

Accepted Manuscript

Efficient relaxed-Jacobi smoothers for multigrid on parallel computers

Xiang Yang, Rajat Mittal

PII: S0021-9991(16)30651-9
DOI: <http://dx.doi.org/10.1016/j.jcp.2016.12.010>
Reference: YJCPH 7009

To appear in: *Journal of Computational Physics*

Received date: 30 August 2016
Accepted date: 7 December 2016

Please cite this article in press as: X. Yang, R. Mittal, Efficient relaxed-Jacobi smoothers for multigrid on parallel computers, *J. Comput. Phys.* (2016), <http://dx.doi.org/10.1016/j.jcp.2016.12.010>

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.



Efficient Relaxed-Jacobi Smoothers for Multigrid on Parallel Computers

Xiang Yang^{a,*}, Rajat Mittal^a

^a Department of Mechanical Engineering, the Johns Hopkins University, Baltimore, MD, 21218

Abstract

In this Technical Note, we present a family of Jacobi-based multigrid smoothers suitable for the solution of discretized elliptic equations. These smoothers are based on the idea of scheduled-relaxation Jacobi proposed recently by Yang & Mittal (*Journal of Computational Physics*, Vol 274, DOI: 10.1016/j.jcp.2014.06.010.) and employ two or three successive relaxed Jacobi iterations with relaxation factors derived so as to maximize the smoothing property of these iterations. The performance of these new smoothers measured in terms of convergence acceleration and computational workload, is assessed for multi-domain implementations typical of parallelized solvers, and compared to the lexicographic point Gauss-Seidel smoother. The tests include the geometric multigrid method on structured grids as well as the algebraic grid method on unstructured grids. The tests demonstrate that unlike Gauss-Seidel, the convergence of these Jacobi-based smoothers is unaffected by domain decomposition, and furthermore, they outperform the lexicographic Gauss-Seidel by factors that increase with domain partition count.

Keywords: Multigrid, Jacobi, over-relaxation, scheduled-relaxation, structured grid, unstructured grid, elliptic equations, smoothers

1. Introduction

With the possibility of solving problems of size N with $O(N)$ computation, the multigrid (MG) method is among the most efficient methods for solving elliptic partial differential equations on a single processor [1, 2, 3]. It is therefore not surprising that since the early work in the 1970s [4, 5], significant effort has put into implementing MG in physics and engineering simulation codes, particularly in fluid dynamics, heat transfer and structural mechanics [3, 6, 7, 8]. While parallel computing with large threads counts was not possible in the 70s, simulations in the current era routinely tackle problems with processor counts up to and even exceeding $O(10^3)$ (see e.g. Refs. [9, 10]). The increase in thread counts is putting an ever-increasing premium on the parallel scalability of computational algorithms. In this context, the multigrid method, despite its unmatched performance on a single processor, is losing its preeminence to highly scalable methods such as conjugate gradient [11, 12].

Domain decomposition, i.e. partitioning the entire set of nodes/elements/grid points/degrees-of-freedom in the computational domain into subdomains, and assigning the computational work in each subdomain to a different processor/thread, is the preferred method for effective parallelization in both structured and unstructured grid based solvers, and it is in this computational paradigm, that MG fails to provide scalability on large thread counts. The cause of this can be traced to two factors: first, the reduction in floating-point operation (FLOP) count on the coarse grids reduces the overall ratio of computation-to-communication. The degree to which this factor would affect the parallel performance is, however, highly dependent on the computational platform, and the answer to ameliorating this factor likely lies in improvements in the hardware.

The second factor implicated here is that the smoothing property of commonly used smoothers such as lexicographic point Gauss-Seidel (LEX-GS) is affected adversely by the presence of the subdomain boundaries. In domain partitioned implementations of multigrid, the updated values in neighboring subdomains are not available during the LEX-GS iteration in a given subdomain, and the common practice is to employ the simple Jacobi iteration on these subdomain boundaries. Given that the Jacobi iteration lacks the smoothing property, it stands to reason that a mixing of LEX-GS with the Jacobi iteration at the subdomain boundaries will compromise the overall smoothing property of the multidomain iteration and lead to deterioration of the multigrid convergence with increasing domain count. Indeed, this behavior has been confirmed in previous studies [13, 14, 12] and is also demonstrated in later sections of this paper. This second factor is therefore a more fundamental issue with the

*email: xiangyang@jhu.edu

convergence properties of typical MG smoothers in multi-domain solvers; this issue is independent of the hardware employed, and is *the* issue that is the subject of this paper.

