Nonsymmetric preconditioning for conjugate gradient and steepest descent methods

Henricus Bouwmeester\textsuperscript{1}, Andrew Dougherty\textsuperscript{1}, and Andrew V. Knyazev\textsuperscript{2}

\textsuperscript{1} University of Colorado Denver, USA
Henricus.Bouwmeester@ucdenver.edu and Andrew.Dougherty@ucdenver.edu
\textsuperscript{2} Mitsubishi Electric Research Laboratories, Cambridge, MA, USA
Andrew.Knyazev@merl.com, http://www.merl.com/people/knyazev

Abstract
We analyze a possibility of turning off post-smoothing (relaxation) in geometric multigrid when used as a preconditioner in preconditioned conjugate gradient (PCG) linear and eigenvalue solvers for the 3D Laplacian. The geometric Semicoarsening Multigrid (SMG) method is provided by the \textit{hypre} parallel software package. We solve linear systems using two variants (standard and flexible) of PCG and preconditioned steepest descent (PSD) methods. The eigenvalue problems are solved using the locally optimal block preconditioned conjugate gradient (LOBPCG) method available in \textit{hypre} through BLOPEX software. We observe that turning off the post-smoothing in SMG dramatically slows down the standard PCG-SMG. For flexible PCG and LOBPCG, our numerical tests show that removing the post-smoothing results in overall 40–50 percent acceleration, due to the high costs of smoothing and relatively insignificant decrease in convergence speed. We demonstrate that PSD-SMG and flexible PCG-SMG converge similarly if SMG post-smoothing is off. A theoretical justification is provided.

Keywords: linear equations, eigenvalue, iterative, multigrid, smoothing, preconditioning, convergence, nonsymmetric, conjugate gradient, steepest descent, parallel, \textit{hypre}, BLOPEX, LOBPCG

1 Introduction

Smoothing (relaxation) and coarse-grid correction are the two cornerstones of multigrid techniques. In algebraic multigrid, where only the system matrix is (possibly implicitly) available, smoothing is more fundamental since it is often used to construct the coarse grid problem. In geometric multigrid, the coarse grid is generated by taking into account the geometry of the fine grid, in addition to the chosen smoothing procedure. If full multigrid is used as a standalone solver, proper smoothing is absolutely necessary for convergence. If multigrid is used as a preconditioner in an iterative method, one is tempted to check what happens if smoothing is turned partially off, although we have actually first experienced it by accident in a class project.

* A preliminary version posted at http://arxiv.org/abs/1212.6680
For symmetric positive definite (SPD) linear systems, the preconditioner is typically required to be also a fixed linear SPD operator, to preserve the symmetry of the preconditioned system; for exceptions, see, e.g., [4, Section 12.3], [5, 15], and [17, Section 10.2]. In the multigrid context, the preconditioner symmetry is achieved by using balanced pre- and post-smoothing, and by properly choosing the restriction and prolongation pair. In order to get a fixed linear preconditioner, all multigrid components, such as smoothing, restriction, prolongation, and coarse solves, must be linear. The positive definiteness is obtained by performing enough, e.g., in practice, even one may be enough, pre- and post-smoothing steps, where the number of the pre-smoothing steps and the number of the post-smoothing steps must be equal; see, e.g., [6].

If smoothing is unbalanced, e.g., there is one step of pre-smoothing, but no post-smoothing, the multigrid preconditioner becomes nonsymmetric. Traditional assumptions of the standard convergence theory of iterative solvers are no longer valid, and convergence behavior may be unpredictable. The main goals of this paper are describing our numerical experience experimenting with the influence of unbalanced smoothing in practical geometric multigrid preconditioning, specifically, the Semiccoarsening Multigrid (SMG) method, see [16], provided by the parallel software package hypre [1] and explaining theoretically the observed interesting behavior.

We numerically analyze the possibility of turning off the post-smoothing in the geometric multigrid used as a preconditioner in iterative linear and eigenvalue solvers for the 3D Laplacian in hypre. The linear systems are solved using two variants (standard and flexible, e.g., [8]) of the preconditioned conjugate gradient (PCG) and preconditioned steepest descent (PSD) methods. The eigenvalue problems are solved using the locally optimal block preconditioned conjugate gradient (LOBPCG) method, see [9], readily available in hypre through BLOPEX [3].

