Multigrid on Massively Parallel Architectures*

Robert D. Falgout and Jim E. Jones

Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, P.O. Box 808, L-561, Livermore, CA 94551, USA

Abstract. The scalable implementation of multigrid methods for machines with several thousands of processors is investigated. Parallel performance models are presented for three different structured-grid multigrid algorithms, and a description is given of how these models can be used to guide implementation. Potential pitfalls are illustrated when moving from moderate-sized parallelism to large-scale parallelism, and results are given from existing multigrid codes to support the discussion. Finally, the use of mixed programming models is investigated for multigrid codes on clusters of SMPs.

1 Introduction

Computer simulations play an increasingly important role in scientific investigations. As a result, codes are being developed to solve complex multi-physics problems at very high resolutions. Such large-scale simulations require massively parallel computing, but this is not sufficient. One also needs scalable algorithms such as multigrid, and scalable implementations of these algorithms.

The development of scalable linear solver algorithms is difficult, and currently an active area of research in the numerical analysis community. These solvers must be robust and have optimal computational complexity, but they must also exhibit enough concurrency to be effectively parallelized. For example, in Algebraic Multigrid, the coarsening procedure is inherently sequential, so new parallel algorithms are needed. Once a scalable algorithm has been developed, it is usually fairly straightforward (using a *domain partitioning* approach) to write an effective parallel implementation for machines with a few hundred or so processors. However, for machines with several thousands of processors, scalable implementations are more challenging to achieve.

In Section 2 we present performance models for three different multigrid algorithms. In Section 3 we describe how these models can be used to guide the implementation of parallel multigrid codes, illustrating some potential pitfalls when moving from moderate-sized parallelism to large-scale parallelism. Supporting results from existing multigrid codes are given.

* This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract no. W-7405-Eng-48.

2 Parallel Performance Models

Consider a 3D problem, discretized on a logically rectangular grid Ω^0, and assume that the problem is distributed in a domain-partitioned manner across a logical $p \times p \times p$ process grid. Assume that each process q holds an $n \times n \times n$ subgrid Ω_q^0, so that the total problem size is $N = (pn)^3$, and the total number of processes is $P = p^3$.

We use the same performance model analyzed in [2,3]. In the model, we assume that the time to access m doubles from non-local memory is

$$\alpha + \beta m,$$

and the time to perform a floating-point operation is γ. We will use the following values for α, β, and γ

$$\alpha = 230\mu\text{sec}, \quad \beta = .16\mu\text{sec/double}, \quad \gamma = .074\mu\text{sec/flop},$$

meant to model an early IBM SP, but chosen mainly for consistency with [2].

In all of the performance models presented below, we consider only the cost of relaxation. Also, the distribution of the coarse grids Ω^l $(l \geq 1)$ is assumed to be the domain partitioning naturally induced by that of the fine grid so that $\Omega_q^l \subset \Omega_q^0$. Note that, as a result, processes will become "idle" on the coarsest grids. See [4] for a discussion of this issue.

2.1 PFMG Model

In this section, we present the performance model for a semicoarsening multigrid algorithm, denoted PFMG (for historical reasons; it is similar to the method described in [1]). This method uses semicoarsening to handle problem anisotropies, and works well when the anisotropies are uniform and grid-aligned throughout the domain. The relaxation method used is point Jacobi.

To be precise, consider solving the PDE

$$-u_{xx} - \varepsilon_1 u_{yy} - \varepsilon_2 u_{zz} = f, \quad 0 < \varepsilon_2 \leq \varepsilon_1 \leq 1, \tag{1}$$

on the unit square. Assume a finite difference discretization on a uniform grid with grid spacing h, so that the fine-grid operator is a 7-point operator.