Within the context of the discussion in the previous paragraph, the properties that are desirable in any new smoother are: (1) it should exhibit performance on a single processor that is comparable or better than the lexicographic point-Gauss-Seidel (LEX-GS); (2) its convergence properties should be undiminished by domain decomposition; and (3) it should be applicable to structured as well as unstructured grid solvers. In this regard, most Gauss-Seidel based smoothers (including lexicographic point-Gauss-Seidel, line-Gauss-Seidel and block-Gauss-Seidel) do not satisfy the second property, and we will explore this issue further in this paper. It should be pointed out that unstructured grid solvers employ algebraic [8] or agglomeration [15] multigrid methods (as opposed to geometric multigrid on structured grids) and the only smoother compatible with these MG methods is the lexicographic point-Gauss-Seidel. This is also the reason why we choose lexicographic point-Gauss-Seidel (LEX-GS) as the baseline for comparison in the current paper. The red-black Gauss-Seidel [16, 12] is one smoother that is designed to be agnostic to domain decomposition, but it only works in conjunction with structured grids. Finally, the under-relaxed Jacobi (typically with an under-relaxation factor between 0.5 and 0.7) can function as a smoother [3] that satisfies the second and third conditions, but its convergence significantly lags behind even the simplest lexicographic Gauss-Seidel smoother [3, 17]. Thus, no currently available smoother exhibits all the three properties outlined above.

In this Technical Note, we propose MG smoothers based on relaxed Jacobi(RJ) iterations. This idea is inspired by the recently proposed “scheduled relaxation Jacobi” (SRJ) method [18, 19, 20, 21] which employs a sequence of Jacobi iterations with a predetermined schedule of under and over-relaxations, and which has been shown to generate convergence accelerations exceeding the GS method by factors of $O(10^2)$ for solving discretized elliptic equations. The relaxed-Jacobi(RJ) smoother here consists of two or three successive iterations with under and/or over-relaxation factors derived so as to maximize the smoothing property of the iterative procedure. We demonstrate using a simple model problem on structured as well as unstructured grids, that this smoother satisfies all the three conditions outlined above.

2. RJ-based Multigrid Smoothers

The model problem here is the homogeneous Laplace equation,

$$\nabla^2 u = 0, \quad (1)$$

where u is defined on a square domain $[0, \pi]^d$; $d = 1, 2, 3$, is the dimension of the problem, and the equation is solved using a 2nd-order central-difference scheme on a uniform mesh of size N^d . Neumann boundary conditions are used on all boundaries and we use ghost-cells to enforce the boundary conditions. Unless noted otherwise, a random field generated from a uniform probability distribution is used as the initial guess for the iterations. It is noted that since the convergence property of the Poisson equation is no different from that of the Laplace equation, conclusions made here for the Laplace equation are equally applicable to Poisson equations.

A brief derivation of relaxed Jacobi(RJ) smoothers is described in this section. A single Jacobi iteration for a 2D problem with relaxation factor ω can be written as:

$$u_{i,j}^{n+1} = (1 - \omega) u_{i,j}^n + \frac{\omega}{4} (u_{i,j-1}^n + u_{i,j+1}^n + u_{i-1,j}^n + u_{i+1,j}^n) \quad (2)$$

Clearly, a single Jacobi iteration (with or without relaxation) cannot serve as an effective smoother and we therefore need to employ multiple Jacobi iterations at each smoothing step. However, a large number of iterations at each smoothing step are also not desirable since this increases the FLOP count and inter-domain communication. It also goes against the fundamental principle in multigrid of performing the least amount of computational work between each coarsening. Given these considerations, we limit the current analysis to schemes with two or three iterations. In keeping with the SRJ methodology, we allow for different relaxation factors in each iteration and the objective here is to determine the relaxation factors for these schemes so as to maximize the smoothing property of these iterations.

In Von-Neumann-based convergence analysis, the amplification factor $G_{\mathbf{k}}$ associated with an iterative scheme is defined as the factor by which a Fourier mode of the error ϵ , given by $\hat{\epsilon}_{\mathbf{k}} = e^{-i\mathbf{k} \cdot \mathbf{x}}$ (where \mathbf{k} is the vector wavenumber) is attenuated (or amplified) in one iteration. Given this, the “smoothing factor” ν can be formally defined to be:

$$\nu = \max \{ G_{\mathbf{k}} | \mathbf{k} \in K_h \}, \quad (3)$$

where K_h is the high wavenumber space [7]. In d -dimensional space with the computational domain in the Fourier space $[0, \pi]^d$, $K_h = [0, \pi]^d \setminus [0, \pi/2]^d$.

Table 1: Smoothing factors determined for $M = 2$ (top three rows), $M = 3$ (bottom three rows) RJ multigrid smoothers. d is the problem dimension, ω_m is the relaxation factor for the m^{th} Jacobi iteration and ν is the smoothing factor as defined in Eq. 3. The smoothing factors of the LEX-GS scheme are also included.

d	ω_1	ω_2	ω_3	ν_{RJ}	$\nu'_{\text{RJ}} = \nu_{\text{RJ}}^{(1/M)}$	$\nu_{\text{LEX-GS}}$
1	0.8723	0.5395	-	0.059	0.243	0.447
2	1.3895	0.5617	-	0.220	0.469	0.500
3	1.7319	0.5695	-	0.342	0.585	0.567
1	0.9372	0.6667	0.5173	0.010	0.216	
2	1.6653	0.8000	0.5264	0.074	0.420	
3	2.2473	0.8571	0.5296	0.148	0.528	

