scalability of AMG on HPC platforms, we developed a performance model for the AMG solve cycle, with penalties to account for various performance problems: distance of communication, low effective bandwidth, and contention among the multiple cores on each node. Our results found that distance and contention were both substantial performance bottlenecks for AMG. With core counts soon to reach the millions, and eventually the billions in future exascale machines, we expect these bottlenecks to become more severe unless changes are made to AMG to alleviate the extensive communication requirements on the coarser grid levels.

In the future, we will use the models developed in this work to guide changes to the AMG solve cycle. In particular, we will reduce AMG's communication burden and explore possibilities that include the use of redundant computation in place of communication and algorithmic changes that re-

duce the amount of required data movements. We will also investigate and model the setup phase on AMG, looking for ways to reduce its communication burden as well. Other issues of interest are the impact of data movement through the memory hierarchy, using threads, different sparse storage formats, and the influence of the mapping of tasks to nodes in the machine. Our ultimate goal is to ensure the scalability of AMG for the exascale machines of tomorrow.

Acknowledgments

We thank the reviewers for their constructive and very helpful comments. This work was supported in part by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under award DE-FG02-08ER25835, and in part by the National Science Foundation award 0837719. Part of this work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344 (LLNL-CONF-473462). 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.

8. REFERENCES

[1] A. H. Baker, T. Gamblin, M. Schulz, and U. M. Yang. Challenges of Scaling Algebraic Multigrid across Modern Multicore Architectures. In *25th IEEE Parallel and Distributed Processing Symposium*, Anchorage, AK, May 2011.

[2] A. H. Baker, M. Schulz, and U. M. Yang. On the Performance of an Algebraic Multigrid Solver on Multicore Clusters. In *VECPAR'10: 9th International Meeting on High Performance Computing for Computational Science*, Berkeley, CA, June 2010.

[3] B. Barnes, B. Rountree, D. Lowenthal, J. Reeves, B. R. de Supinski, and M. Schulz. A Regression-Based Approach to Scalability Prediction. June 2008.

[4] J. Dongarra and P. Luszczek. Introduction to the HPCChallenge Benchmark Suite. Technical Report ICL-UT-05-01, University of Tennessee, Knoxville, March 2005.

[5] C. C. Douglas, J. Hu, M. Kowarschik, U. Rüde, and C. Weiss. Cache Optimization for Structured and Unstructured Grid Multigrid. *Electronic Transactions on Numerical Analysis*, 10:21–40, 2000.

[6] R. D. Falgout, J. E. Jones, and U. M. Yang. Pursuing Scalability for *hypre*'s Conceptual Interfaces. *ACM Transactions on Mathematical Software*, 31:326–350, September 2005.

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

[8] H. Gahvari and W. Gropp. An Introductory Exascale Feasibility Study for FFTs and Multigrid. In *24th IEEE International Parallel and Distributed Processing Symposium*, Atlanta, GA, April 2010.

[9] H. Gahvari, M. Hoemmen, J. Demmel, and K. Yelick. Benchmarking Sparse Matrix-Vector Multiply in Five Minutes. In *SPEC Benchmark Workshop 2007*, Austin, TX, January 2007.

[10] 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.

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

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

[13] D. Kerbyson, H. Alme, A. Hoisie, F. Petrini, A. Wasserman, and M. Gittings. Predictive performance and scalability modeling of a large-scale application. Nov. 2001.

[14] S. S. Shende and A. D. Malony. The TAU Parallel Performance System. *International Journal of High Performance Computing Applications*, 20:287–311, May 2006.

[15] A. Snavely, N. Wolter, and L. Carrington. Modeling application performance by convolving machine signatures with application profiles. In *IEEE Workshop on Workload Characterization, 2001.*, December 2001.

[16] H. D. Sterck, R. D. Falgout, J. W. Nolting, and U. M. Yang. Distance-two interpolation for parallel algebraic multigrid. *Numerical Linear Algebra With Applications*, 15:115–139, April 2008.

[17] H. D. Sterck, U. M. Yang, and J. J. Heys. Reducing complexity in parallel algebraic multigrid preconditioners. *SIAM Journal on Matrix Analysis and Applications*, 27:1019–1039, 2006.

[18] K. Stüben. An introduction to algebraic multigrid. In U. Trottenberg, C. Oosterlee, and A. Schüller, editors, *Multigrid*, pages 413–528. Academic Press, San Diego, CA, 2001.

[19] U. M. Yang. On long-range interpolation operators for aggressive coarsening. *Numerical Linear Algebra With Applications*, 17:453–472, April 2010.