SSAI: A SYMMETRIC SPARSE APPROXIMATE INVERSE PRECONDITIONER FOR THE CONJUGATE GRADIENT METHOD

SHAKED REGEV* AND MICHAEL A. SAUNDERS†

Abstract. We propose a method for solving a Hermitian positive definite linear system $Ax = b$, where A is an explicit sparse matrix (real or complex). A sparse approximate right inverse M is computed and replaced by $\tilde{M} = (M + M^H)/2$, which is used as a left-right preconditioner in a modified version of the preconditioned conjugate gradient (PCG) method. M is formed column by column and can therefore be computed in parallel. PCG requires only matrix-vector multiplications with A and $\tilde{M}$ (not solving a linear system with the preconditioner), and so too can be carried out in parallel. We compare it with incomplete Cholesky factorization (the gold standard for PCG) and with MATLAB's backslash operator (sparse Cholesky) on matrices from various applications.

Key words. sparse matrix, iterative method, preconditioning, PCG, parallel computing, SPAI

AMS subject classifications. 65F08, 65F10, 65F50

1. Introduction. We consider a linear system $Ax = b$, where $A \in \mathbb{C}^{n\times n}$ is an explicit, sparse, Hermitian positive definite (HPD) matrix (real or complex), and $x, b \in \mathbb{C}^n$ are column vectors. Such systems are common in many fields including computational fluid dynamics, power networks, economics, material analysis, structural analysis, statistics, circuit simulation, computer vision, model reduction, electromagnetics, acoustics, combinatorics, undirected graphs, and heat or mass transfer. A reliable and efficient solution method is therefore crucial. Solving directly (for example with sparse Cholesky factorization) may be computationally prohibitive. Iterative methods are then preferred, especially when they use the sparsity structure of the matrix [12, 30]. For HPD systems, the conjugate gradient method (CG) [14] and preconditioned CG (PCG) are such methods.

In general, the rate of convergence of iterative methods for solving $Ax = b$ depends on the condition number of A and the clustering of its eigenvalues. Preconditioning requires a matrix $M \approx A^{-1}$ such that $AM \approx I$ or $MA \approx I$ [3]. The transformed systems $AMy = b$, $x = My$ or $MAx = Mb$ have the same solution as $Ax = b$ but are typically better conditioned. One strategy to approximate A^{-1} is choosing a sparse M to minimize the Frobenius norm of $AM - I$ or $MA - I$ [2]. As MATLAB uses column major storage for matrices, we focus on minimizing

$$\|AM - I\|_F^2 \equiv \sum_{j=1}^{n} \|Am_j - e_j\|_2^2 , \tag{1}$$

where m_j and e_j respectively denote column j of M and I.

If we have $\tilde{M} = CC^H$ for some nonsingular C, then $\tilde{M}$ is HPD and we can precondition $Ax = b$ by solving the left-right preconditioned system

$$CAC^H y = Cb,\ \ x = C^H y. \tag{2}$$

The matrices C and C^H are not needed for PCG, just products $\tilde{M}v$ for given vectors v.

Incomplete Cholesky factorization, as implemented in MATLAB's `ichol`, is a popular method for computing a sparse triangular matrix L such that $LL^H \approx A$ (and $C = L^{-1}$ in (2)). In this case, PCG requires triangular solves with L and L^H each iteration. A typical choice for the sparsity structure of L is that of the lower triangular part of A. The cost of each PCG iteration with LL^H preconditioner is then about twice that of CG. Choosing a denser L is usually undesirable unless it substantially decreases the number of iterations. Choosing a sparser L will generally lead to

*Institute for Computational and Mathematical Engineering, Stanford University, Stanford, CA (E-mail: sregev@stanford.edu).

†Systems Optimization Laboratory, Department of Management Science and Engineering, Stanford University, Stanford, CA (E-mail: saunders@stanford.edu).

Version of October 9, 2019

Algorithm 1 Diagonal scaling of A

1: Define a diagonal matrix D with $D_{ii} = 1/\sqrt{A_{ii}}$
2: $A \leftarrow D*\texttt{tril}(A)*D$ (the strictly lower triangular part of A)
3: $A \leftarrow A + A^H + I$

more iterations, while not changing the asymptotic cost of each iteration. Therefore, we restrict our discussion to the iChol factorization [12, 19, 20] with no fill-in, and its modified version (miChol). While both versions omit elements of the exact L, miChol compensates the diagonal for the dropped elements by ensuring $Ae = LL^He$, where e is an n-vector of all 1s.