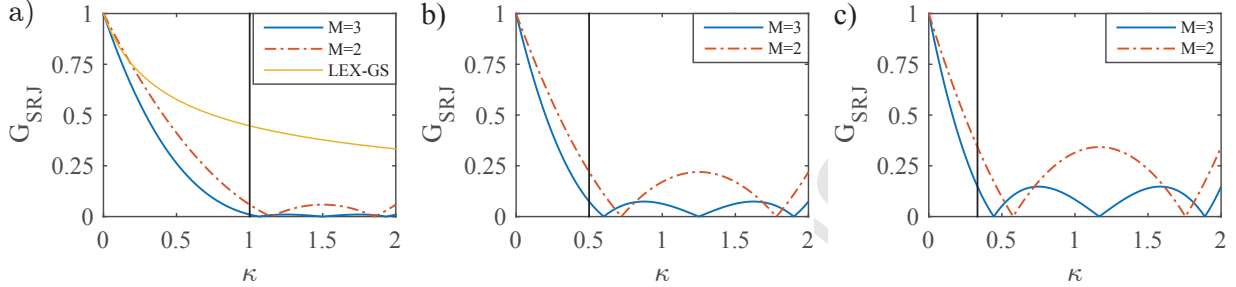


Figure 1: Amplification factor curves of the optimal RJ smoothers for different problem dimensions. a) $d = 1$ RJ smoothers compared to LEX-GS b) $d = 2$ RJ smoothers and c) $d = 3$ RJ smoother. The vertical lines demarcate the low and high-wavenumbers in the κ space. For $d=2$ and 3, the amplification of LEX-GS cannot be expressed in terms of κ and are therefore not shown here.

Von-Neumann analysis shows that the amplification factor of an RJ smoother reads:

$$G_{\text{RJ}} = \prod_{m=1}^M |1 - \omega_m \kappa|, \quad \kappa = \frac{2}{d} \sum_{i=1}^d \sin^2 \left(\frac{k_i \Delta x_i}{2} \right). \quad (4)$$

where M , the count of Jacobi iterations employed in the smoothing step, is limited here to $M = 2$ and 3. The goal here is to determine the ω_m that minimize ν (defined in Eq. 3) for the given scheme. Conveniently, the problem becomes one-dimensional when shifted to the space of κ (defined in Eq. 4) and K_h maps to a one-dimensional domain (denoted by κ_h) ranging from $\kappa = (1/d)$ to 2, where d denotes the dimension of the problem. The mathematical problem of determining an effective smoother therefore becomes equivalent to minimizing G_{RJ} in the domain of κ_h . Analysis indicates that there is one local maximum in $G_{\text{RJ}}(\kappa)$ for $M = 2$ at $\kappa'_2 = (\omega_1 + \omega_2)/(2\omega_1\omega_2)$, and two local maxima in $G_{\text{RJ}}(\kappa)$ for $M = 3$ at $\kappa'_3 = (\beta + \sqrt{\beta^2 - \alpha \cdot \gamma})/3\gamma$ and $\kappa''_3 = (\beta - \sqrt{\beta^2 - \alpha \cdot \gamma})/3\gamma$, where $\alpha = \omega_1 + \omega_2 + \omega_3$, $\beta = \omega_1\omega_2 + \omega_2\omega_3 + \omega_1\omega_3$, $\gamma = \omega_1\omega_2\omega_3$. and the subscript denotes the value of M for the scheme. The objective now is to choose ω 's that minimizes the maximum value of G_{RJ} in the high wavenumber space.

Following the procedure in Ref. [18], we achieve this by equalizing all the local maxima in the closed interval $[1/d, 2]$, which include the values at $\kappa = 1/d$ and $\kappa = 2$. The readers are directed to Ref. [18] for detailed discussion on this mathematical lemma and empirical evidence for this lemma is provided in next section. For $M = 2$, this condition leads to the equalization of $G_{\text{RJ}}(1/d) = G_{\text{RJ}}(\kappa'_2) = G_{\text{RJ}}(2)$, which provides two conditions for determining the two unknown ω . Similarly, for $M = 3$, we require $G_{\text{RJ}}(1/d) = G_{\text{RJ}}(\kappa'_3) = G_{\text{RJ}}(\kappa''_3) = G_{\text{RJ}}(2)$ which provides the requisite three conditions. These conditions lead to sets of coupled, non-linear algebraic equations for the ω s, which are solved numerically using a Newton-iterative method. The resulting relaxation factors are tabulated in table 1 and Fig. 1 show the amplification factor curves of the resulting RJ-based smoothers. The table and the figure show that the “optimal” relaxation factors as well as the amplification curves depend on the problem dimension, and this is because the high wavenumber space κ_h , depends on d . Fig. 1a also suggests that the new smoothers might be superior to the LEX-GS iteration. Table 1 also provides the smoothing factors for the two RJ based smoothers as well as the conventional LEX-GS iteration [7, 22] and a number of observations can be made: first, the smoothing factors for both RJ smoothers deteriorate with increasing problem dimension, a trend that is also observed for the LEX-GS method. Second, $M = 3$ RJ smoothers provides greater smoothing than the corresponding $M = 2$ schemes. Direct comparison of the smoothing factor for the RJ smoothers with the LEX-GS seems to indicate that both the $M = 2$ and $M = 3$ RJ smoothers have smoothing properties that exceed those of the corresponding LEX-GS iterations. However, this direct comparison does not account for the fact that the LEX-GS only consists of a single iterations whereas the RJ smoothers require 2 or 3 iterations.

