Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Computational Physics 188 (2003) 593–610
JOURNAL OF COMPUTATIONAL PHYSICS
www.elsevier.com/locate/jcp

Parallel multigrid smoothing: polynomial versus Gauss–Seidel

Mark Adams [a,1], Marian Brezina [b,2], Jonathan Hu [a,1], Ray Tuminaro [a,*,1]

[a] *Department of Computational Mathematics and Algorithms, Sandia National Laboratories, P.O. Box 969, MS 9217, Livermore, CA 94551, USA*
[b] *Department of Applied Mathematics, University of Colorado at Boulder, Campus Box 526, Boulder, CO 80309, USA*

Received 22 October 2002; received in revised form 3 March 2003; accepted 17 March 2003

Abstract

Gauss–Seidel is often the smoother of choice within multigrid applications. In the context of unstructured meshes, however, maintaining good parallel efficiency is difficult with multiplicative iterative methods such as Gauss–Seidel. This leads us to consider alternative smoothers. We discuss the computational advantages of polynomial smoothers within parallel multigrid algorithms for positive definite symmetric systems. Two particular polynomials are considered: Chebyshev and a multilevel specific polynomial. The advantages of polynomial smoothing over traditional smoothers such as Gauss–Seidel are illustrated on several applications: Poisson's equation, thin-body elasticity, and eddy current approximations to Maxwell's equations. While parallelizing the Gauss–Seidel method typically involves a compromise between a scalable convergence rate and maintaining high flop rates, polynomial smoothers achieve parallel scalable multigrid convergence rates without sacrificing flop rates. We show that, although parallel computers are the main motivation, polynomial smoothers are often surprisingly competitive with Gauss–Seidel smoothers on serial machines.

AMS: 76D05; 76D07; 65F10; 65F30

Keywords: Multigrid; Gauss–Seidel; Polynomial iteration; Smoothers; Parallel computing

1. Introduction

Multigrid methods (e.g. [9,11,20]) are among the most efficient iterative algorithms for solving the linear systems associated with elliptic partial differential equations. The basic idea of multigrid is to capture errors

* Corresponding author. Tel.: +1-925-294-2564; fax: +1-925-294-2234.
E-mail addresses: mfadams@ca.sandia.gov (M. Adams), brezina@newton.colorado.edu (M. Brezina), jhu@sandia.gov (J. Hu), tuminaro@ca.sandia.gov, rstumin@sandia.gov (R. Tuminaro).

[1] Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-ACO4-94AL85000.

[2] This work was sponsored by the National Institute of Health under Grant No. 1-R01-EY12291-01, the National Science Foundation under Grant No. DMS-0084438, and the Department of Energy under Grant Nos. DE-FG03-94ER25217 and DE-FC02-01ER25479.

0021-9991/03/$ - see front matter
doi:10.1016/S0021-9991(03)00194-3

by utilizing multiple resolutions in the iterative scheme. High-energy (or oscillatory) components are effectively reduced through a simple smoothing procedure, while the low-energy (or smooth) components are tackled using an auxiliary lower-resolution version of the problem (coarse grid). Since the error after smoothing should lack the high-energy components, it is generally assumed that this error can be well-approximated using a coarser resolution. Thus, the residual equation is transferred to the coarser level, and its solution is used to correct the fine-level solution. The idea is applied recursively on the next coarser level.

The Gauss–Seidel method has long been the smoother of choice within multigrid schemes. It is effective on problems of practical interest, and on model problems its superior smoothing characteristics cost little to implement in the serial computation context [5]. While numerically attractive, constructing efficient parallel true Gauss–Seidel algorithms is challenging as is shown in Sections 2 and 5. As an alternative, Processor Block (or local) Gauss–Seidel is often used. Here, each processor performs Gauss–Seidel as a subdomain solver for a block Jacobi method. While Processor Block Gauss–Seidel is easy to parallelize, the overall multigrid convergence rate usually suffers and can even lead to divergence if not suitably damped, as discussed in Section 2.

Given the parallel Gauss–Seidel difficulties, we consider smoothing schemes based on a polynomial iteration of the form:

$$x^{(m+1)} = x^{(m)} + \sum_{0 \leqslant j \leqslant n} \alpha_j A^j (b - Ax^{(m)}) \tag{1.1}$$

to solve the positive definite symmetric system

$$Ax = b$$

with initial guess $x^{(0)}$. The α_j are the polynomial coefficients and are precomputed based on some knowledge of the spectrum of A. To simplify the description, it is always assumed in this paper that the matrix A has been scaled so that all diagonal entries are one. However, symmetric scaling is easily incorporated into (1.1) and is used within our multigrid codes. The case $n = 0$ corresponds to the well known damped-Jacobi method. Using polynomial smoothers with $n > 0$ as a solver has been suggested in [25]. Polynomial smoothers within multigrid methods have also been known for quite some time. Multi-stage Runge–Kutta methods [16,17,26], for example, enjoy significant success within the computational fluid dynamics community. These methods arise by introducing an artificial time-derivative to the equation obtained by discretization of hyperbolic or almost hyperbolic partial differential equations. For symmetric problems, Chebyshev polynomials have also been known for quite some time. They are discussed in works as early as [5] and [19]. However, these papers recommend Gauss–Seidel methods over polynomial methods. In fact, our central thrust is to dispell a perception that polynomial smoothers are computationally inferior to Gauss–Seidel. This previous perception was supported by comparing convergence rates and floating point operations on simple structured mesh problems. However, the comparisons between Gauss–Seidel and polynomial methods are completely different when unstructured meshes are considered (where multi-color Gauss–Seidel is problematic) on modern CPUs (where multi-color Gauss–Seidel flop rates can be low) within a parallel simulation (where high Gauss–Seidel efficiency may be harder to obtain).

We focus on two particular polynomials as a competitive replacements for a Gauss–Seidel iteration within an unstructured parallel multigrid algorithm. In Section 3, we describe two specific polynomials and motivate why they may be appropriate within a multigrid solver. Section 4 analyses the performance of polynomial and Gauss–Seidel methods. Finally, Section 5 compares the polynomial smoothers with both Parallel Block Multi-colored Gauss–Seidel and processor-based Gauss–Seidel on three different computational problems coming from Poisson's equation, Maxwell's equations, and elasticity. While the primary focus is on parallel performance, surprisingly, the polynomial methods are usually superior with Gauss–Seidel in serial computations as well.

2. Parallel Gauss–Seidel smoothing

The Gauss–Seidel iteration is widely used as a multigrid smoother because it is effective on a variety of common model problems. The Gauss–Seidel method can be succinctly described by

$$x^{(m+1)} = x^{(m)} + L^{-1}(b - Ax^{(m)}),$$

where L is the lower triangular portion of the matrix A. The Gauss–Seidel iteration proceeds by updating one unknown at a time. The kth unknown is modified so that the kth equation is satisfied exactly using the most up-to-date approximation for the other unknowns. The numerical convergence of the Gauss–Seidel method depends on the order in which the unknowns are updated. In particular, red-black ordered Gauss–Seidel can provide for improved convergence rates (see Table 1 for convergence properties of several common smoothers). It should be further noted that the damped version of the Gauss–Seidel method is referred to as successive over relaxation (SOR). In this paper, we consider only Gauss–Seidel since choosing an optimal SOR damping parameter can be difficult for many problems of practical interest and is not generally used in multigrid smoothers.