We observe that turning off the post-smoothing in SMG dramatically slows down the standard PCG-SMG. However, for the flexible PCG and LOBPCG, our numerical tests show that post-smoothing can be dropped. Moreover, turning off the post-smoothing in SMG results in overall acceleration, due to the high costs of smoothing and relatively insignificant decrease in convergence speed. Our observations are also expected to be generally applicable for algebraic multigrid preconditioning, e.g., for graph Laplacians used in spectral image segmentation, as originally tested in [10] and appearing, e.g., in computational photography problems [14].

Our numerical experiments are executed in both a strictly shared memory environment and in a distributed memory environment, demonstrating that the effect of the acceleration does not depend on the memory interconnection speed. All our numerical tests use multigrid preconditioning, see also [7]; however, our theory predicts that nonsymmetric preconditioning for SPD linear systems and eigenvalue problems could be efficient in general, e.g., for domain decomposition and inexact factorization preconditioning.

A different case of non-standard preconditioning, specifically, variable preconditioning, in PCG is considered in our earlier work [11]. There, we also find a dramatic difference in convergence speed between the standard and flexible versions of PCG. The better convergence behavior of the flexible PCG is explained in [11] by its local optimality, which guarantees its convergence with at least the speed of PSD. Our numerical tests in [11] show that, in fact, the convergence of PSD is practically similar to the convergence of the flexible PCG. We perform the same comparison in this work, and obtain analogous results. We demonstrate for SPD linear systems that PSD-SMG converges almost as fast as the flexible PCG-SMG method, while the standard PCG-SMG method stalls, if the SMG post-smoothing is off.

The rest of the paper is organized as follows. We formally describe the PSD and PCG methods used for testing in this work, and explain their differences in Section 2. In Section 3, we briefly discuss the SMG preconditioning in hypre and present our numerical results for linear systems. Section 4 deals with eigenvalue problems. Section 5 contains some relevant theory.
2 PSD and PCG methods for linear systems

For a general exposition of PSD and PCG, let SPD matrices $A$ and $T$, and vectors $b$ and $x_0$ be given, and denote $r_k = b - Ax_k$. Algorithm 1 is described in [11].

Algorithm 1: PSD and PCG methods

1. for $k = 0, 1, \ldots$ do
2. \hspace{1em} $s_k = Tr_k$
3. \hspace{1em} if $k = 0$ then
4. \hspace{2em} $p_0 = s_0$
5. \hspace{1em} else
6. \hspace{2em} $p_k = s_k + \beta_k p_{k-1}$ (where $\beta_k$ is either (1) or (2) for all iterations)
7. \hspace{1em} end
8. \hspace{1em} $\alpha_k = \frac{(s_k, r_k)}{(p_k, Ap_k)}$
9. \hspace{1em} $x_{k+1} = x_k + \alpha_k p_k$
10. \hspace{1em} $r_{k+1} = r_k - \alpha_k Ap_k$
11. end

Various methods are obtained by using different formulas for the scalar $\beta_k$. We set

$$\beta_k = \frac{(s_k, r_k)}{(s_{k-1}, r_{k-1})} \tag{1}$$

for the standard PCG, or

$$\beta_k = \frac{(s_k, r_k - r_{k-1})}{(s_{k-1}, r_{k-1})} \tag{2}$$

for the flexible PCG, or $\beta_k = 0$ for PSD.

We note that in using (2), we are merely subtracting one term, $(s_k, r_{k-1})$, in the numerator of (1), which appears in the standard CG algorithm. If $T$ is a fixed SPD matrix, this term in fact vanishes; see, e.g., [11]. By using (2) in a computer code, it is required that an extra vector be allocated to either calculate $r_k - r_{k-1}$ or store $-\alpha_k Ap_k$, compared to (1). The associated cost increase may be noticeable for large problems solved on parallel computers. To measure the actual increase, we numerically evaluate the standard and flexible PCG with no preconditioning for a variety of problem sizes.