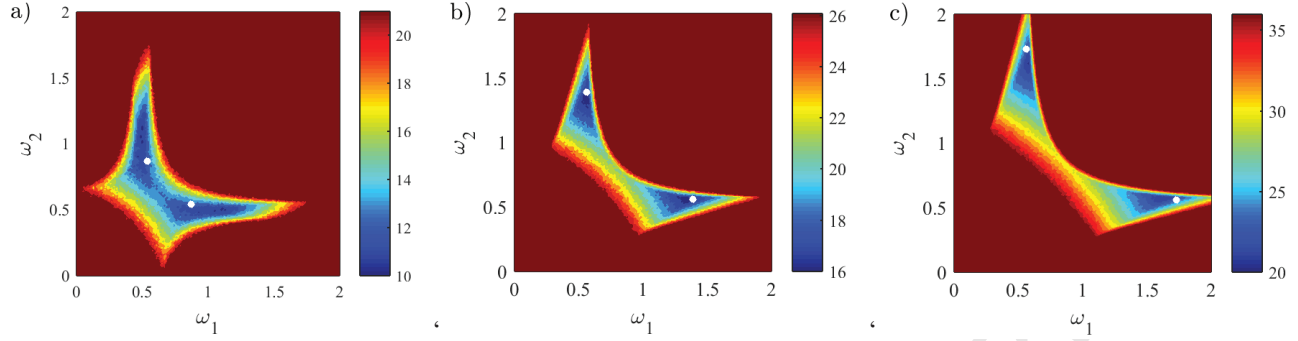


Figure 2: Multigrid cycles needed to reduce the initial residual by 10 orders-of-magnitude for with the $M = 2$ RJ smoothers with the relaxation factors ω_1, ω_2 swept from 0 to 2 with a step-size of 0.01. The initial field is randomly generated from a uniform probability distribution with no spatial correlation. We truncate at the number of multigrid cycles needed by point LEX-GS based multigrids (see section 3). The problem dimension is $d = 1$ to $d = 3$ from left to right. The optimal relaxation factors listed in table 1 are indicated with white circles in the plots.

In order to account for this, we estimate the smoothing-per-iteration for the RJ smoothers as $\nu'_{RJ} = \nu_{RJ}^{(1/M)}$ and compare this with the smoothing factor for the LEX-GS scheme. This comparison shows that for all except the $M = 2; d = 3$ smoother, the RJ smoothers provide smoothing factors that are superior to the LEX-GS iteration. For the $M = 2; d = 3$ RJ smoother, the smoothing factor ν'_{RJ} is just marginally (about 3%) inferior to the LEX-GS iteration. This analysis therefore suggests that even on a single processor, the RJ smoothers should be generally more efficient than the LEX-GS smoother. This issue is explored further in the next section.

3. Performance of RJ-Based Smoothers

In this section the performance of multigrid solvers with RJ smoothers is assessed for both structured grids that employ geometric multigrid, as well as unstructured grids that employ algebraic multigrid methods. The relaxation factors employed in these numerical “experiments” are the ones tabulated in table 1.

3.1. Geometric Multigrid on Structured Grids

Before conducting a detailed assessment of the RJ smoothers derived here, we present evidence that the procedure employed to derive the optimal RJ smoothers does generate smoothers within the framework of the SRJ methodology that maximize convergence. This is done by implementing the relaxation procedure in a typical geometric-multigrid method with a fully coarsened “V-cycle” on uniform Cartesian grids. Coarsening is always carried down to a level that admits no further coarsening. A multigrid cycle begins with a smoothing operation on the finest grid and ends with a prolongation operation to the finest grid. Linear (bilinear and trilinear in 2D and 3D respectively) interpolation is used for both restriction (coarsening) and prolongation. These initial tests are on a single domain implementation.

For $M = 2$, we solve the model problem (Eq. 1) computationally for values of ω_1 and ω_2 ranging from 0 to 2 and plot the number of multigrid cycles required to reduce the residual by 10 orders-of-magnitude. The initial guess for all cases is chosen to be a random field in the range $[-1, 1]$ and the results shown are for a grid with 32 points in each dimension. Because the convergence rate of a multigrid method is in-principle, independent of the problem size, the results here are meaningful to problems of all grid sizes. The numerical tests therefore indicate that there is indeed a unique combination of ω s that maximizes the convergence of the multigrid iteration and confirm that this combination is very much consistent with the optimal values predicted by our analysis (Table 1). While not shown here, numerical tests for $M = 3$ schemes also lead to a similar conclusion.