To employ the method within a conjugate-gradient preconditioner, we use symmetric Gauss–Seidel to maintain symmetry. Symmetric Gauss–Seidel processes the equations in the reverse order of the previous Gauss–Seidel application. It should be noted that there is no convergence benefit to a symmetric red-black Gauss–Seidel *solver* over just a red-black Gauss–Seidel *solver* on a 2-cyclic matrix [12]. Our situation, however, is significantly different in that we use Gauss–Seidel as a *smoother* within a multigrid *preconditioner*. This "symmetric" Gauss–Seidel smoother actually uses a single forward Gauss–Seidel sweep within the pre-smoother and a single backward Gauss–Seidel sweep within the post-smoother. Thus, unlike a symmetric Gauss–Seidel *solver*, a coarse grid correction occurs between the last forward sweep color and the first backward sweep color. Further, our parallel Gauss–Seidel smoother corresponds to a block coloring where lexicographical Gauss–Seidel is employed within the blocks. Finally, symmetric Gauss–Seidel smoothing follows standard practice of maintaining preconditioner symmetry for a symmetric problem thereby allowing conjugate-gradient to be applied.

Though Gauss–Seidel smoothers typically yield good multigrid convergence properties, they lose their luster on unstructured grids and modern parallel computers. Specifically, parallelization can be obtained by identifying groups of unknowns that are independent from each other. Unknowns within a group (or color) can then be updated simultaneously. Unfortunately, the elegant simplicity of structured grid multi-color Gauss–Seidel is lost on 3D unstructured finite element applications as the number of required colors

Table 1
2-Level multigrid on 255×255 structured grid Laplacian with Dirichlet boundary conditions

Smoother	Iterations	Convergence rate
1 Lex. Gauss–Seidel	28	.384
2 Lex. Gauss–Seidels	16	.184
3 Lex. Gauss–Seidels	13	.112
1 Red-black Gauss–Seidel	20	.246
2 Red-black Gauss–Seidels	11	.067
3 Red-black Gauss–Seidels	10	.049
1 Damped Jacobi	53	.596
2 Steps of damped Jacobi	27	.36
2nd order Chebyshev	19	.216
3rd order Chebyshev	13	.120

Convergence determined when the initial residual is reduced by 10^{12}. Convergence rate is the ratio of the last two residuals. Random initial guess and zero right hand side.

increases dramatically. Additionally, multi-color ordering possesses pathological cache behavior as the solution vector is almost completely swept through for each color. To avoid parallelization difficulties, a processor-localized Gauss–Seidel is often employed instead of a true Gauss–Seidel method. In this case, however, multigrid convergence rates usually suffer. The rest of this section discusses the parallel Gauss–Seidel methods that are used in this study: a recent *Parallel Block Multi-color Gauss–Seidel* algorithm (Section 2.1) and *Processor Block Gauss–Seidel* (Section 2.2).

2.1. Parallel block multi-color Gauss–Seidel

We are aware of only one efficient parallel true Gauss–Seidel algorithm for unstructured problems, developed by Adams [1]. This algorithm takes advantage of the domain decomposition provided by the distribution of the stiffness matrix that is common in parallel computing. These domains are generally chosen to reduce communication required by common operations such as residual calculation. If processor subdomains are relatively large, they will contain many *interior* nodes (i.e., nodes that can be processed without communication) relative to the number of *boundary* nodes (i.e., nodes requiring off processor updates in Gauss–Seidel). The general idea is to order nodes such that interior nodes are processed while waiting for communication necessary to process boundary nodes. This algorithm first colors the processors and uses an ordering of the colors to provide a processor inequality operator (this ordering is reversed to symmetrize the algorithm). Each processor partitions its nodes into interior and boundary nodes. Boundary nodes are further partitioned into three sets:

- *Bot*: requires only communication with higher processors,
- *Top*: requires only communication with lower processors,
- *Mid*: all remaining boundary nodes.

Interior nodes are partitioned into two sets: Int_1 and Int_2 so as to satisfy

$$|Int_1| + |Top| \approx |Int_2| + |Bot|, \tag{2.1}$$

where $|\cdot|$ measures the cost to apply Gauss–Seidel to the equations in the set. This cost is approximated by the number of non-zeros in the equations. Int_1 and Int_2 are each partitioned into two additional sets (Int_{1a}, Int_{1b}, Int_{2a} and Int_{2b}) to allow for overlapping computation with the two primary communication steps in this algorithm.

Fig. 1 illustrates the partitions of a 2D, four processor problem where the processor colors are represented with integers (i.e., 1–4). Once nodes are partitioned, the idea of the algorithm is to process first *Top* nodes, then *Mid* nodes, and then *Bot* nodes. While communication that is needed for these boundary nodes occurs, interior nodes are processed. Fig. 2 shows a schematic time line for this algorithm corresponding to the model problem in Fig. 1. The primary communication steps are indicated with arrows. Eq. (2.1) insures that all processors update the *Mid* nodes at roughly the same time (i.e., after half the work has been completed). Note that this is a "soft" synchronization point. In order to use the multigrid method as a preconditioner for a symmetric Krylov method, we need to ensure symmetry of the smoother. To symmetrize the algorithm, the sets Int_1 and Int_2 are interchanged as well as the sets *Top* and *Bot*, and the equations in each partition are processed in reverse order, see [1] for more details.

2.2. Processor block Gauss–Seidel

To avoid complexities associated with parallel Gauss–Seidel method and the soft synchronization points required for any true parallel Gauss–Seidel implementation, it is possible to combine Gauss–Seidel with Jacobi's method. This compromise approach is commonly used in practice, sometimes with interprocessor damping to account for the overall Jacobi character of the method (see for example [13]). Specifically, each

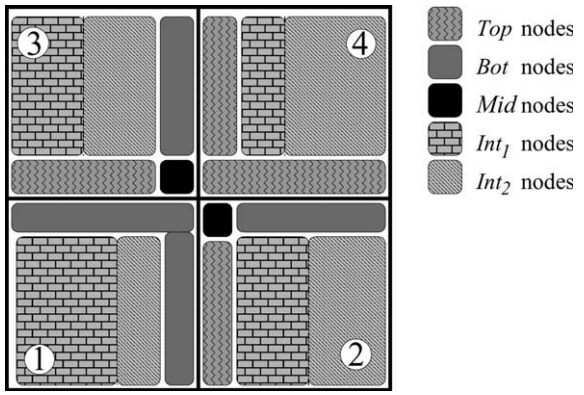

Fig. 1. Sample partitioning for 2D mesh with four processors.