Our model problem used for all calculations in the present paper is for the three-dimensional negative Laplacian in a brick with homogeneous Dirichlet boundary conditions approximated by the standard finite difference scheme using the 7-point stencil with the grid size one in all three directions. The number of the grid points varies in our tests, resulting in the size of the matrix $A$ in the range from sixteen thousand to over two billion; see Appendix A for details.

For the CG method without preconditioning, we observe a 20-25% cost overhead per iteration incurred due to the extra storage and calculation for the flexible variant, (2), relative to the standard variant, (1) over the whole range of the matrix sizes and in both shared and distributed memory tests. For PCG, the cost overhead of using formula (2) vs. (1) per iteration is smaller, because of the additional costs of the application of the preconditioner. Specifically for SMG preconditioning, described in the next section, the cost overhead per iteration is negligible.

Nonsymmetric preconditioning for CG and SD

Bouwmeester, Dougherty, and Knyazev

278
3 Preconditioning linear systems with SMG

The CG method can be accelerated by preconditioning. We use the SMG solver as a preconditioner, provided by hypre. The SMG solver/preconditioner uses plane-relaxation as a smoother at each level in the V-cycle; see [1]. The SMG preconditioner is fixed and linear, according to [16], and uses pre- and post-relaxation smoothing steps. In our tests, we either use the “balanced relaxation,” i.e. one step of pre-relaxation and one step of post-relaxation, resulting in an SPD preconditioner, as required by PCG, or “no post-relaxation,” still with one step of pre-relaxation, but turning off the post-relaxation smoothing, thus making the SMG preconditioner nonsymmetric, violating the standard PCG assumptions.

We present results of numerical experiments, preconditioning linear systems for the three-dimensional negative Laplacian using SMG. In all timing and iteration count figures below, the horizontal axis represents the number \( n \) of grid points in each of the three directions per processor. The size of the matrix is thus \( np \times n \times n \times n \), where \( np \) denotes the number of processors (cores), e.g. \( n = 180 \) gives the problem size \( 16 \times 180 \times 180 \times 180 = 93,312,000 \) for \( np = 16 \) on one node and \( 384 \times 180 \times 180 \times 180 = 2,239,488,000 \) for \( np = 384 = 16 \times 24 \) on 24 nodes with 16 processors (cores) on each node, using distributed memory.

![Figure 1](image1.png)

(a) Convergence on 16 processors on 1 node, \( n = 80 \)

(b) Iteration count on 16 processors on 1 node

Figure 1: Convergence and iteration count for PCG-SMG using (1) or (2) and PSD-SMG

Figure 1 displays the convergence history (left panel, Figure 1a, where \( n = 80 \)) and the iteration count (right panel, Figure 1b) for PSD-SMG and PCG-SMG using (1) or (2) with and without balanced SMG relaxation on 16 processors on 1 node. If no post-relaxation is performed within the SMG preconditioner, the convergence of the standard PCG method, i.e. with (1), is dramatically slowed down, compared to PCG-SMG using (2) and even to PSD-SMG, as demonstrated in Figure 1a. We thus drop the standard PCG method with (1) with no SMG post-relaxation from further consideration, as evidently non-competitive, and plot the iteration count reaching the default tolerance for other methods in Figure 1b. Specifically, Figure 1b compares the iteration count of PSD-SMG and PCG-SMG using (2), both with no post-relaxation, with the ground truth, PCG-SMG with the same number of pre- and post-relaxation steps, one, in SMG that creates an SPD preconditioner, and makes formulas (1) and (2) mathematically equivalent generating the identical iteration count in Figure 1b. We observe in Figure 1b a 50 (60) percent slowdown of PCG-SMG using (2) (PSD-SMG) with no post-relaxation, compared to PCG-SMG with (1) or (2) and the balanced relaxation.
The surprising convergence behavior in Figure 1 is similar to that observed in [11], where a variable SPD preconditioner makes the standard PCG, i.e. using (1), almost stall, while the convergence rates of PSD and flexible PCG, i.e. with (2), are good and close to each other. However, the SMG preconditioner is fixed and linear, according to its description in [16] and our numerical verification, in contrast to the variable preconditioner in [11]. Moreover, turning off the post-relaxation smoothing in the multigrid preconditioner makes it nonsymmetric—the case not covered in [11], where the assumption is made that the preconditioner is SPD. We revisit the theoretical arguments of [11] in Section 5, and find that the SPD preconditioner assumption in [11] is actually never significantly used and can be dropped, as in [17, Section 10.2], providing theoretical justifications for the unexpected numerical results displayed in Figure 1.