With the optimality of the RJ smoothers derived here now confirmed, we move to comparing their computational performance to the benchmark LEX-GS smoother within the context of a geometric-multigrid implemented on a single thread. The analysis of the smoothing factors for the various smoothers shown in Table 1 suggests that even adjusted for increased iteration count per smoothing, RJ smoothers might be superior to the LEX-GS scheme, and we assess this in the current section.

Convergence results for the V-cycle multigrid implementation of the model problem (Eq. 1) on a single processor for a grid size of 128^d are shown in Table 2. Larger grid sizes show similar trends and those results are therefore not included here. The first comparison is in terms of the number of multigrid cycles required to reduce the residual down by 10 orders-of-magnitude. In figure 3, we have plotted the residual against the multigrid cycle counts for a 3D problem and it clearly shows that an asymptotic converge rate is achieved. This comparison clearly demonstrates that the RJ smoothers significantly outperform the

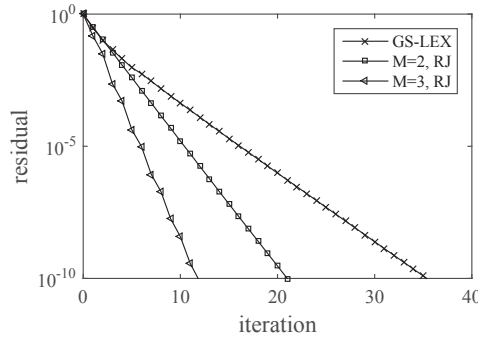


Figure 3: The residual (normalized by the initial residual) versus the multigrid cycle count for LEX-GS, and the $M = 2$ and $M = 3$ RJ smoother based multigrid for a 3D problem of size 128^3 .

Table 2: Multigrid cycles and FLOPs (per grid point) needed to reduce the initial residual by 10 orders-of-magnitude in a single-thread implementation. The FLOP counts are obtained by taking the product of the multigrid cycle count and the FLOP count per cycle per grid point (listed in table 5).

d	LEX-GS		RJ $M = 2$		RJ $M = 3$	
	Cycle	FLOP	Cycle	FLOP	Cycle	FLOP
1	23	1104	12	828	10	900
2	27	1269	16	1056	12	1008
3	37	2146	22	1694	13	1248

LEX-GS in terms of this metric. The $M = 2$ RJ smoothers require roughly between 55-60% of the multigrid cycles of LEX-GS whereas the $M = 3$ RJ smoother requires only about 35-45% of the same.

The above comparison however, does not account for the fact that the RJ smoothers employ a higher FLOP count per cycle than the LEX-GS iteration, and it is ultimately the total FLOPs required to achieve a convergence target, that provide the most practical measure of convergence acceleration. In order to make such a comparison, we estimate the total FLOPs per multigrid cycle for each of the smoothers employed here (see Appendix for how the estimate is derived) and Table 2 also shows the total FLOPs per grid point needed for achieving the convergence target (i.e. 10 orders-of-magnitude reduction in initial residue) for each smoother. It can be seen that even with this metric, the RJ smoothers handily outperform the LEX-GS smoother. The number of FLOPs required by the $M = 2$ smoother for $d = 3$ is about 81% of LEX-GS where as for the $M = 3$ smoother, it is 60%.

Lastly, we assess the performance of the RJ smoothers in an implementation that mimics the presence of subdomains as in a parallel implementation on multiple processors. The grid size for this comparison is also 128^d . It is noted that domain partitioning associated with parallel implementation limits the depth of coarsening in a multigrid cycle. To eliminate this issue, the technique of full-depth amalgamated multigrid [23] is employed. When coarsening reaches the limit imposed by domain partitioning, this technique amalgamates the grid points from different processors onto a single (master) processor and allows coarsening to proceed until the grid size admits no further coarsening. Each dimension is decomposed into $P = 2, 4, 8, 32$ segments and the total partition counts involved for dimension d is P^d . Note that 32 is highest linear partition count that can be used in these tests, since the LEX-GS iteration transitions to a pure Jacobi method for $P = 64$ and the multigrid ceases to be convergent.

In Table. 3 we list the multigrid cycle counts and the FLOPs per grid point needed to reduce the initial residual by 10 orders-of-magnitude as functions of the number of partitions (P) along each dimension for 1D, 2D and 3D problems. As expected, the RJ smoothers were found to be completely insensitive to domain decomposition and their performance unchanged from

Table 3: Multigrid cycles and FLOP counts needed for LEX-GS based MG to reduce the initial residual by 10 orders of magnitude. P is domain partition per dimension and the total partition size is therefore P^d .

d	$P = 2$		$P = 4$		$P = 8$		$P = 16$		$P = 32$	
	Cycle	FLOP	Cycle	FLOP	Cycle	FLOP	Cycle	FLOP	Cycle	FLOP
1	48	2304	48	2304	50	2400	53	2544	72	3456
2	107	5029	115	5405	116	5452	119	5593	158	7426
3	193	11194	198	11484	201	11658	211	12238	335	19430