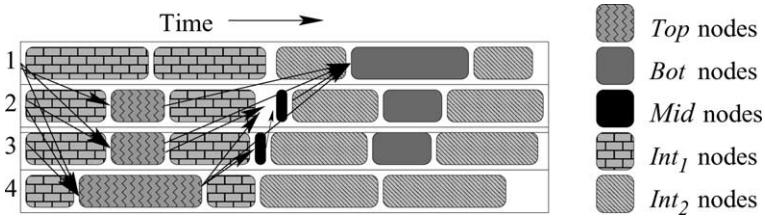

Fig. 2. Time line of model 2D problem.

processor performs a Gauss–Seidel iteration with the exception that previous iteration values are used for all off-processor unknowns (unknowns computed on other processors). We write this Processor Block Gauss–Seidel iteration as

$$x^{(m+1)} = x^{(m)} + block\_diag(L)^{-1}(b - Ax^{(m)}),$$

where

$$block\_diag(L)_{i,j} = \begin{cases} 0 & \text{if } i \text{ and } j \text{ are on different processors,} \\ a_{i,j} & \text{if } i \text{ and } j \text{ are on the same processor.} \end{cases}$$

This can be viewed as an additive Schwarz domain decomposition method where the subdomain solver is one iteration of Gauss–Seidel.

While Processor Block Gauss–Seidel method is perfectly parallel, multigrid convergence is usually slower than with true Gauss–Seidel smoothing. In fact, multigrid might converge acceptably with Gauss–Seidel smoothing but diverge dramatically with Processor Block Gauss–Seidel. One transparent example of this phenomenon occurs when a single unknown is assigned to each processor. In this case, Processor Block Gauss–Seidel is equivalent to undamped Jacobi which is unsuitable as a multigrid smoother. While dramatic divergence might seem unlikely when only a few processors are employed, this is not necessarily true. We have observed non-convergence for both Maxwell's equations and elasticity applications with only four processors (where multigrid with true Gauss–Seidel smoothing converges acceptably). This poor performance mirrors the poor performance of Jacobi's method. Specifically, Maxwell's equations and elasticity commonly yield symmetric positive definite operators that are not M-matrices. Unlike M-matrices, the corresponding spectral radius of the diagonally scaled operator, $\rho(D^{-1}A)$, is significantly greater than two.

This means that undamped Jacobi iteration (without multigrid) is divergent, i.e., $\rho(I - D^{-1}A) > 1$. Given that Processor Block Gauss–Seidel is a combination of Jacobi and Gauss–Seidel methods, its convergence problems are not really surprising.

Simple non-M matrix examples exhibiting divergence of the Processor Block Gauss–Seidel method can be constructed. For example, consider a uniform grid on a unit square with the following grid stencil:

$$\begin{pmatrix} -.5 & .99 & -.5 \\ -1. & 2.03 & -1. \\ -.5 & .99 & -.5 \end{pmatrix}$$

and periodic boundary conditions. The corresponding matrix is a discrete version of the anisotropic problem

$$u_{xx} + \varepsilon_1 u_{yy} + \varepsilon_2 u = f, \quad 0 < \varepsilon_1, \varepsilon_2 \ll 1. \tag{2.2}$$

The above stencil yields a 'strange' discretization of (2.2). This discretization does not correspond to any standard scheme and has been constructed artifically to demonstrate a point. In particular, the resulting matrix is positive definite symmetric (but not an M-matrix) and thus Gauss–Seidel method is guaranteed to converge. However, Jacobi's method is divergent and Processor Block Gauss–Seidel (without multigrid) is also divergent when we associate one subdomain with each vertical grid line. In fact, Processor Block Gauss–Seidel is even divergent with only three subdomains: one corresponding to the middle grid line (i.e., $x = .5$), one corresponding to all points left of the middle grid line and one corresponding to all points right of the middle grid line. For strongly anisotropic problems such as these, line relaxation should be employed within multigrid algorithms. When each vertical line corresponds to a subdomain, vertical line Processor Block Gauss–Seidel is identical to vertical line block Jacobi which is unsuitable as a smoother (though it is not divergent). It is possible to analyze this system with Fourier analysis. We forego this analysis here as it is somewhat tedious and provides no additional insight.

Finally, it should be noted that it is possible to damp Processor Block Gauss–Seidel method to obtain convergence. However, it is not clear how best to do this. Ultimately, the optimal choice depends on processor and partitioning information. This implies that damping will depend on the number of processors and so convergence rates will also depend on the number of processors.

3. Polynomial smoothers

Motivated by the parallel Gauss–Seidel difficulties discussed in Sections 2.1 and 2.2, we investigate polynomial smoothers as an alternative. Consider the iterative procedure

$$x^{(m+1)} = x^{(m)} + p(A)(b - Ax^{(m)})$$

to solve the positive definite symmetric problem [3]

$$Ax = b$$

with initial guess $x^{(0)}$, exact solution x^*, and with

$$p(A) = \sum_{0 \leqslant j \leqslant n} \alpha_j A^j.$$

[3] To simplify the presentation, it is assumed that A is scaled so that all diagonal entries are one. It is quite easy to incorporate symmetric diagonal scaling within a polynomial smoother.

The error propagation of the above method is defined by

$$e^{(m+1)} = q(A)e^{(m)},$$

where

$$q(A) = I - p(A)A, \quad e^{(m)} = x^{(m)} - x^*.$$

To use the above iteration as a multigrid smoother, the error reduction properties of the polynomial $q(A)$ must be complementary to those of the coarse grid correction. For elliptic problems, this typically means damping high frequency errors.

3.1. Chebyshev polynomials

To motivate the use of smoothing based on Chebyshev polynomials, consider an idealized coarse grid correction. Let the eigenvalue/eigenvector pairs of A be given by

$$(\lambda_1, x_1), (\lambda_2, x_2), \ldots, (\lambda_n, x_n),$$

where

$$\lambda_1 \leqslant \lambda_2 \leqslant \cdots \leqslant \lambda_n.$$

Choose a λ^* to split the eigenvalues into two groups: low energy ($\lambda_k < \lambda^*$) and high energy ($\lambda_k \geqslant \lambda^*$). Assume that the multigrid coarse grid correction, $C_{mg}(A)$, completely annihilates error associated with small eigenvalues and does not affect error associated with large eigenvalues:

$$C_{mg}(A)x_k = \begin{cases} 0 & \text{if } \lambda_k < \lambda^*, \\ x_k & \text{if } \lambda_k \geqslant \lambda^*. \end{cases}$$

The ideal smoother is then given by a Chebyshev polynomial $q(z)$ that minimizes over the range $\lambda^* \leqslant z \leqslant \lambda_n$ subject to the constraint that $q(0) = 1$. A jth degree Chebyshev polynomial has the property that it has the smallest maximum amplitude over an interval of all jth degree polynomials. Computing the coefficients of a Chebyshev polynomial involves simple recurrences using the two eigenvalues defining the interval: λ^* and λ_n. The resulting smoother closely resembles the Chebyshev semi-iterative method to accelerate a Jacobi iteration. However, in this context the Chebyshev method is used only to damp error over the high frequencies. It is easy to show that the associated kth degree polynomial, $q(x)$, damps high frequency error by at least μ_k (i.e., the maximum polynomial amplitude over the high frequency interval is μ_k) where

$$\begin{aligned} \mu_0 &= 1, \\ \mu_1 &= \frac{\lambda_n - \lambda^*}{\lambda_n + \lambda^*}, \\ \mu_k &= \frac{\mu_1 \mu_{k-1} \mu_{k-2}}{2\mu_{k-2} - \mu_{k-1}\mu_1} \quad k \geqslant 2. \end{aligned}$$

If the idealized coarse grid correction is a good approximation to the actual multigrid coarse grid correction, μ_k should accurately approximate the actual multigrid convergence. This is the basis of classical multigrid smoothing analysis [5]. We omit the details and refer the interested reader to one of the many references on the Chebyshev semi-iterative method and on Chebyshev polynomials, e.g., [10, 15,25].