Our next numerical results are even more amazing. Figure 2 compares the CPU timing of the flexible PCG-SMG using (2) with no post-relaxation with the ground truth, which is PCG-SMG with the same number of pre- and post-relaxation steps, one, in SMG that creates an SPD preconditioner. PCG-SMG with (1) and PCG-SMG with (2), both with the balanced relaxation, generate nearly the same iterative approximations, as expected, since they are mathematically equivalent (up to round-off errors) due to the fact that the only different term \((s_{k}, r_{k-1})\) between (1) and (2) is known to vanish if the preconditioner in SPD. Although using (2) introduces some extra timing overhead, in the case of the SMG preconditioning the overhead is negligible, as can be seen in Figure 2, where the timing curves CG-SMG with (1) and with (2) are nearly indistinguishable, since the SMG preconditioning, first, is relatively expensive computationally and, second, its high quality makes the PCG method converge fast.

The results displayed in Figure 2 are easy to explain, knowing that the relaxation step in SMG is the main contributor to the computational costs of applying the SMG preconditioner on every iteration. Turning off the post-relaxation and keeping one step of pre-relaxation, on the one hand, cuts almost in half the time of application of the SMG preconditioner per iteration. On the other hand, as seen in Figure 1b, it results in 50 percent slowdown of flexible PCG-SMG using (2), not enough to outweigh the double cost reduction of not using the post-relaxation in the SMG preconditioner. The overall time saving is thus expected to be approximately 75 percent, corresponding to the improvement of approximately 40 percent, actually observed in Figure 2 for all problem sizes tested both for shared and distributed memory.
4 Preconditioning of eigenvalue problems with SMG

The LOBPCG method [9] computes the $m$ smallest eigenvalues of a linear operator and is implemented within *hypre* via BLOPEX; see [3]. We conclude our numerical experiments with a comparison of the use of balanced vs. unbalanced relaxation in the SMG preconditioner for the LOBPCG method with $m = 1$, keeping the same setup as in the previous section. Figure 3 top panels display the numbers of LOBPCG iterations for the non-balanced SMG preconditioner being only 0–40% more than that for the balanced SMG preconditioner. Thus, the cost savings of not using the post-relaxation in the SMG preconditioner may lead to over 50% acceleration, compared to the conventional balanced relaxation, as Figure 3 bottom panels indeed show.

LOBPCG is locally optimal, requiring no changes in the code to handle nonsymmetric preconditioning, even though the existing LOBPCG convergence theory in [9, 13] assumes an SPD preconditioner $T$. We explain the convergence for a nonsymmetric $T$ in the next section.
5 Nonsymmetric preconditioning theoretical justification

For linear systems with SPD coefficient matrices, the use of nonsymmetric and variable preconditioning in PSD and PCG-like methods has been justified, e.g., in [4, Section 12.3], [5, 15], and [17, Section 10.2]. Many key properties of Krylov-type methods are lost, such as orthogonality and global optimality. However, it is shown in [17, Section 10.2] that the flexible PCG, i.e. using (2), is locally optimal, i.e. on every step it converges not slower than PSD, no matter whether the preconditioner is SPD or not. The convergence rate bound for PSD, where $\beta_k = 0$, which thus also holds for flexible PCG, with $\beta_k$ given by (2), is established in [17, Section 10.2],

$$\|r_{k+1}\|_{A^{-1}} \leq \delta \|r_k\|_{A^{-1}},$$

under the assumption that the preconditioner $T$, which is non necessarily SPD, satisfies

$$\|I - AT\|_{A^{-1}} \leq \delta < 1,$$

where $\cdot \|_{A^{-1}}$ denotes the operator norm induced by the corresponding vector norm $\sqrt{x^t A^{-1} x}$.