An alternative to incomplete factorization is sparse approximate inverse preconditioning (AINV in Benzi et al. [4] and SPAI in Grote and Huckle [13]). For sparse HPD systems we propose SSAI, a symmetric sparse approximate inverse preconditioner. It is an inherently parallel SPAI algorithm adapted from the left-preconditioner described for GMRES [25] in [26]. (Some other SPAI algorithms described in [7] were implemented, but they were less efficient in preliminary tests.)

The paper is organized as follows. Section 2 details the algorithm for computing our preconditioner $\tilde{M} \approx A^{-1}$ and solving $Ax = b$. Section 3 compares $\tilde{M}$ with iChol, miChol, and sparse Cholesky (via MATLAB's '\' operator) on a wide variety of matrices from different applications from the SuiteSparse Matrix Collection [9]. Section 4 discusses complexity and parallel implementation. Section 5 discusses results and future directions.

2. A PCG algorithm for HPD linear systems. For SSAI and comparisons with existing methods, we start by diagonally scaling A using Algorithm 1. Each off-diagonal entry is scaled by the square root of the product of the diagonal entries in its row and column. Thus, $A_{ii} = 1$ and $|A_{ij}| \lesssim 1$. The symmetry and unit diagonal of the scaled A are not affected by round-off. Algorithm 1 takes $O(k)$ time, where $k \equiv \texttt{nnz}(A)$ is the number of nonzeros in A. It could be parallelized (less trivially than the rest of SSAI) but its cost is negligible. The system to be solved becomes $DADy = Db$, $x = Dy$. From now on we assume $Ax = b$ has been scaled in this way.

For each column of M, if we define the residual to be $r_j = e_j - Am_j$, Algorithm 2 is essentially coordinate descent on the function $\frac{1}{2}\|r_j\|_2^2$. The main differences between Algorithm 2 and the method of [26] are application to PCG (not GMRES), simplification of the for loops, parameter selection (`lfil` and `itmax`), and the fact that the symmetrization on line 16 is mandatory instead of optional. Algorithm 2 computes the ith entry of m_j with a maximum number of nonzeros per column ($\texttt{nnz}(m_j) \leq \texttt{lfil}$). Note that on line 12 $r[i]$ becomes zero; so the next i will be different.

We must choose $\texttt{itmax} \geq \texttt{lfil}$ for SSAI to work, but not so large that the preconditioner is too expensive to compute. (In our experiments, we set $\texttt{itmax} = 2*\texttt{lfil}$.) We construct M column by column (not row by row) because this leads to efficient memory access in MATLAB.

It is known that many entries of the inverse of a sparse HPD matrix are small in magnitude [10]. Lines 6–8 of Algorithm 2 tend to generate the larger values of m_j. M itself has previously been used with GMRES, but PCG requires a Hermitian preconditioner. If M is a good right preconditioner, i.e., it minimizes (1) in some subspace, we know that M^H minimizes $\left\|M^HA - I\right\|_F$ in the same subspace, and so is a good left preconditioner. Thus, the Hermitian matrix $\tilde{M} = (M + M^H)/2$ should be a good left-right preconditioner.

If M is PD, it follows that M^H is PD and $\tilde{M}$ is HPD. However, M may not be PD. Grote and Huckle [13] prove that if $\|r_j\|_2 \equiv \|Am_j - e_j\|_2 < \epsilon$ and p denotes the maximum number of nonzeros in any column of the residual, the eigenvalues of AM lie inside a disk of radius $\sqrt{p}\epsilon$, centered around 1. As $p \leq n$, taking $\epsilon = 1/\sqrt{n}$, changing line 5 of Algorithm 2 to "While $\|r_j\|_2 \geq \epsilon$ do", and removing lines 9–11, guarantees M will be PD [26], but would be expensive.

In practice, enforcing $\|r_j\|_2 \equiv \|Am_j - e_j\|_2 < \epsilon$ would substantially increase computation time. Fortunately, Algorithm 2 usually produces an HPD $\tilde{M}$. In Algorithm 3 we use an alternative and less expensive way of ensuring $\tilde{M}$ is HPD, and this has proved effective in the numerical tests.