3.2. MLS: multilevel smoother polynomial

The MLS smoother is based on a combined effect of two different smoothing procedures which are constructed to complement each other on the range of the coarse grid correction. Let us denote these procedures by $\mathbb{S}_l()$ and $\hat{\mathbb{S}}_l()$. One iteration of their application to the system $A_l x = b_l$ is given by $x \leftarrow \mathbb{S}_l(A_l, b_l, x)$ and $x \leftarrow \hat{\mathbb{S}}_l(A_l, b_l, x)$, respectively. Typically, $\mathbb{S}_l()$ is a pre-smoother while $\hat{\mathbb{S}}_l()$ is a post-smoother within a multigrid iteration. When multigrid preconditioning is done for a conjugate gradient method, $x \leftarrow \hat{\mathbb{S}}_l(A_l, b_l, \mathbb{S}_l(A_l, b_l, x))$ is both the pre-smoother and post-smoother.

The procedures $\mathbb{S}_l()$ and $\hat{\mathbb{S}}_l()$ are constructed so that that their error propogation operators, S_l and $\hat{S}_l$, respectively, have certain optimum properties. Specifically, $S_l = S_l(A_l)$ is a polynomial of degree d in A_l such that the expression $\varrho(S_l^2 A)$ is minimized subject to the constraint $S_l(\Theta) = I$ where Θ denotes the zero matrix. A recurrence relationship for S_l of degree $d > 1$ can be found in [6,23]. A more convenient direct construction in terms of a transformed Chebyshev polynomial can be found in [8]:

$$S_l = p_1(A_l) \equiv \left(I - \frac{1}{r_1} A_l\right) \ldots \left(I - \frac{1}{r_d} A_l\right), \tag{3.1}$$

where $r_k = \frac{\varrho(A_l)}{2}\left(1 - \cos\frac{2k\pi}{2d+1}\right), \quad k = 1, \ldots, d$. Once S_l is defined, we can construct $A_l^S = S_l^2 A_l$ and define

$$\hat{S}_l = p_2(A_l) \equiv I - \frac{\omega}{\bar{\varrho}(A_l^S)} A_l^S, \quad \omega \in (0, 2),$$

where $\bar{\varrho}(X)$ denotes an upper bound on the spectral radius of a matrix X. For all results in this paper ω is set to one. This completes the specification of the MLS polynomial.

The precise details concerning this choice of polynomial is given in [6] where the smoother was first developed in conjunction with the smoothed aggregation multigrid. Here, we give a brief overview of the relevant material in [6]. In smoothed aggregation multigrid, the polynomial S_l is *also* used as a prolongator smoother. Prolongator smoothers take *simple* grid transfer operators and smooth them to produce improved transfer operators

$$I_{l+1}^l = S_l P_{l+1}^l, \tag{3.2}$$

where l ($l+1$) denotes the fine (coarse) level. P_{l+1}^l are the *simple* transfer operators referred to as tentative prolongators. They are obtained by a generalized aggregation procedure [24] and have the property that $(P_{l+1}^l)^{\mathrm{T}} P_{l+1}^l = I$. The coarse level matrices are then constructed in the variational multigrid fashion,

$$A_{l+1} = (I_{l+1}^l)^{\mathrm{T}} A_l I_{l+1}^l. \tag{3.3}$$

Using (3.1) as a prolongator smoother assures that the coarse level matrix based on I_{l+1}^l has the smallest spectral radius of all matrices $Ap(A)^2$ over all polynomials $p(x)$ of degree d such that $p(0) = 1$. In this sense, the smoothing effect produced by S_l is theoretically optimal. We note that the theory presented in [21] requires such a minimization in order to derive convergence rate estimates while the computational experiments justify such a choice practically.

The MLS smoother has been described and analyzed in the two-level context with $d \geqslant 1$ in [6,23]. The favorable two-level performance and analysis of [21,23] have served as a heuristic motivation for practically extending the application of MLS from the two-level to the multilevel environment. We note that the MLS smoother was first used in the multilevel context in the SAMISdat(AMG) code [7].

Given the definition of smoothing procedures $\mathbb{S}_l, \hat{\mathbb{S}}_l$, the definition of coarse level matrices (3.3) and transfer operators (3.2), the two-level multigrid error propagation operator can be written as

$$E = \hat{S}(I - SP(P^{\mathrm{T}}SASP)^{-1}P^{\mathrm{T}}SA)S.$$

Here we have dropped the level subscripts on all the operators. We note that matrices $S, \hat{S}$ are both symmetric and A-symmetric and commute both with A and with each other. Hence we can easily estimate

$$\|E\|_{A_S} \leqslant \max_{x \in Ker(P^{\mathrm{T}}A^S)\setminus\{0\}} \|\hat{S}Sx\|_{A^S} \leqslant \max_{x \in Ker(P^{\mathrm{T}}A^S)\setminus\{0\}} \min\left\{ \frac{\|Sx\|_{A^S}}{\|x\|_{A^S}}, \frac{\|\hat{S}x\|_{A^S}}{\|x\|_{A^S}} \right\}.$$

Therefore, convergence properties will be bounded by the more effective of the smoothers $\mathbb{S}, \hat{\mathbb{S}}$. In fact, a stronger result can be proved showing that the effect of the smoother $\hat{\mathbb{S}}$ increases with diminishing effect of smoother $\mathbb{S}$:

$$\frac{\|\hat{S}x\|^2_{A^S}}{\|x\|^2_{A^S}} \leqslant 1 - \frac{\|Sx\|^2_{A^S}}{\|x\|^2_{A^S}} \frac{\omega(2-\omega)}{K} \quad \forall x \in Ker(P^{\mathrm{T}}A^S) \setminus \{0\},$$

where K is a constant depending on the approximation property provided by the tentative prolongator P, which is used in deriving the latter estimate. In that sense, we can view the smoothers defined by $S, \hat{S}$ as not only complementing each other, but at the same time being intimately related to the selection of coarse spaces. Whenever smoother $\mathbb{S}$ converges very slowly on the range of the coarse-grid correction, convergence of $\hat{\mathbb{S}}$ should improve, and vice versa. While derived for the smoothed aggregation multigrid, the MLS smoother is not restricted to this multigrid scheme. In Section 5, results are given with the MLS smoother and geometric multigrid.

3.3. Eigenvalue estimates

The main disadvantage of polynomial methods is that they require extreme eigenvalues of the system. This is a well-known problem associated with the Chebyshev semi-iterative method. When used as a stand-alone iterative solver, it requires the lowest eigenvalue which is often not available nor practical to compute. However, this is not an issue for smoothers. The MLS smoother does not need a lower eigenvalue estimate and the Chebyshev smoothers need an estimate of the lower end of the high energy modes. In fact, it is possible to simply divide the largest eigenvalue by a constant factor (e.g., 30). This fraction should be related to how rapidly coarsening occurs within the multigrid method. We note, however, that multigrid convergence does not seem very sensitive to this estimate. These lower eigenvalue estimates are discussed further in Section 5.1.