The key identity for PSD that can help deriving (3) is presented in the following theorem.

**Theorem 1.** If $\beta_k = 0$ in Algorithm 1, then the identity holds,

$$\|r_{k+1}\|_{A^{-1}}/\|r_k\|_{A^{-1}} = \sin (\angle_{A^{-1}} \{r_k, AT r_k\}),$$

where the angle in the right-hand side is defined via

$$\cos (\angle_{A^{-1}} \{r_k, AT r_k\}) = \frac{|(r_k)^t T r_k|}{\|r_k\|_{A^{-1}}\|AT r_k\|_{A^{-1}}}.$$  

**Proof.** Identity (5) is already actually proved, although not explicitly formulated, in the proof of [17, Theorem 10.2]. Alternatively, substituting $r_k = A e_k$, identity (5) is equivalent to

$$\|e_{k+1}\|_A/\|e_k\|_A = \sin (\angle_A \{e_k, T A e_k\}),$$

which is the statement of [11, Lemma 4.1]. Reference [11] assumes that the preconditioner $T$ is SPD, but this assumption is never, in fact, used in the proof of [11, Lemma 4.1].

Assumption (4) is simple, but has one significant drawback—it does not allow arbitrary scaling of the preconditioner $T$, while the PCG and PSD methods are invariant with respect to scaling of $T$. The way around it is to scale the preconditioner $T$ before assumption (4) is verified. We now illustrate such a scaling under an additional assumption that $T$ is SPD, following [11], first connecting assumption (4) with its equivalent and arguably more traditional form.

**Theorem 2.** Let the preconditioner $T$ be SPD. Then assumption (4) is equivalent to

$$\|I - TA\|_{T^{-1}} \leq \delta < 1.$$  

**Proof.** Since $T$ is SPD, on the one hand, the matrix product $AT$ is also SPD, but with respect to the $A^{-1}$ scalar product. This implies that assumption (4) is equivalent to the statement that $\Lambda(AT) \in [1 - \delta, 1 + \delta]$ with $\delta < 1$, where $\Lambda(\cdot)$ denotes the matrix spectrum. On the other hand, the matrix product $TA$ is SPD as well, with respect to the $T^{-1}$ scalar product. Thus, assumption (6) is equivalent to the statement that $\Lambda(TA) \in [1 - \delta, 1 + \delta]$. This means the equivalence of assumptions (4) and (6), since $\Lambda(AT) = \Lambda(TA)$. 

282
Let us now, without loss of generality, as in [12, p. 96] and [11, pp. 1268–1269], always scale the SPD preconditioner $T$ in such a way that $\max\{\Lambda(TA)\} + \min\{\Lambda(TA)\} = 2$. Then we have $\delta = (\kappa(TA) - 1)/(\kappa(TA) + 1)$ and, vice versa, $\kappa(TA) = (1 + \delta)/(1 - \delta)$, where $\kappa(\cdot)$ denotes the matrix spectral condition number. The convergence rate bound (3) for the PSD with nonsymmetric preconditioning in this case turns into the standard PSD convergence rate bound for the case of SPD preconditioner $T$; see, e.g., [11, Bound (1.3)]. Moreover, [11, Theorem 5.1] shows that this convergence rate bound is sharp for PSD, and cannot be improved for flexible PCG, i.e. using (2), if the SPD preconditioner $T$ changes on every iteration. The latter result naturally extends to the case of nonsymmetric preconditioning of [17, Section 10.2].

Compared to linear systems, eigenvalue problems are significantly more complex. Sharp convergence rate bounds for symmetric eigenvalue problems have been obtained in the last decade, and only for the simplest preconditioned method; see [12, 13] and references therein. A possibility of using nonsymmetric preconditioning for symmetric eigenvalue problems has not been considered before, to our knowledge. However, our check of arguments of [12] and preceding works, where a PSD convergence rate bound is proved assuming (4) and SPD preconditioning, surprisingly reveals that the latter assumption, SPD, is actually never significantly used, and can be dropped without affecting the bound; see our idea implemented in details in [2].