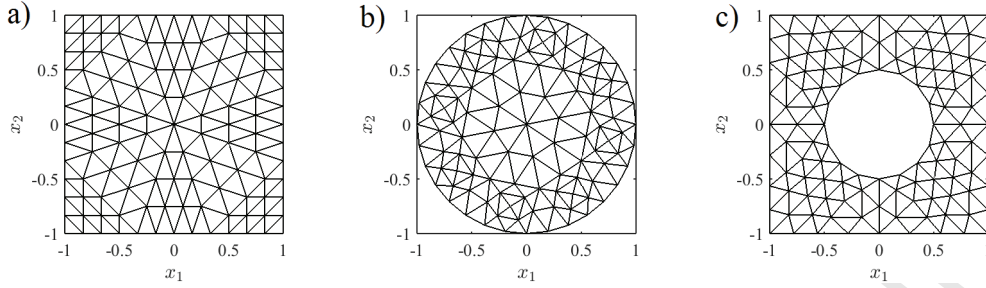


Figure 4: Topology of the unstructured grid employed for the various cases. a) square; b) circle; c) square with a circular slot of radius=0.5. The grids have nearly 100×10^3 triangular elements and a suitably coarsened mesh is shown in these figures for ease of presentation.

that for a single domain (and shown in Table 2). Furthermore, also as expected, the table indicates that the performance of LEX-GS based multigrid, both in terms of cycle count and FLOP count, deteriorates noticeably with increase in partition count. In particular, the largest deterioration occurs in going from one to two partitions for all the cases. For 3D problems, which are particularly relevant for large-scale parallel implementations, we find a factor of over 4 increase in cycle and FLOP count in going from $P = 1$ to 2. For P beyond 2, the computational performance of LEX-GS diminishes only very slowly until $P = 32$, where again, there is a noticeable (factor of nearly 1.5) further increase in cycle and FLOP count.

The net result of this is that the LEX-GS based multigrid not only starts with a significant performance deficit for a single domain when compared to the RJ smoothers, this deficit in performance amplifies significantly with partition count. For 3D problems with $P = 2$, LEX-GS requires a factor of 6.6 more FLOPs than the $M = 2$ RJ smoother, and nearly a factor of 9.0 more FLOPs than the $M = 3$ scheme. For $P = 32$ these factors increase to 11.5 and 15.6 for the two smoothers respectively.

3.2. Algebraic Multigrid on Unstructured Grids

In this section we provide some preliminary evidence that the RJ smoothers also offer an advantage for algebraic multigrid methods on unstructured grids. For this purpose, the model problem (Eq. 1) is discretized with a standard 2nd-order finite-element method (FEM) [24] and is solved on the 2D domains shown in figure 4. The first is a simple square domain; the second is a circular domain, and the third is a square domain with a circular slot in the center. The computational domain is discretized using triangular elements and homogeneous Dirichlet boundary conditions are imposed on all boundaries. We employ a relatively large grid with about 1×10^5 elements in these tests. It is noted at the outset that the optimal relaxation factors determined for the RJ smoothers in Sec. 2 are based on a second-order finite-difference scheme on a structured mesh, which employs a 5-point stencil for 2D problems. A second-order FEM on a triangular mesh, on the other hand, employs a 4-point stencil. Therefore the RJ smoothers derived in Sec. 2 for a structured grid are not necessarily “optimal” for the FEM discretization used here. Because it is not entirely clear how the analysis in Sec. 2 could be extended to grids with triangular elements, we simply use the RJ smoother that was derived for 2D problems in Sec. 2 in the current tests without any modification. The model problem is then solved with a multi-thread implementation (P extending from 1 to 1024) using the LEX-GS based multigrid as well as the $M = 2$ and 3 RJ smoother based multigrids. The technique of full-depth amalgamation is used here also. For LEX-GS, we also follow the established practice of reordering the equations to maximize diagonal dominance. This re-ordering can be computationally expensive, especially for adaptive meshes, leading to increased computational cost of LEX-GS based multigrid. Jacobi iteration, on the other hand, are insensitive to the ordering of the equations.

In table 4 the MG cycles and FLOPs needed to reduce the initial residual by 10 orders-of-magnitude are listed. The estimate for the FLOP count is described in the Appendix. A number of interesting trends emerge from this table. First, for $P = 1$, the RJ smoothers appear to provide some improvement in the convergence rate per multigrid cycle. This improvement is marginal for the $M = 2$ smoother but is more substantial for the $M = 3$ smoother, which is found to converge in about half the cycles of the LEX-GS based multigrid for two cases. In terms of the FLOP count however, the $M = 2$ RJ smoother is found to be noticeably inferior to the LEX-GS for all cases, where as the $M = 3$ RJ smoother is only marginally inferior to LEX-GS.

With multi-domain implementation, however, the advantage of the RJ smoothers becomes apparent. As expected, we find that the RJ based multigrids are completely insensitive to domain partitioning. The performance of LEX-GS based multigrid, however, exhibits a trend that is similar to that observed for the structured grids. First, there is a significant (nearly a two-fold) increase in cycle count in going from $P = 1$ to 2. Beyond that, the cycle count increases slowly with P up to about $P = 128$ beyond which, it starts to increase more rapidly. For $P = 1024$ the cycle count has increased by factors ranging from about 3 to 5.3 for the three cases tested here, and the gains in terms of the FLOP count also have a similar magnitude.