While the largest eigenvalue estimate is still needed, it is much easier to obtain. For example, when A is an M-matrix, the Gershgorin theorem can be applied to get a fairly sharp upper bound on the largest eigenvalue. When the M-matrix is scaled to have unit diagonal entries, the Gershgorin estimate is often just the number '2' for the largest eigenvalue. For non-M-matrices, it is possible to estimate the eigenvalue using a small number of Lanczos or conjugate gradient iterations. In many cases, this estimate is required elsewhere in the code and so is readily available. In fact, most of our numerical results are generated using smoothed aggregation algebraic multigrid. Smoothed aggregation already requires the largest eigenvalue estimate, so there is no additional parameter estimation cost associated with the polynomial smoothers. One somewhat subtle issue is that polynomial methods are very sensitive to an under-estimate of the largest eigenvalue but not too sensitive to an over-estimate. Thus, we recommend scaling the estimate by a small factor because most computational methods (e.g., Lanczos) give lower bounds to the largest eigenvalues.

4. Performance analysis

For the most part, polynomial relaxation is not seriously used within the multigrid community. This is largely due to the good smoothing properties of Gauss–Seidel and the parameter estimation required by polynomial methods. Gauss–Seidel's advantages are easily seen by examining multigrid convergence rates for the standard 5-point approximation to the Laplacian on two-dimensional structured grids. The convergence rates shown in Table 1 are well known and correspond to using two-level standard geometric multigrid. The superiority of Gauss–Seidel is fairly evident if it is assumed that k Gauss–Seidel steps cost roughly the same as the application of a kth degree polynomial smoother. In particular, multigrid using 1 step of red-black Gauss–Seidel requires only 20 iterations while using a damped Jacobi smoother (equivalent to a 1 step Chebyshev polynomial) requires 53 iterations (2.5 times larger). This strong Gauss–Seidel performance combined with the freedom from parameter estimation have left polynomial methods somewhat ignored.

On closer examination of Table 1, it seems that the situation is not quite so bleak for polynomial schemes. If we look at the 2-step methods, red-black Gauss–Seidel smoothing requires 11 iterations while Chebyshev requires 19, a somewhat smaller ratio. Further, 2 steps of lexicographical Gauss–Seidel requires 16 iterations (only 3 less than Chebyshev). It could certainly be argued that comparisons with lexicographical Gauss–Seidel are more representative than comparisons with red-black Gauss–Seidel as it is not common practice to color Gauss–Seidel smoothers. This is, in part, due to the potentially poor cache performance of multi-color orderings. Of course, our focus is not to advocate the application of polynomial smoothers to structured grid Poisson problems on serial computers. Gauss–Seidel works extremely well in this situation. Our discussion is intended to show that even in the serial structured environment the price for using polynomial smoothers is not as high as commonly believed.

A comparison between polynomial and Gauss–Seidel methods must also consider computational work. On close inspection, the cost is not identical for k-step Gauss–Seidel and k-degree polynomial methods. Specifically, a polynomial smoother can easily skip the first matrix–vector product when a zero initial guess is present. A zero guess occurs when pre-smoothing on coarse grids within a multigrid V cycle and when pre-smoothing on the finest grid when multigrid is used as a preconditioner. [4] This means that a 2nd degree polynomial would only require one matrix–vector product on each grid level. Gauss–Seidel method can also take advantage of a zero guess by suppressing all calculations involving information that has not yet been updated. This corresponds to savings of half a matrix–vector product. This means that 2 steps of Gauss–Seidel require about one and a half matrix–vector products on each level. However, Gauss–Seidel smoothers can take advantage of the zero initial guess only if lower and upper triangular parts of the matrix can be accessed inexpensively. This is possible when the lower triangle, upper triangle and diagonal of the matrix are stored separately. Unfortunately, many commonly used sparse storage formats do not support this and so in this case the Gauss–Seidel method can not take advantage of a zero initial guess. When all of these factors are considered together, the gap between Gauss–Seidel and polynomial methods is not nearly as great as on first inspection.

While minimizing run time is our primary focus, there are two significant software advantages to polynomial smoothers when developing numerical libraries/packages. First, smoothing properties on multiple processors are identical to the one processor case. In contrast, Processor Block Gauss–Seidel smoothing rates typically deteriorate with increasing number of processors, and Parallel Block Multi-color Gauss–Seidel requires different orderings depending on the number of processors. Second, polynomial smoothers only rely on matrix–vector products. These matrix–vector products are usually optimized by application users as well as by any third party package which might be used to produce an outer Krylov

[4] This is fairly common place especially within algebraic multigrid codes.

Table 2
Summary of trade-offs between Gauss–Seidel and polynomial smoothers

	Gauss–Seidel	Polynomial
Convergence	Excellent smoothing	Good smoothing with higher order
Parameters	None needed	Largest eigenvalue estimate
Parallel	Complex code for block coloring and high efficiency. Sub-optimal smoothing for local Gauss–Seidel.	Trivial
Mflops	High rates require special code, block multi-color can have poor cache utilization	Trivial
Data neutrality	Requires special kernel for good performance	Trivial
Zero guess	$\frac{1}{2}$ matrix–vector product saved with special code requiring particular matrix storage	1 Matrix–vector product saved trivially

method. Thus, while Gauss–Seidel methods require special matrix kernels and formats for optimal performance, efficient polynomial methods have no such requirements and easily take advantage of often already existing optimized matrix–vector products. Table 2 summarizes the advantages and disadvantages of the smoothers that have been discussed.

5. Numerical studies

To study the behavior of the polynomial smoothers, three application areas are considered. The first problem (Section 5.2) is the Poisson equation

$$\nabla^2 \mathbf{u} = \mathbf{f},$$

where u is a scalar valued function on the unit cube with Dirichlet boundary conditions. The second problem (Section 5.3) is the linear elasticity

$$(\lambda + \mu)\nabla(\nabla \cdot \mathbf{u}) + \mu\nabla^2 \mathbf{u} = \mathbf{f},$$

where λ and μ are the Lamé parameters and $\mathbf{u}$ is a vector valued function of displacements.

The third problem (Section 5.4) corresponds to an eddy current formulation of Maxwell's equations

$$\nabla \times \nabla \times \mathbf{u} + \sigma(x, y, z)\mathbf{u} = \mathbf{f},$$

where $\sigma(x, y, z)$ is the conductivity of the material and $\mathbf{u}$ is a vector valued function corresponding to the electric field.

The smoothed aggregation multigrid method [22] is used for the Poisson and for some of the elasticity problems. Since this multigrid method requires an estimate of the spectral radius of the matrix, this estimate can be reused within the polynomial smoothers without additional expense if the same scaling is used for the prolongation smoother and the multigrid smoother (e.g., diagonal). The solver is conjugate gradient preconditioned with one iteration of V-cycle multigrid. To ensure symmetry of the preconditioner, either symmetric Gauss–Seidel is employed for pre and post smoothing or the same polynomial is invoked for both pre and post smoothing. When an initial guess is not present, the polynomial smoothers omit the first matrix–vector product. It is not possible to efficiently omit work for the Gauss–Seidel method as the already existing matrix formats do not separately store the upper and lower parts of the matrix. All MLS results use the lowest degree MLS polynomial which is 4. All parallel data was obtained using the ASCI Red machine at Sandia National Laboratories.