The arguments above lead us to a surprising determination that whether or not the preconditioner is SPD is of no importance for PSD convergence, given the same quality of preconditioning, measured by (4) after preconditioner prescaling. If the preconditioner is fixed SPD then the standard PCG is the method of choice. The cases, where the preconditioner is variable or nonsymmetric, are similar to each other—the standard non-flexible PCG, using (1), stalls, while the flexible PCG, with (2), converges, due to its local optimality, but may not be much faster compared to PSD. This explains the numerical results using nonsymmetric preconditioning reported in this work, as related to results of [11] for variable SPD preconditioning.

### Conclusions

Preconditioning for linear systems and eigenvalue problems with symmetric positive definite (SPD) matrices may require extra design and computational efforts to set up and apply preconditioners that are fixed and also SPD, to fit the standard theory of the preconditioned steepest descent (PSD) and preconditioned conjugate gradient (PCG) methods. In contrast to the variable preconditioning, the nonsymmetric preconditioning for SPD linear systems and eigenvalue problems has attracted less attention. We test nonsymmetric multigrid preconditioning in hypre software. The geometric multigrid without post-relaxation is demonstrated to be surprisingly efficient as a preconditioner for locally optimal iterative methods, such as the flexible PCG for linear systems and LOBPCG for eigenvalue problems, leading to a 40–50% acceleration, compared to the standard SPD preconditioning using the balanced relaxation. Numerical and theoretical explanations are given, by comparing flexible PCG to PSD and by showing that the PSD convergence is robust even if the preconditioner is nonsymmetric.

### Acknowledgment

The authors would like to thank Rob Falgout, Van Henson, Panayot Vassilevski, and other members of the hypre team for their attention to our numerical results reported here and the Center for Computational Mathematics University of Colorado Denver for the use of the cluster. This work has partially been supported by NSF awards CNS 0958354 and DMS 1115734.
References

[1] *HYPRE User’s Manual*. Center for Applied Scientific Computing, Lawrence Livermore National Lab, 2012. URL: https://computation.llnl.gov/casc/hypre/software.html.

[2] Merico E Argentati, Andrew V Knyazev, Klaus Neymeyr, Evgueni E Ovtchinnikov, and Ming Zhou. Convergence theory for preconditioned eigenvalue solvers in a nutshell. *arXiv preprint arXiv:1412.5005*, 2014. URL: http://arxiv.org/abs/1412.5005.

[3] Merico E. Argentati, Ilya Lashuk, Andrew V. Knyazev, and Evgenii E. Ovtchinnikov. Block locally optimal preconditioned eigenvalue solvers (BLOPEX) in HYPRE and PETSC. *SIAM J. Sci. Comput.*, 29(4):2224–2239, 2007. doi:10.1137/060661624.

[4] Owe Axelsson. *Iterative solution methods*. Cambridge University Press, Cambridge, 1994.

[5] Radim Blaheta. GPCG-generalized preconditioned CG method and its use with non-linear and non-symmetric displacement decomposition preconditioners. *Numer. Linear Algebra Appl.*, 9(6-7):527–550, 2002. doi:10.1002/nla.295.

[6] James H. Bramble and Xuejun Zhang. The analysis of multigrid methods. In P.G. Ciarlet and J.L. Lions, editors, *Solution of Equation in $R^n$ (Part 3), Techniques of Scientific Computing*, volume 7 of *Handbook of Numerical Analysis*, pages 173–415. Elsevier, 2000. URL: http://www.sciencedirect.com/science/article/pii/S1570865900070034, doi:10.1016/S1570-8659(00)07003-4.

[7] Laurent Debreu, Emilie Neveu, Ehouarn Simon, François-Xavier Le Dimet, and Arthur Vidard. Multigrid solvers and multigrid preconditioners for the solution of variational data assimilation problems. October 2013. URL: https://hal.inria.fr/hal-00874643.

[8] Gene H. Golub and Qiang Ye. Inexact preconditioned conjugate gradient method with inner-outer iteration. *SIAM J. Sci. Comput.*, 21(4):1305–1320, 1999. URL: http://link.aip.org/link/?SCE/21/1305/1, doi:10.1137/S1064827597323415.