Table 4: Multigrid cycles and FLOP counts (per grid point) needed for $M = 2, 3$ RJ and LEX-GS based MG to reduce the initial residual by 10 orders of magnitude. P is the thread count.

	RJ				LEX-GS									
	$M = 2$		$M = 3$		$P = 1$		$P = 2$		$P = 32$		$P = 128$		$P = 1024$	
	Cycle	FLOP	Cycle	FLOP	Cycle	FLOP	Cycle	FLOP	Cycle	FLOP	Cycle	FLOP	Cycle	FLOP
Square	18	828	10	610	20	620	41	1271	48	1488	54	1674	72	2232
Circle	15	690	12	732	19	558	44	1364	60	1860	63	1953	101	3131
Slot	14	644	9	549	17	527	35	1085	41	1271	43	1333	55	1705

4. Conclusions

The iterative solution of discretized elliptic equations is central to computational mechanics and in the era of GPU computing, the widespread use of processor/thread counts exceeding $O(10^3)$ in scientific computing has resulted in a premium on methods that provide effective convergence acceleration on such large partitions. We have presented a family of relaxed Jacobi-based smoothers for multigrid methods that, unlike the standard Gauss-Seidel based smoothers, provide convergence properties that are undiminished by domain decomposition. Furthermore, the overall computational efficiency of geometric multigrid with these smoothers, measured in terms of total FLOP count for convergence, significantly exceeds that of the lexicographic point Gauss-Seidel smoother. Additionally, the point-iterative nature of the new smoothers makes them applicable as well as effective for use in conjunction with unstructured multigrid solvers, although the efficiency enhancement for the few unstructured grid cases tested here, is not as high as for the structured grid cases. In the case of unstructured grids however, it is noted that the RJ smoothers obviate the need for reordering of the equations as is mandatory for the point Gauss-Seidel smoother, and this could further improve the computational efficiency afforded by the new smoothers.

There are a few limitations of the current work. First, the FLOP count estimate employed here for the different smoothers is not exact and might vary somewhat based on the details of the implementation. Counting all floating point operations (such as multiplications and additions) equally, as has been done here, might incorrectly estimate the actual processing time required for the calculations. Finally, due to the use of multiple successive iterations required at each smoothing step, the Jacobi-based smoothers presented here require additional inter-processor communications, and this is sure to impact the parallel scalability of these methods and the overall turn-around time required for a solution. However, all of these factors are best addressed by the wider scientific computing community by implementing and testing these smoothers on a wide variety of platforms. Finally, while the application of the RJ smoothers to unstructured grids with triangular elements seems to confirm their benefits, formal derivation of optimal RJ smoothers for such grid topologies remains to be addressed.

Appendix

We provide the details of estimating the multigrid cycle FLOP count on a 2D grid with a LEX-GS smoother. The smoothing step, in its most general form involves the following calculation: $u_{i,j}^{n+1} = 0.25u_{i-1,j}^{n+1} + 0.25u_{i,j-1}^{n+1} + 0.25u_{i+1,j}^{n+1} + 0.25u_{i,j+1}^{n+1} + 0u_{i,j} + \text{const} \cdot f_{i,j}$. For generality such as in the case of non-uniform grids, the stencil weight of 0.25 is not factored out. Furthermore, the $u_{i,j}$ term is retained as in the case of a relaxed iteration as is the source term. Finally, for this FLOP estimate, we do not distinguish between add and multiplication operations. For a V-cycle lead, the above calculation leads to $11 + 2 \times (11/4^1 + 11/4^2 + 11/4^3 \dots) = 18.3$ FLOPs per grid point where the factor of $1/4$ accounts for coarsening in both directions and the factor of 2 for the two legs of a V-cycle. For the sake of simplification, we assume a infinite number of coarsenings in the current estimate, but this generates a very small error for the 6 or more level of coarsening employed in the structured grid test cases. For $M=2(3)$ RJ smoother, which employs two(three) successive iterations, we simply double(triple) this FLOP count.

The FLOP counts of the other elements of the V-cycle are the same irrespective of the smoother. For restriction, the calculation of the residual requires $11 + 11/4 + 11/4^2 + \dots = 14.7$ FLOPs and the restriction operation (bilinear interpolation) itself takes $7/4 + 7/4^2 + \dots = 2.3$ FLOPs per grid point per cycle. The restriction step therefore requires a total of 17 FLOPs per grid point per cycle. Similarly, it can be shown that prolongation requires 12 FLOPs per grid point per cycle. Operations needed to enforce boundary conditions are neglected. Adding up all the FLOPs leads to 47 FLOPs per grid point per cycle for LEX-GS and Table 5 lists the FLOP counts for 1D and 3D problems, as well as for the RJ smoothers.

The same exercise is done for the algebraic multigrid solver assuming a grid with triangular elements and a 4-point stencil. We employ a direct fill-in for the prolongation, and the resulting FLOP count estimates are also listed in table 5.