For Maxwell's equations a different algebraic multigrid method is used [4,18]. This scheme uses a 2-step distributed relaxation smoother. Step one consists of applying either symmetric Gauss–Seidel or polyno-

mial smoothing. Step two corrects this solution by projecting a residual equation into the kernel or null space of the curl operator and then applying either polynomial or symmetric Gauss–Seidel smoothing to the projected operator. The details of this smoother can be found in [14]. For this paper, it is sufficient to understand that this distributed relaxation ultimately depends on either Gauss–Seidel or polynomial methods. Conjugate gradient is again employed as an outer iteration. To maintain symmetry, the distributed pre-smoothing operates on the original operator followed by the projected system while the post-smoothing operates first on the projected system. The second system within the smoother always starts with a zero initial guess while the first system has a zero initial guess only during the pre-smoothing.

5.1. Eigenvalue estimate sensitivity

We investigate the sensitivity of the polynomial methods to eigenvalue estimates on a truncated cone linear elasticity problem with 21,600 degrees of freedom. Fig. 3 shows the problem which is meshed with first order mixed hexahedral elements and is fixed at the base and loaded at the other end. In these experiments, we use 100 iterations of the conjugate gradient method to compute a relatively accurate estimate λ_{est} of the largest eigenvalue. (In practice, we have found that 5 or 10 iterations to be adequate for most of the problems that we have encountered.) We then compare the number of iterations to converge to the solution using various perturbations of λ_{est} for the largest eigenvalue. For Chebyshev polynomials a lower eigenvalue, λ^*, is also needed to delimit the low and high frequencies. The λ^* estimate is also perturbed in these experiments. The data in Table 3 shows that convergence is not very sensitive to the λ^* used for the lower end of the high frequency spectrum. The data also illustrates that the Chebyshev smoothers are very sensitive to underestimating the highest eigenvalue. This is expected because Chebyshev polynomials lose their effectiveness rapidly in the part of the spectrum above the point at which they are optimal. For the remainder of this paper, we estimate the largest eigenvalue by using 10 steps of conjugate gradient and then boosting it by a factor of 1.1 (i.e., $\lambda_{\max} = 1.1\lambda_{\text{est}}$), and take $\lambda^* = \lambda_{\max}/30$.

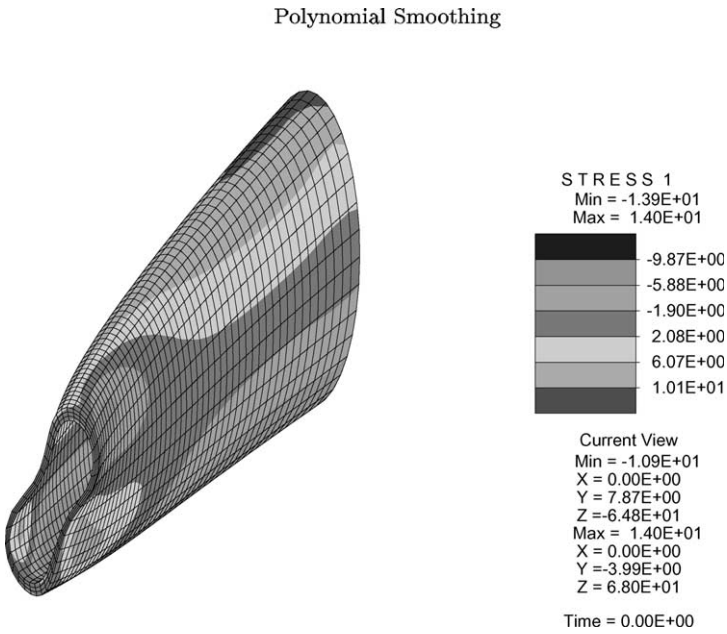

Fig. 3. Deformed shape of cone problem.

Table 3
Sensitivity of fourth order Chebyshev smoother to eigenvalue estimates

λ_{max}	λ^*	Chebyshev
$1.1\lambda_{est}$	$\lambda_{est}/18.9$	34
$1.1\lambda_{est}$	$\lambda_{est}/37.1$	31
$1.1\lambda_{est}$	$\lambda_{est}/109.8$	30
$1.0\lambda_{est}$	$\lambda_{est}/109.8$	28
$.9\lambda_{est}$	$\lambda_{est}/109.8$	73

Total iterations to reduce initial residual by 6 orders of magnitude.

5.2. Poisson equation

Table 4 gives the iteration counts and solve times required to reduce the initial residual by 10^6 for Poisson's equation on the unit cube with Dirichlet boundary conditions. This data shows that the second order Chebyshev smoother and the Processor Block Gauss–Seidel (indicated by B-SGS in Table 4) give almost identical convergence rates and solve times. While the MLS convergence is fine, the times are greater because the lowest degree MLS polynomial is 4 which is over-kill for this problem (i.e., too much smoothing is done for this simple problem).

5.3. Linear elasticity

The next test problem is that of an automotive wheel modeled with triangular shell elements (data courtesy of Charbel Farhat). The problem has 59,490 degrees of freedom (dof). A detail of the associated mesh is shown in Fig. 4. Table 5 shows the iteration counts and total times required to reduce the residual by a factor of 10^6 for the shell problem (''SHELL''), and for two finite-difference discretizations of an elasticity problem associated with atomic positions in nanostructures (''HELIX'', data courtesy of Stefan Goedecker). The latter problem is singular, with kernel dimension of 6 and many eigenvalues very close to zero. Performance of the multigrid using one step of the MLS smoother and 2 steps of symmetric Gauss–Seidel is compared on a single 1 GHz Pentium III processor. The data shows that on a serial machine the multigrid iteration using Gauss–Seidel smoother beats the one using MLS smoother (fourth degree polynomial) by a small margin in terms of computation time.

The next elasticity problem is a series of thin concentric spheres enclosed in a cube of ''soft'' material (with symmetric boundary conditions so that only one octant need be modeled). The elements are first

Table 4
Iterations and run times to solve the Poisson equation

Number of elements	Procs	Smoother	Iterations	Time (s)
129^3	16	1 B-SGS	11	18.0
		2nd order Chebyshev	10	17.1
		MLS	13	32.7
221^3	121	1 B-SGS	17	20.4
		2nd order Chebyshev	15	18.7
		MLS	13	33.7
385^3	1024	1 B-SGS	19	19.5
		2nd order Chebyshev	15	17.5
		MLS	14	29.3

B-SGS indicates Processor Block Gauss–Seidel.

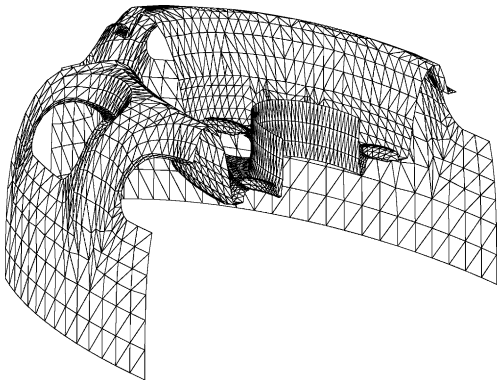

Fig. 4. "Shell" problem mesh.