[9] Andrew V. Knyazev. Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. *SIAM J. Sci. Comput.*, 23(2):517–541, 2001. URL: http://link.aip.org/link/?SCE/23/517/1, doi:10.1137/S1064827500366124.

[10] Andrew V. Knyazev. Modern preconditioned eigensolvers for spectral image segmentation and graph bisection. In *Workshop on Clustering Large Data Sets*. Third IEEE International Conference on Data Mining (ICDM 2003), 2003. URL: http://math.ucdenver.edu/~aknyazev/research/conf/ICDM03.pdf.

[11] Andrew V. Knyazev and Ilya Lashuk. Steepest descent and conjugate gradient methods with variable preconditioning. *SIAM J. Matrix Anal. Appl.*, 29(4):1267–1280, 2007. URL: http://link.aip.org/link/?SML/29/1267/1, doi:10.1137/060675290.

[12] Andrew V. Knyazev and Klaus Neymeyr. A geometric theory for preconditioned inverse iteration. III. A short and sharp convergence estimate for generalized eigenvalue problems. *Linear Algebra Appl.*, 358:95–114, 2003. doi:10.1016/S0024-3795(01)00461-X.

[13] Andrew V. Knyazev and Klaus Neymeyr. Gradient flow approach to geometric convergence analysis of preconditioned eigensolvers. *SIAM J. Matrix Anal. Appl.*, 31(2):621–628, 2009. doi:10.1137/080727567.
A Hypre drivers used for testing

We use the code called struct with the -solver 10 option provided in hypre [1] to test SMG, with different command-line options, executing the following command,

\texttt{mpiexec -np 16 ./struct -n $n \$n \$n -solver 10}

for the shared memory experiments and

\texttt{mpiexec -np 384 ./struct -n $n \$n \$n -solver 10}

for the distributed memory experiments, where $n$ runs from 10 to 180, and represents the number of grid points in each of the three directions per processor. The size of the brick here is $np$-times-$n$-by-$n$-by-$n$, i.e. the brick gets longer in the first direction with the increase in the number of cores. For example, using the largest value $n=180$, the maximum problem size we solve for $np=16$ is $16 \times 180$-by-$180$-by-$180=93,312,000$ unknowns and for $np=384$ we get $384 \times 180$-by-$180$-by-$180=2,239,488,000$ unknowns. The option -solver 10 tells the driver struct to use the SMG preconditioning. The MPI option -np 16 means that we run on 16 cores and we restrict to using only one node for the shared memory whereas for distributed memory we use 16 cores on 24 nodes with the MPI option -np 384. In fact, all our tests in this paper are performed on either 16 cores on one node or 16 cores on each of the 24 nodes, so in the rest of the appendix we omit the “mpiexec -np 16(384)” part of the execution command for brevity. The standard PCG and LOBPCG methods are already coded in hypre. We have written the codes of flexible PCG and PSD by modifying the hypre standard PCG function.

The number of pre- and post-relaxation smoothing steps in SMG is controlled by a command line parameter specified by the -v flag in the struct test driver, e.g., one step each of pre- and post-relaxation, and one step each of pre-relaxation and no post-relaxation, correspondingly,

\texttt{./struct -n $n \$n \$n -solver 10 -v 1 1}

\texttt{./struct -n $n \$n \$n -solver 10 -v 1 0}

We generate the data for Figure 3 on eigenvalue problems with the following calls,

\texttt{./struct -n $n \$n \$n -solver 10 -lobpcg -v 1 0}

\texttt{./struct -n $n \$n \$n -solver 10 -lobpcg -v 1 1}

where $n$ runs from 10 to 120. We also use the -P option in struct, not shown above, to make the brick more even-sided for the eigenvalue problems as $np$ varies. In the shared memory experiments we use -P 4 2 2, which creates a $4n$-by-$2n$-by-$2n$ brick, and in the distributed case we take -P 8 8 6, which creates an $8n$-by-$8n$-by-$6n$ brick.