Algorithm 2 SSAI: left-right HPD preconditioner

1: $M \leftarrow 0_{n\times n}$
2: **for** $j \leftarrow 1$ `to` n **do**
3: $m \leftarrow 0_{n\times 1}$
4: $r \leftarrow e_j$ $(n \times 1)$
5: **for** $k \leftarrow 1$ to `itmax` **do**
6: $i =$ `indmax`$(|r|)$ (index of maximum element-wise absolute value)
7: $\delta \leftarrow r[i]$
8: $m[i] \leftarrow m[i] + \delta$
9: **if** `nnz`$(m) \geq$ `lfil` **then**
10: break;
11: **end if**
12: $r \leftarrow r - \delta * a_i$
13: **end for**
14: $m_j \leftarrow m$
15: **end for**
16: $\tilde{M} \leftarrow (M + M^H)/2$

If $\tilde{M} = CC^H$, it is natural to ask whether $CAC^H \approx I$. The eigensystem of $\tilde{M}$ proves there is a unique HPD $\tilde{C}$ such that $\tilde{M} = \tilde{C}^2$. Further, for a Hermitian A, $A\tilde{C}^2 = I \Rightarrow \tilde{C}A\tilde{C} = I$. By continuity, we can expect that minimizing $\|A\tilde{M} - I\|_F^2$ tends to minimize $\|\tilde{C}A\tilde{C} - I\|_F^2$.

For simplicity, the rest of the paper uses M in place of $\tilde{M}$.

In Algorithm 3 the computed preconditioner $\tilde{M}$ (now M) is used in a version of PCG, modified to include a matrix-vector multiplication instead of a linear system solve, since typically $M \approx A$, but in our case $M \approx A^{-1}$. This algorithm is mathematically equivalent to using CG for solving system (2). In exact arithmetic, PCG is guaranteed to converge in at most n iterations. In practice this number is usually much less than n with any reasonable preconditioner. We can therefore safely choose `itmax`$= n$, knowing that our break condition will usually be the one to stop the loop. We store our initial guess as x_0 and use dx to compute the difference from the guess accrued by the algorithm. If x_0 is a very good guess, the small steps defining dx won't be lost to round-off error.

A further safeguard is to detect indefiniteness or near-singularity in M (lines 18–20) and restart PCG with an updated x_0 and a more HPD M. The normalized inner product of r with M is $\hat{\rho} \equiv (r^H Mr)/\|r\|_2^2$. If $\hat{\rho}$ is small or negative, M must be close to singular or indefinite. In this case, we define `tolM` to be the minimal $\hat{\rho}$ that we accept, and otherwise modify M in a linear fashion to increase $\hat{\rho}$. We then update x_0 and restart PCG.

The original PCG has a solve with M instead of the matrix-vector product in line 16. The product can be parallelized (another advantage of SSAI). Benzi et al. [4] introduce a sparse approximate inverse of the Cholesky factor, again allowing a parallel product. However, their method does not parallelize the computation of the preconditioner, and should not be expected to work when Cholesky fails. (SSAI does not depend on Cholesky factorization.) Some known methods such as [26] aim to compute an HPD preconditioner, but use it on the left or right with more general methods such as GMRES, because the resulting matrix AM or MA is not guaranteed to be Hermitian. These methods are far less efficient than PCG, which is tailored for HPD matrices and has fixed storage.

If A is not already scaled, diagonal scaling $M_D = \text{diag}(A)^{-1}$ is typically far more effective than no preconditioner, but it is not "greedy enough" as M_D is much cheaper to apply than A. Each iteration costs roughly the same on large matrices, but we may neglect important information on the off-diagonals, costing valuable iterations. SSAI is the best of both worlds: it retains the advantages of a good sparse approximate HPD inverse, and allows the iterative solver to be PCG.

Algorithm 3 Modified PCG Typical parameters: $\texttt{tolM} = 10^{-2}$, $\delta = 10$