The coarse grids are defined by first coarsening in the x-direction (by a factor of 2) until the anisotropy in the xy-plane is made as small as possible. That is, we coarsen c_1 times until

$$2^{c_1} h \approx \varepsilon_1^{-1/2} h.$$

Next, coarsening is done in both the x- and y-directions until anisotropy in both the xz- and yz-planes is made as small as possible. That is, we coarsen c_2 times until

$$2^{c_1+c_2} h \approx \varepsilon_2^{-1/2} h.$$

Finally, coarsening is done in all directions until some sufficiently small coarse grid (e.g., a single point).

The coarse grid operators are formed by the Galerkin process, where interpolation is taken to be bi-linear interpolation. Hence, given a 7-point fine-grid operator, the first c_1 coarse-grid operators are 15-point, and the remaining operators are 27-point.

The time for doing relaxation in a $V(1,0)$-cycle is given by

$$T = K_\alpha \alpha + K_\beta n^2 \beta + K_\gamma n^3 \gamma, \tag{2}$$

where

$$\begin{aligned}
K_\alpha &\approx 6 + 14c_1 + 26(c_2 + L - 1), \\
K_\beta &\approx 12 + 2c_1 - (4/3)(2^{-c_1} + 2^{-c_2} + 2^{-c_1}2^{-c_2}), \\
K_\gamma &\approx 22 - 6(2^{-c_1}) - (36/7)(2^{-c_1}4^{-c_2}),
\end{aligned}$$

and where $L \approx \log_2(pn)$. Estimating the first term of (2) involves counting the number of messages sent in relaxation. Each process must communicate with its neighbors on each grid level, and the number of neighbors depends on the stencil size. Estimating the second term of (2) involves computing the size of each plane of data communicated to neighboring processes. Thus,

$$\begin{aligned}
K_\beta \approx 6 + 2\sum_{l=1}^{c_1}(1 + 2^{-l} + 2^{-l}) &+ \\
2\sum_{l=1}^{c_2}(2^{-l} + 2^{-c_1} \cdot 2^{-l} + 2^{-c_1} \cdot 4^{-l}) &+ \\
2\sum_{l=1}^{\infty}(2^{-c_2} \cdot 4^{-l} + 2^{-c_1}2^{-c_2} \cdot 4^{-l} + 2^{-c_1}4^{-c_2} \cdot 4^{-l})& \\
= 6 + 2c_1 + 4(1 - 2^{-c_1}) &+ \\
2(1 + 2^{-c_1})(1 - 2^{-c_2}) + (2/3)(2^{-c_1})(1 - 4^{-c_2}) &+ \\
(2/3)(2^{-c_2} + 2^{-c_1}2^{-c_2} + 2^{-c_1}4^{-c_2}).&
\end{aligned}$$

Estimating the third term of (2) involves computing the number of flops done in relaxation, which is approximated by the stencil size times the number of grid points on a process. Thus,

$$\begin{aligned}
K_\gamma &\approx 7 + 15\sum_{l=1}^{c_1} 2^{-l} + 27(2^{-c_1})\sum_{l=1}^{c_2} 4^{-l} + 27(2^{-c_1}4^{-c_2})\sum_{l=1}^{\infty} 8^{-l} \\
&= 7 + 15(1 - 2^{-c_1}) + 9(2^{-c_1})(1 - 4^{-c_2}) + (27/7)(2^{-c_1}4^{-c_2}).
\end{aligned}$$

2.2 SMG Model

In this section, we present the performance model for another semicoarsening multigrid algorithm, denoted SMG. The algorithm was introduced in [5], and its model was derived in [2]. SMG is a particularly robust method, designed primarily to solve the diffusion equation

$$-\nabla \cdot (D \nabla u) + \sigma u = f \tag{3}$$

on logically rectangular grids, where the diffusion coefficient, D, is a $d \times d$ matrix in d dimensions, and $\sigma \geq 0$.

The main difference between SMG and PFMG is the smoother. The SMG algorithm semicoarsens in the z-direction and uses plane smoothing. The xy plane-solves are effected by one V-cycle of the 2D SMG algorithm, which semicoarsens in y and uses line smoothing.