Table 5
3D Elasticity problems in serial

Problem	dof	Smoother	Iterations	Time (s)
Small HELIX	52,665	SGS	21	28.1
		MLS	18	28.7
Large HELIX	210,489	SGS	22	124.2
		MLS	19	129.9
SHELL	59,490	SGS	18	18.4
		MLS	19	20.2

SGS indicates symmetric Gauss–Seidel.

order mixed hexahedral elements. The sphere is composed of seventeen alternating layers of *hard* and *soft* materials. Table 6 shows a summary of the constitution of the two material types. The loading and boundary conditions are an imposed uniform displacement (down), on the top surface. Fig. 5 shows the deformed shape of the 79,679 dof version of the problem. The mesh is parameterized and the number of processors is selected to keep about 40 K dof per processor. Each successive problem has one more layer of elements through each of the seventeen shell layers and in the outer soft domain. A similar refinement is done in the other two directions. This study uses ten versions of this problem ranging in size from about 80 K to about 76 M dof. In this example, a geometric multigrid method is used as a preconditioner for conjugate gradient [2].

Fig. 6 shows iteration counts plotted against number of processors for first and fourth order Chebyshev polynomial smoothers, the MLS smoother, and one iteration of Parallel Block Multi-color Gauss–Seidel (symmetrized by reversing the order of the equations in the post smoothing step). The data shows that the iteration counts are about constant for each method as the problem size is increased and that Gauss–Seidel is more effective than a first order Chebyshev smoother in reducing the residual. Fig. 7, however, shows that

Table 6
Concentric spheres model materials

Material	Elastic modulus	Poisson ratio
Soft	10^{-4}	0.49
Hard	1	0.3

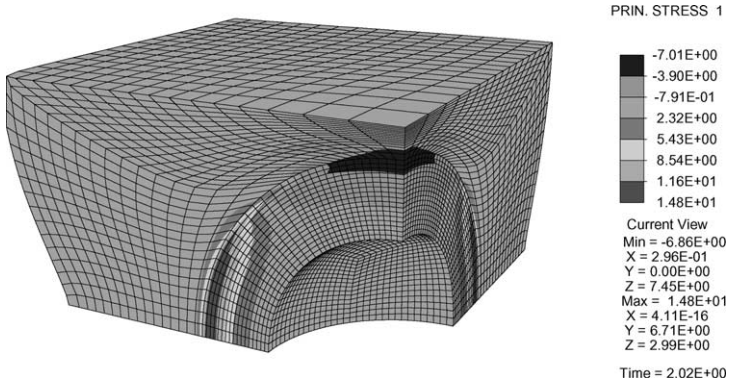

Fig. 5. Deformed shape of concentric spheres problem.

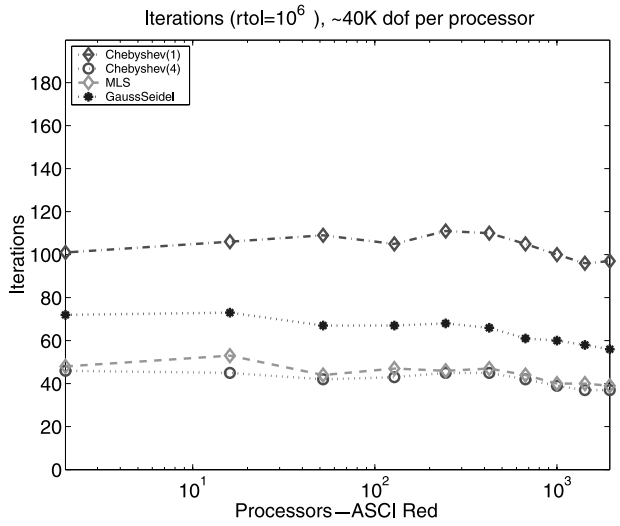

Fig. 6. Number of iterations using geometric multigrid with polynomial smoothers and Gauss–Seidel smoothers on concentric spheres problem.

the polynomial smoothers are faster, especially as the degree of parallelism increases. Fig. 8 shows the Megaflop rates per processor. This data illustrates the main disadvantage of Gauss–Seidel, namely that the flop rate per processor decreases as the number of processors increases. The flop rates for the polynomial smoothers decrease only slightly as almost all the work is in highly optimized matrix–vector product kernels.

5.4. Eddy current formulation of Maxwell's equations

For Maxwell's equations we consider an application arising from modeling the Z-pinch machine at Sandia [4]. In this case, σ varies from 10^{-3} to 10^{3} and the problem includes both Dirichlet and Neumann boundary conditions. A picture of the computational domain of interest is given in Fig. 9. Table 7 gives

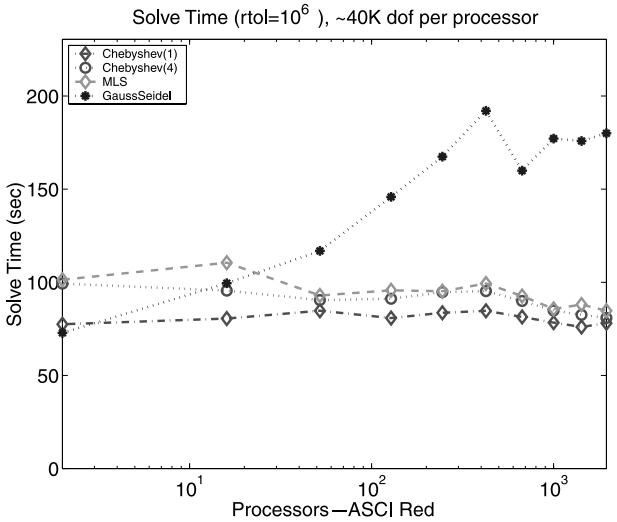

Fig. 7. Run time using geometric multigrid with polynomial smoothers and Gauss–Seidel smoothers on concentric spheres problem.

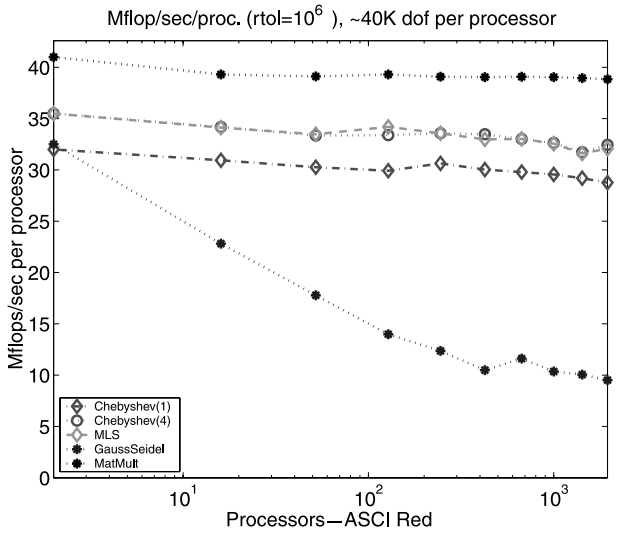

Fig. 8. Megaflop rates using geometric multigrid with polynomial smoothers and Gauss–Seidel smoothers on concentric spheres problem.

iterations and run time using both Processor Block SOR and polynomial methods within the distributed smoother. Without damping (not shown here for multiple processors), multigrid using Processor Block Gauss–Seidel does not converge on four or more processors. While damping improves robustness, it can degrade the overall smoother performance. This is seen in Table 7 where a damping parameter is chosen experimentally based on the number of processors. [5] In particular, the multigrid method degrades

[5] The single processor run does not use any damping.