1: $r \leftarrow b - A*x_0$, $dx \leftarrow 0_{n\times 1}$
2: $p \leftarrow z \leftarrow M*r$
3: $\rho_{new} \leftarrow \texttt{real}(z^H*r)$
4: **for** $j \leftarrow 1$ `to itmax` **do**
5: $\texttt{q} \leftarrow A*p$
6: $\beta \leftarrow \texttt{real}(p^H*q)$
7: **if** $\beta \leq 0$ **then**
8: break; (A is not PD)
9: **end if**
10: $\rho \leftarrow \rho_{new}$, $\alpha \leftarrow \rho/\beta$
11: $dx \leftarrow dx + \alpha*p$
12: $r \leftarrow r - \alpha*q$
13: **if** $\|r\|_2 \,/\, \|b\|_2 < \texttt{tol}$ **then**
14: break
15: **end if**
16: $z \leftarrow M*r$
17: $\rho_{new} \leftarrow \texttt{real}(z^H*r)$, $\hat{\rho} = \rho_{new}/\,\|r\|_2^2$
18: **if** $\hat{\rho} < \texttt{tolM}$ **then**
19: Restart at line 1 with $x_0 \leftarrow x_0 + dx$, $M \leftarrow M + (\delta*(\texttt{tolM} - \hat{\rho}))*I$
20: **end if**
21: $p \leftarrow z + (\rho_{new}/\rho)*p$
22: **end for**
23: $x = x_0 + dx$

3. Numerical results. We compare the performance of SSAI (Algorithm 2) with the iChol and miChol preconditioners and the intrinsic backslash (\) solver, using MATLAB on a serial PC. For all methods, we apply Algorithm 1 to HPD matrices A to obtain a scaled matrix A with unit diagonal. We then attempt to solve $Ax = b$, where $b = Aw$ and $w^T = [1, 2, \ldots, n]/n$. The first section uses matrices from the SuiteSparse collection, and the other sections contain scale-up tests for problems submitted to SuiteSparse and also available in [23].

iChol and miChol are computed by the built-in function `ichol(A)` with `opts.type = 'nofill'` and `opts.michol = 'off'` and `'on'` respectively. We implement SSAI as in Algorithm 2 and use it with PCG as in Algorithm 3. For iChol and miChol, a simpler form of PCG replaces the product with M by two triangular solves (corresponding to the iChol factors). Backslash computes the Cholesky factors of A if possible, and then two triangular solves. Otherwise, it tries MATLAB's `ldl` function (LDL^H with possibly indefinite D) or sparse LU factorization. Note that decompositions take far more time than triangular solves. If there are multiple right-hand sides, it is best to compute the decomposition explicitly using $\tilde{A} = \texttt{decomposition(A,'chol')}$ and $x = \tilde{A}\backslash b$ for each b.

In Algorithm 2 we use $\texttt{itmax} = 2*\texttt{lfil}$ and $\texttt{lfil} = \texttt{ceil(nnz(A)/n)}$, giving $\texttt{nnz}(M) < \texttt{nnz}(A) + n$ and hence $\texttt{nnz}(M) \approx \texttt{nnz}(A)$. These values were chosen to be roughly consistent with iChol, and for the same reason of not substantially affecting the run-time of each PCG iteration. It is possible that different values would give better results, including selecting `lfil` on a column by column basis, but here we focus on the method itself and not optimizing on the edges.

In Algorithm 3 we use $\texttt{tol} = 10^{-8}$, $\texttt{tolM} = 10^{-2}$, $\delta = 10$. All final residual norms $\|b - Ax\|_2$ were verified to be below `tol`. The residual for backslash was always much closer to machine precision.

In the tables of results, miChol is omitted because when it worked, the number of iterations was very similar to that of iChol, but it was less robust, frequently encountering a non-positive pivot even when iChol worked seamlessly. iChol and '\' are also omitted if they failed. T_1 and T_2 are the times (in seconds) for Algorithms 2 and 3. For '\', only total time is reported as T_2.

The MATLAB code for Algorithms 2 and 3 is available from [24].