The time for doing relaxation in a $V(1,0)$-cycle is given by

$$T \approx (4L + 8L^2 + 8L^3)\alpha + 20Ln^2\beta + 48n^3\gamma, \tag{4}$$

where $L \approx \log_2(pn)$.

2.3 MG Model

In this section, we present the performance model for a full-coarsening multigrid algorithm, denoted MG. Here, the coarse grids are defined by coarsening by a factor of 2 in all directions, and the relaxation method is point Jacobi. The fine-grid and coarse-grid operators are all assumed to be 7-point.

The time for doing relaxation in a $V(1,0)$-cycle is then given by

$$\begin{aligned} T &= 6L\alpha + 6n^2\beta \sum_{l=0}^{L-1} 4^{-l} + 7n^3\gamma \sum_{l=0}^{L-1} 8^{-l} \\ &\approx 6L\alpha + 8n^2\beta + 8n^3\gamma, \end{aligned} \tag{5}$$

where $L \approx \log_2(pn)$ represents the number of grid levels.

3 Model-Guided Implementation

Parallel performance models such as the three presented in Section 2 can be used to great advantage when implementing a multigrid algorithm. This is particularly true when the target platform is a massively parallel computer with upwards of ten thousand processors. In this section, we consider three main implementation issues, primarily in the context of developing structured multigrid codes.

To set the stage, consider the implementation of a parallel library of sparse linear solvers, to be interfaced with parallel multi-physics codes to solve the linear systems that arise from finite difference, finite volume, or finite element discretizations of PDEs on logically rectangular grids. We assume that the problem data has already been distributed, and is given to the solver library in this distributed form. We also assume that the distribution represents a partitioning of the domain into roughly equal-sized rectangular subdomains with minimal surface-to-volume ratios. In particular, assume that it is not advantageous to redistribute the data. These assumptions will generally hold for multi-physics codes designed to run on large-scale parallel computers, as well as a large number of codes designed for smaller-scale parallel computers.

In order to use the performance models derived earlier, much of the discussion in this section will be centered around problems that use the same data-distribution given in the models. However, this is only to simplify the presentation. It is important to keep in mind the more general library setting described above.

3.1 Replicated Computations

Given a description of the fine grid and its distribution, consider the computation of the coarse grids and their distributions. In the library setting assumed here, a grid and its distribution can be represented by a list of *boxes* and corresponding process numbers, where a box is defined to be a pair of indexes in the 3D index-space,

$$\mathcal{I} = \{(i, j, k) : i, j, k \text{ integers}\}.$$

That is, a box represents the "lower" and "upper" corner points of a subgrid via the indices $(i_l, j_l, k_l) \in \mathcal{I}$ and $(i_u, j_u, k_u) \in \mathcal{I}$.

On each process q, the full description of each grid's distribution is not needed, only the description of the subgrid Ω_q^l and its "nearest" neighboring subgrids. However, to compute this on all grid levels requires that at least one of the processes—one containing a nonempty subgrid of the coarsest grid—has information about the coarsening of every other subgrid. That is, for at least one process, computing the coarsening information requires that all of the subgrids on the fine grid be visited at least once. We will consider two approaches for coarsening, denoted **A1** and **A2**.

In **A1**, each process q coarsens subgrid Ω_q^l and receives neighbor information from other processes. This requires $O(1)$ computations and $O(\log_2 N)$ communications. In **A2**, the coarsening procedure is replicated on all processes, which requires $O(P)$ computations and no communications. This latter approach works well for moderate numbers of processors, but becomes prohibitive for large P.