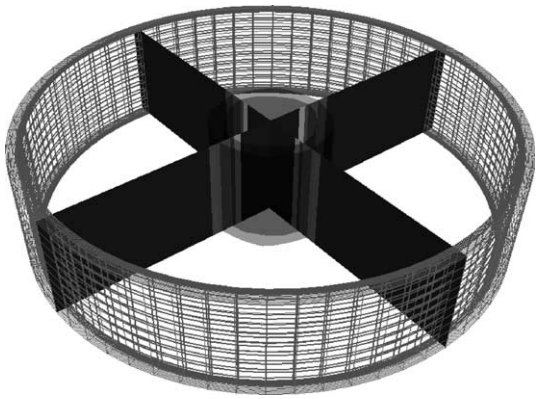

Fig. 9. Idealized Zpinch picture.

Table 7
Iteration counts and times (s) for Maxwell's equations

Procs	Number of elements	Dis. B-SSOR		Dis. 4th order Chebyshev		Dis. MLS	
		Iter	Time	Iter	Time	Iter	Time
1	18,720	15	60.2	21	90.7	27	98.1
20	149,760	71	138.8	37	69.8	46	73.1
50	508,560	61	173.2	31	86.1	36	85.4
160	1,608,480	86	311.5	40	142.1	49	154.0
400	4,068,480	95	652.6	43	316.5	53	346.5

'Dis.' indicates 2-stage distributed relaxation using either Processor Block symmetric SOR (B-SSOR), Chebyshev or MLS in both stages.

noticeably as the damped version does not perform as well as the undamped version. By contrast, the polynomial smoothers perform well even when many processors are employed. It should be noted that the iteration growth using the polynomial smoothers is due to the somewhat poor approximation property of the grid transfers. A new algebraic multigrid variant has been developed which partially addresses this problem [3]. Overall, the Chebyshev smoother is more than twice as fast as the Processor Block SOR smoother on the larger simulations.

6. Conclusions

We have shown that polynomial smoothing within a multigrid method may be preferable to traditional Gauss–Seidel smoothing for parallel unstructured grid problems corresponding to positive semi-definite matrices. They are easy to implement and integrate within existing codes, perform identically in serial and in parallel, and require only a well-tuned matrix–vector product kernel. By contrast Gauss–Seidel methods are problematic. Parallel Block Multi-color Gauss–Seidel can be difficult to implement and will run at slower Megaflop rates than polynomial methods. When many processors are used, Processor Block (or local) Gauss–Seidel methods often suffer severe convergence degradation (especially for non-M matrix applications). Experiments have been shown from three different codes and from three different application areas. The results illustrate that polynomial smoothers are very competitive in serial and out-perform Gauss–Seidel smoothers in parallel.

References

[1] M.P. Adams, A distributed memory unstructured Gauss–Seidel algorithm for multigrid smoothers, in: ACM/IEEE Proceedings of SC01: High Performance Networking and Computing, 2001.
[2] M.F. Adams, J. Demmel, Parallel multigrid solver algorithms and implementations for 3D unstructured finite element problem, in: ACM/IEEE Proceedings of SC99: High Performance Networking and Computing, Portland, Oregon, November, 1999.
[3] P. Bochev, C. Garasi, J. Hu, A. Robinson, R.T. Ro, An improved algebraic multigrid method for solving Maxwell's equations, Sandia National Laboratories Technical Report # 2002-8222, 2003, to appear.
[4] P. Bochev, J. Hu, A. Robinson, R. Tuminaro, Towards robust 3D Z-pinch simulations: discretization and fast solvers for magnetic diffusion in heterogeneous conductors, Elec. Trans. Numer. Anal. 15 (2003).
[5] A. Brandt, Multi-level adaptive solutions to boundary-value problems, Math. Comput. 31 (1977) 333–390.
[6] M. Brezina, Robust iterative solvers on unstructured meshes, Ph.D. Thesis, University of Colorado at Denver, Denver, CO, 1997, pp. 80217–83364.
[7] M. Brezina, SAMISdat(AMG) version 0.98-User's Guide, 2002.
[8] M. Brezina, C.I. Heberton, J. Mandel, P. Vaněk, An iterative method with convergence rate chosen a priori, UCD/CCM Report 140, Center for Computational Mathematics, University of Colorado at Denver, February 1999. Available from http://www-ath.cudenver.edu/ccmreports/repl40.ps.gz.
[9] W.L. Briggs, V.E. Henson, S. McCormick, A Multigrid Tutorial, second ed., SIAM, Philadelphia, 2000.
[10] G.H. Golub, C.F.V. Loan, Matrix Computations, Johns Hopkins University Press, Baltimore, 1996.
[11] W. Hackbusch, Multigrid Methods and Applications, Computational Mathematics, vol. 4, Springer, Berlin, 1985.
[12] W. Hackbusch, Iterative Solution of Large Sparse Systems of Equations, Springer, New York, 1994.
[13] V. Henson, U. Yang, BoomerAMG: A parallel algebraic multigrid solver and preconditioner, Technical Report UCRL-JC-139098, Lawrence Livermore National Laboratory, 2000.
[14] R. Hiptmair, Multigrid method for Maxwell's equations, SIAM J. Numer. Anal. 36 (1998) 204–225.
[15] E. Isaacson, H.B. Keller, Analysis of Numerical Methods, Wiley, New York, 1966.
[16] A. Jameson, T.J. Baker, Multigrid solution of the Euler equations for aircraft configurations, AIAA Paper No. 84-0093, 1984.
[17] A. Jameson, W. Schmidt, E. Turkel, Numerical solution of the Euler equations by finite volume methods using Runge–Kutta time-stepping schemes, AIAA Paper No. 8I-1259, 1981.
[18] S. Reitzinger, J. Schöberl, An algebraic multigrid method for finite element discretizations with edge elements, Numer. Linear Algebra Appl. 9 (2002) 223–238.
[19] K. Stüben, U. Trottenberg, Multigrid methods: fundamental algorithms, model problem analysis and applications, in: W. Hackbusch, U. Trottenberg (Eds.), Multigrid Methods, Lecture Notes in Mathematics, vol. 960, Springer, Berlin, 1982, pp. 1–176.
[20] U. Trottenberg, C. Oosterlee, A. Schüller, Multigrid, Academic Press, London, 2001.
[21] P. Vaněk, M. Brezina, J. Mandel, Convergence of algebraic multigrid based on smoothed aggregation, Numerische Mathematik 88 (2001) 559–579.
[22] P. Vaněk, J. Mandel, M. Brezina, Algebraic multigrid based on smoothed aggregation for second and fourth order problems, Computing 56 (1996) 179–196.
[23] P. Vaněk, M. Brezina, R. Tezaur, Two-grid method for linear elasticity on unstructured meshes, SIAM J. Sci. Comput. 21 (1999) 900–923.
[24] P. Vaněk, J. Mandel, M. Brezina, Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems, Computing 56 (1996) 179–196.
[25] R.S. Varga, Matrix Iterative Analysis, Prentice-Hall, Englewood Hills, NJ, 1962.
[26] P. Wesseling, An Introduction to Multigrid Methods, Wiley, Chichester, 1992, Reprinted by www.MGNet.org.