Table 1: Results with unmodified M

Field	**Matrix**	n	nnz	**Method**	**Itns**	T_1	T_2
Acoustics	aft01	8,205	125,567	iChol	87	1.15e−3	1.56e−1
				\	-	-	1.15e−2
				SSAI	228	2.08e+0	2.30e−1
	qa8fm	66,127	1,660,579	iChol	6	1.20e−2	1.33e−1
				\	-	-	1.71e+0
				SSAI	11	3.07e+1	1.14e−1
Circuit	G2_circuit	150,102	726,674	iChol	486	5.77e−3	1.32e+1
Simulation				\	-	-	3.84e−1
				SSAI	827	1.24e+1	1.09e+1
	G3_circuit	1,585,478	7,660,826	iChol	564	9.44e−2	2.08e+2
				\	-	-	8.41e+0
				SSAI	1132	8.79e+2	1.93e+2
Fluid	shallow_water	81,920	327,680	iChol	11	2.78e−3	1.60e−1
Dynamics				\	-	-	1.31e−1
				SSAI	16	4.12e+0	1.45e−1
	parabolic_fem	525,825	3,674,625	iChol	696	3.61e−2	7.35e+1
				\	-	-	1.57e+0
				SSAI	892	9.39e+1	7.10e+1
Computer	bundle1	10,581	770,811	iChol	22	1.88e−1	1.07e−1
Graphics/				\	-	-	1.33e−1
Vision				SSAI	25	1.42e+1	8.89e−2
	Andrews	60,000	760,154	iChol	43	1.85e−2	9.73e−1
				\	-	-	6.26e+0
				SSAI	59	8.70e+0	9.84e−1
Economics	finan512	74,752	596,992	iChol	8	5.72e−3	1.20e−1
				\	-	-	1.21e−1
				SSAI	11	7.32e+0	1.33e−1
Electro-	mhd1280	1,280	22,778	iChol	6	6.68e−4	9.63e−3
magnetics				\	-	-	1.51e−3
				SSAI	12	4.13e−1	1.18e−2
	mhd4800b	4,800	27,520	iChol	2	4.07e−4	2.25e−3
				\	-	-	1.72e−3
				SSAI	12	3.67e−1	7.58e−3
	2cubes_sphere	101,492	1,647,264	iChol	9	3.00e−2	3.04e−1
				\	-	-	2.86e+0
				SSAI	10	1.96e+1	2.54e−1
	tmt_sym	726,713	5,080,961	iChol	1043	4.87e−2	1.69e+2
				\	-	-	2.49e+0
				SSAI	1976	1.59e+2	1.79e+2
Geomechanics	Bump_2911	2,911,419	127,729,899	SSAI	784	4.97e+3	6.14e+2
Materials	crystm03	24,696	583,770	iChol	2	5.09e−3	1.90e−2
				\	-	-	2.21e−1
				SSAI	11	8.80e+0	4.14e−2
Optimization	torsion1	40,000	197,608	iChol	-	-	1.81e+0
				\	651	8.69e+1	2.53e+1
				SSAI	-	-	1.24e+1
	gridgena	48,962	512,084	iChol	9353	8.20e+2	2.62e+3
				\	3147	1.02e+3	9.55e+2
				SSAI	18	1.52e−3	1.35e−1
Power	1138_bus	1,138	4,054	iChol	-	-	4.35e−2
Network				\	29	1.92e+0	1.14e−1
				SSAI	713	3.63e−3	6.64e+0
	ecology2	999,999	4,995,991	iChol	-	-	1.02e−1
				\	987	7.19e+0	4.99e+0
				SSAI	141	2.11e−4	3.84e−2
Random	wathen120	36,441	565,761	iChol	-	-	9.24e−4
				\	451	6.32e−2	8.76e−2
				SSAI	1718	3.94e−2	4.67e+2
Statistics	Chem97ZtZ	2,541	7,361	iChol	-	-	1.74e+0

				\	3248	2.60e+2	2.04e+2
				SSAI	10	3.88e−3	8.81e−2
Structural	Kuu	7,102	340,200	iChol	-	-	6.19e−2
Mechanics				\	18	6.52e+0	1.03e−1
				SSAI	1	2.07e−4	6.00e−4
	boneS01	127,224	5,516,602	\	-	-	1.10e−3
				SSAI	15	7.02e−2	6.40e−3
	boneS10	914,898	40,878,708	\	72	4.60e−3	1.89e−1
				SSAI	-	-	1.45e−2
	bone010	986,703	47,851,783	SSAI	118	6.10e+0	2.07e−1
	Flan_1565	1,564,794	114,165,372	iChol	1948	2.28e+0	1.78e+3
				SSAI	1426	2.40e+3	8.12e+2
Thermal	ted_B_unscaled	10,605	144,579	iChol	1	1.56e−3	3.04e−3
				\	-	-	5.37e−3
				SSAI	8	1.99e+0	1.16e−2
	thermomech_TC	102,518	711,558	iChol	8	1.46e−2	2.46e−1
				\	-	-	2.75e−1
				SSAI	12	8.77e+0	2.56e−1
	thermal2	1,228,045	8,580,313	iChol	1817	1.47e−1	7.24e+2
				\	-	-	4.36e+0
				SSAI	2418	4.36e+2	6.37e+2