To see this, we can use the models presented in Section 2. The dominant cost in **A1** is communication latency, and this is estimated by the α-term in each model. The cost of **A2** is just the cost of coarsening a subgrid, times the number of subgrids, times the number of multigrid levels. We want to find $\hat{P}$, such that for $P > \hat{P}$, **A2** is less efficient than **A1**. For the MG algorithm, we set

$$6L\alpha = LPf_s\gamma,$$

which yields

$$\hat{P} = (6/f_s)(\alpha/\gamma), \tag{6}$$

where f_s is the number of flops required to coarsen a subgrid. Letting $f_s = 90$ (this is representative of what appears in the PFMG code mentioned below), we have that $\hat{P} \approx 207$. For the PFMG algorithm, we set

$$K_\alpha \alpha = (c_1 + c_2 + L) P f_s \gamma,$$

which yields a $\hat{P}$ that depends on n, c_1, and c_2. However, we can bound $\hat{P}$ as follows

$$(6/f_s)(\alpha/\gamma) \leq \hat{P} \leq (26/f_s)(\alpha/\gamma). \tag{7}$$

In the case of an isotropic problem, the smoothing cost per V-cycle for the PFMG algorithm is the same as for MG, hence the lower bound in (7). The upper bound is roughly a factor of four larger, so that $\hat{P} \approx 898$ for the parameters being considered here. Note from (6) and (7) that $\hat{P}$ depends strongly on the ratio of communication latency to computation speed.

This analysis also bears out in practice. In Figure 1, we present results from an MPI-implementation of PFMG run on an Intel Paragon. The problem solved was the anisotropic diffusion problem (1) with $n = 40$, $\varepsilon_1 = 1/10$, and $\varepsilon_2 = 1/100$. The figure compares the cost of coarsening using approach **A2** (labeled "Coarsen") with the cost of a V-cycle. The time for **A2** was not computed directly, but estimated by taking the overall setup time, and subtracting the setup time for the single processor run. The figure suggests that the cost of replicating the grid coarsening procedure is greater than the cost of a $V(1,1)$ cycle when P is larger than about 500.

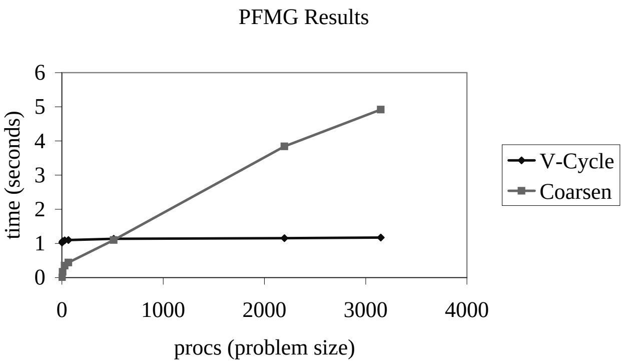

Fig. 1. PFMG results on an Intel Paragon comparing the cost of grid coarsening to the cost of a V-cycle.

3.2 Ghost Zones

The notion of *ghost zones* or *shadow zones* is commonly used in parallel linear solver codes, and is simply the extra "layer" of data needed from off-process to complete an on-process computation. The size of the ghost-zone layer can vary depending on the algorithm implementation. We will consider here the use of a single layer of ghost zones in the library setting described earlier. Figure 2 illustrates (in 2D) the layout of data and ghost zones for two 7×7 subgrids, and shows a typical communication pattern for a 5-point stencil computation. Note that, to simplify code, subgrid data and ghost-zone data are stored together as part of a single array in memory. To reduce the number of copies, this extra ghost-zone memory is always present in the vector data structure (note that ghost-zone memory is usually not persistent in unstructured-grid multigrid codes).

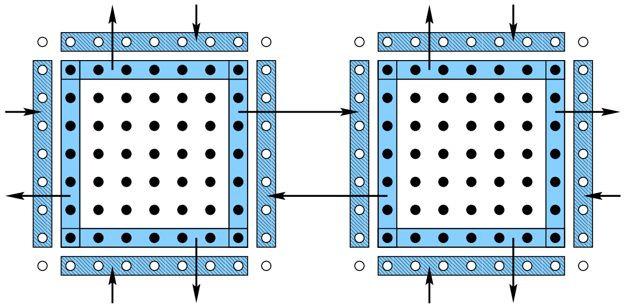

Fig. 2. Ghost zones and communications for a 5-point stencil and 7×7 subgrids.