Table 5: FLOP needed for each operation in a V-cycle. The multigrid solver is used to solve Eq. 1 on a square domain using structured grids. $d = 1, 2, 3$ is the problem dimension.

	Geometric MG			Algebraic MG
	1D	2D	3D	2D
LEX-GS	48	47	58	31
$M = 2$ RJ	69	66	77	46
$M = 3$ RJ	90	84	96	61

Acknowledgement

XY was supported by ONR grant N00014-12-1-0582. RM would like to acknowledge support from ONR as well as NSF (grants IOS1124804, IIS1344772 and OISE 1243482).

Reference

- [1] W. P., Linear multigrid methods, multigrid methods S. F. McCormick (ed.) (Frontiers in Applied Mathematics 3), SIAM, Philadelphia.
- [2] W. K. Briggs, S. F. McCormick, Introduction, multigrid methods S. F. McCormick (ed.) (Frontiers in Applied Mathematics 3), SIAM, Philadelphia.
- [3] P. Wesseling, an introduction to multigrid methods, John Wiley & Sons, New York, 1991.
- [4] A. Brandt, Multi-level adaptive technique (MLAT) for fast numerical solution to boundary value problems, proc. 3rd int. conf. on numerical methods in fluid mechanics, vol.1, H. Cabannes and R. Temam (eds) (1973) 82–89.
- [5] A. Brandt, Multi-level adaptive solutions to boundary value problems, Math. Comput. 31 (1973) 333–390.
- [6] S. R. Fulton, P. E. Ciesielski, W. H. Schubert, Multigrid methods for elliptic problems: A review, Monthly Weather Review 114 (5) (1986) 943–959.
- [7] U. Trottenberg, C. W. Oosterlee, A. Schuller, Multigrid, Academic press, 2000.
- [8] K. Stüben, A review of algebraic multigrid, Journal of Computational and Applied Mathematics 128 (1) (2001) 281–309.
- [9] Y. Kaneda, T. Ishihara, High-resolution direct numerical simulation of turbulence, Journal of Turbulence (7) (2006) N20.
- [10] A. Lozano-Durán, J. Jiménez, Effect of the computational domain on direct simulations of turbulent channels up to $Re\tau = 4200$, Physics of Fluids (1994-present) 26 (1) (2014) 011702.
- [11] A. Brandt, Multiscale scientific computation: Review 2001, in: Multiscale and multiresolution methods, Springer, 2002, pp. 3–95.
- [12] D. Wallin, H. Löf, E. Hagersten, S. Holmgren, Multigrid and Gauss-Seidel smoothers revisited: parallelization on chip multiprocessors, in: Proceedings of the 20th annual international conference on Supercomputing, ACM, 2006, pp. 145–155.
- [13] M. F. Adams, A distributed memory unstructured Gauss-Seidel algorithm for multigrid smoothers, in: Supercomputing, ACM/IEEE 2001 Conference, IEEE, 2001, pp. 14–14.
- [14] D. Braess, The convergence rate of a multigrid method with Gauss-Seidel relaxation for the poisson equation, Mathematics of computation 42 (166) (1984) 505–519.
- [15] K. Sørensen, O. Hassan, K. Morgan, N. Weatherill, Agglomerated multigrid on hybrid unstructured meshes for compressible flow, International journal for numerical methods in fluids 40 (3-4) (2002) 593–603.
- [16] M. Adams, M. Brezina, J. Hu, R. Tuminaro, Parallel multigrid smoothing: polynomial versus Gauss-Seidel, Journal of Computational Physics 188 (2) (2003) 593–610.
- [17] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, Numerical Recipes: The Art of Scientific Computing (3rd edition.), New York: Cambridge Univ. Press, 2007.
- [18] X. I. A. Yang, R. Mittal, Acceleration of the Jacobi iterative method by factors exceeding 100 using scheduled relaxation, Journal of Computational Physics 274 (2014) 695–708.
- [19] V. Babu, Determination of the optimal relaxation parameters for the solution of the Neumann–Poisson problem on uniform and non-uniform meshes using the Scheduled Relaxation Jacobi method, International Journal of Advances in Engineering Sciences and Applied Mathematics (2015) 1–10.
- [20] P. P. Pratapa, P. Suryanarayana, J. E. Pask, Anderson acceleration of the Jacobi iterative method: An efficient alternative to Krylov methods for large, sparse linear systems, Journal of Computational Physics 306 (2016) 43–54.

- [21] J. Adsuaara, I. Cordero-Carrión, P. Cerdá-Durán, M. Aloy, Scheduled relaxation Jacobi method: improvements and applications, *Journal of Computational Physics* 321 (2016) 369–413.
- [22] L. R. Hocking, C. Greif, Closed-form multigrid smoothing factors for lexicographic Gauss–Seidel, *IMA Journal of Numerical Analysis* (2011) drr037.
- [23] D. A. Jacobsen, I. Senocak, A full-depth amalgamated parallel 3D geometric multigrid solver for GPU clusters, *AIAA Paper* (2011-946).
- [24] R. D. Cook, et al., *Concepts and applications of finite element analysis*, John Wiley & Sons, 2007.