For the MG and PFMG algorithms, the storage overhead associated with ghost zones is quite acceptable. But, for more robust methods like SMG, ghost zone storage can be problematic. To see this, we can again use the models presented in Section 2. The coefficient multiplying β in each model also estimates the amount of ghost-zone storage used. In Figure 3, we plot this storage cost for the SMG algorithm relative to n^3, the cost of storing a vector. We see that the ghost-zone overhead is quite high, but we also note that the growth rate is moderate. That is, a $\log_2 N$ dependence of the β-term in a model does not necessarily produce a ghost-zone memory overhead problem. For example, consider using alternating line relaxation in a full-coarsening multigrid method. Using a similar derivation as for SMG, it is easy to see that the ghost-zone storage cost is approximately $6Ln^2$, or about 30% that of SMG. In comparison, the overhead for PFMG for the problem described in Section 3.1 is about 0.6, and does not grow with P.

3.3 Mixed Programming Models

There is a recent trend to build large, parallel computers out of commodity parts. The largest such computers are clusters of shared memory processors (SMPs). In this section, we will discuss the use of mixed programming models for implementing parallel multigrid methods.

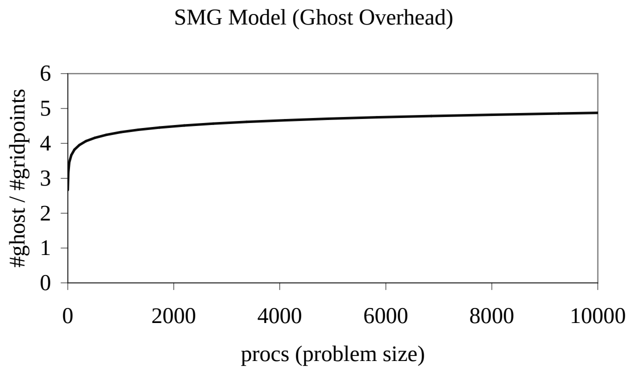

Fig. 3. SMG model illustrating relative storage costs of ghost zones.

Figure 4 illustrates two basic approaches for distributing (and computing on) subgrid data on a 4-processor SMP node. Pictured on the left (the *mixed model*) is one large subgrid with ghost layer (for communicating with other SMP nodes) and four regions of data, each assigned to different threads (these will usually be run on different processors). Pictured on the right (the *message-passing model*) are four subgrids with ghost layers, each subgrid assigned to different processes (again, these will usually be run on different processors).

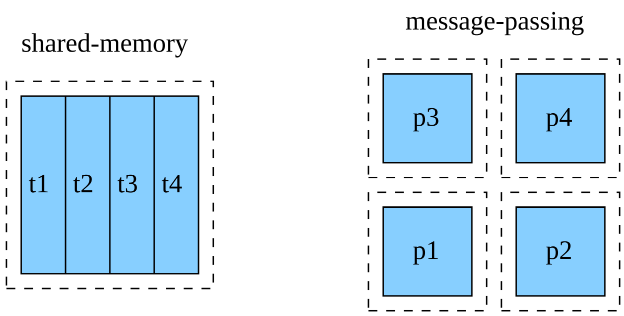

Fig. 4. Schematic of mixed model and message-passing model for a single 4-processor SMP node.

In theory, the mixed model has a couple of advantages over the pure message-passing model. The first advantage is a reduction in the number of messages going in and out of the SMP node. For example, for a 5-point stencil computation, the mixed model depicted in the figure requires 4 communications outside of the SMP and the message-passing model requires 8. The second advantage is the ghost-zone memory savings due to the fewer and larger subgrids in the mixed model. In the figure, the ghost-zone memory savings is a factor of two. On SMPs with larger numbers of processors, the memory savings can be even more substantial.

Although the mixed model has these attractive features, our efforts to outperform the message-passing model have not yet succeeded in practice. We have developed two implementations of the mixed programming model, using MPI to do the message-passing in both cases. The first implementation uses POSIX threads, but we will discuss here only the second implementation, which uses OpenMP compiler directives. The approach taken was straightforward loop-level parallelism of the computational kernels in the code. Each of the kernels is a triply-nested loop over data associated with a subgrid. The OpenMP directives can only parallelize a single loop, so effective parallelism can only be achieved when the size of this loop is at least as large as the number of processors. Since multigrid methods—especially semicoarsening methods—produce grids of varying shapes and sizes, a fourth outer loop was added that explicitly decomposes the subgrid into roughly equal sized regions to be assigned to the different threads.

To be clear, consider the 2D example pictured on the left in Figure 4. Here, we have an outer loop as just described, but with only a doubly-nested inner loop. The outer loop has length four, and on each iteration, the inner loops iterate over the tall rectangular regions. If the outer loop is threaded using OpenMP, this means that each iteration is assigned to a different thread. Hence, the computations on each region in the figure are handled by different processors. The decomposition of the subgrid is done by simply subdividing the largest subgrid dimension by the number of threads being used.

In Figure 5, we show results comparing the MPI implementation to the mixed MPI-OpenMP implementation for conjugate gradient (CG) and CG with three different preconditioners: SMG, PFMG, and diagonal-scaling. We plot MPI time over MPI-OpenMP time. The MPI implementation is fastest in all cases.

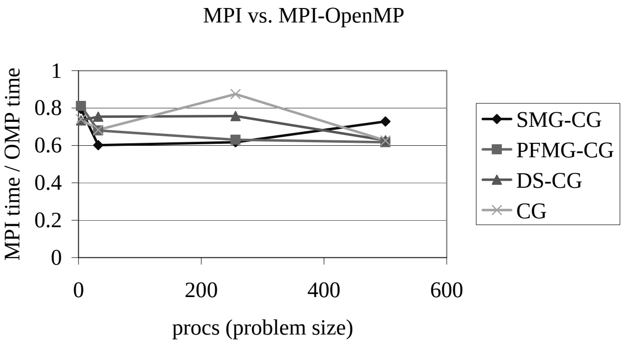

Fig. 5. Comparison of MPI and mixed MPI-OpenMP implementations of various solvers on an IBM SP2.

4 Conclusions

Extra care must be taken when developing codes for large-scale parallel architectures. Techniques commonly used for moderate-sized parallelism can be problematic for large-scale parallelism. Parallel performance models can provide useful implementation guidance, especially regarding the tradeoffs of replicating computations in order to reduce communications. On clusters of SMPs, mixed programming models have several advantages over straight message-passing, but these advantages are not yet born out in practice.

References

1. S. F. Ashby and R. D. Falgout. A parallel multigrid preconditioned conjugate gradient algorithm for groundwater flow simulations. *Nuclear Science and Engineering*, 124(1):145–159, September 1996. Also available as LLNL Technical Report UCRL-JC-122359.
2. P. N. Brown, R. D. Falgout, and J. E. Jones. Semicoarsening multigrid on distributed memory machines. To appear in the SIAM Journal on Scientific Computing special issue on the Fifth Copper Mountain Conference on Iterative Methods. Also available as LLNL technical report UCRL-JC-130720, 1999.
3. W. D. Gropp and D. E. Keyes. Complexity of parallel implementation of domain decomposition techniques for elliptic partial differential equations. *SIAM J. Sci. Stat. Comput.*, 9:312–326, 1988.
4. J. E. Jones and S. F. McCormick. Parallel multigrid methods. In Keyes, Sameh, and Venkatakrishnan, editors, *Parallel Numerical Algorithms*, pages 203–224. Kluwer Academic, 1997.
5. S. Schaffer. A semi-coarsening multigrid method for elliptic partial differential equations with highly discontinuous and anisotropic coefficients. *SIAM J. Sci. Comput.*, 20(1):228–242, 1998.