3.1. Matrices from SuiteSparse. Table 1 gives results for matrices where M did not need to be modified within PCG. It shows there are some large problems where SSAI is the only method that works. There is a way to ensure that iChol succeeds with the '`diagcomp`' option with parameter α, which calculates iChol on `A + α*diag(diag(A))`. (For scaled A, this is equivalent to `A + α*speye(n)`.) However, there is no foolproof way for selecting α. On the matrices tested, $\alpha = 0.1$ did not work but $\alpha = 1$ did. In the latter case, the PCG time T_2 was substantially larger than with SSAI. MATLAB advises that $\alpha =$ `max(sum(abs(A))./diag(A)) - 2` guarantees `A + α*diag(diag(A))` is diagonally dominant—a sufficient condition for iChol to succeed, but a too conservative one. It led to $\alpha \approx 10$ and even more iterations. We recommend iChol be used only when it works with $\alpha = 0$.

Table 1 also shows that the cost per iteration of PCG/SSAI is almost always less than that of PCG/iChol. If the number of PCG iterations is lower or comparable for SSAI, which indeed occurs frequently, it would be preferred even when the other methods work. Note that iChol and '\' are intrinsic MATLAB functions coded in `C` and thus can be expected to outperform `.m` functions such as Algorithms 2 and 3, or PCG with iChol. The disparity between run times of intrinsic and `.m` functions in MATLAB can be 3 orders of magnitude [1, 17]. Thus, although T_1 is often significant for SSAI, if efficiently implemented it could outperform iChol on large systems (given that `nnz`$(M) <$ 2∗`nnz`(L)). Also, the T_1 cost is less significant when there are multiple bs.

Table 2 contains the case where M had to be modified by Algorithm 3 (meaning $\hat{\rho} = r^H M r / \|r\|_2^2$ is small or negative). SSAI is the only method that succeeds on the three largest problems. For smaller problems, it seems that other methods usually work better. The number of iterations reported includes iterations before and after M was changed. The number of changes to M is labeled **R**. Evidently with relatively few corrections to M, SSAI gives an effective HPD preconditioner.

For StocF-1465, the problem that took the most PCG iterations, we conduct a further study in Table 3 to see if larger values of `lfil` and `itmax` give a better M. Monitoring `nnz`(M) shows that the Algorithm 2 is terminating mostly because of `itmax` and not `lfil`. To avoid discrepancies in run-time unrelated to the algorithm, we define a measure **Obj** = [`nnz`(A) + `nnz`(M)]∗`Itns` that roughly represents the number of operations completed by PCG. We see that the number of iterations is monotonically decreasing, and **Obj** has a general downward trend. We conclude that for some applications, tweaking the parameters `lfil` and `itmax` can be advantageous.

3.2. Scale-up for Trefethen challenge matrices. Table 4 shows scale-up results for a challenge matrix [28, 29]. The original matrix has $n = 20,000$ with increasing prime numbers on the main diagonal and 1s wherever $|i - j| = 2^N$ for some non-negative integer N, and the challenge is

TABLE 2
Results with modified M

Field	**Matrix**	n	nnz	**Method**	**Itns**	**R**	T_1	T_2
Fluid	bcsstk13	2,003	83,883	\		-	-	2.08e−2
Dynamics				SSAI	320	1	1.61e+0	1.48e−1
	Pres_Poisson	14,822	715,804	iChol	206	-	1.48e−2	1.19e+0
				\		-	-	8.43e−2
				SSAI	202	2	1.21e+1	8.52e−1
	cfd2	123,440	3,085,406	\		-	-	1.80e+0
				SSAI	1235	1	5.32e+1	2.90e+1
	StocF-1465	1,465,137	21,005,389	SSAI	7983	1	9.29e+2	1.85e+3
Optimization	cvxbqp1	50,000	349,968	\		-	-	1.72e−1
				SSAI	1642	1	4.96e+0	1.34e+1
Structural	sts4098	4,098	72,356	\		-	-	1.02e−2
Mechanics				SSAI	110	1	1.21e+0	8.68e−2
	bcsstk18	11,948	149,090	\		-	-	2.69e−2
				SSAI	441	1	2.40e+0	9.31e−1
	hood	220,542	9,895,422	\		-	-	8.44e−1
				SSAI	567	1	1.88e+2	3.77e+1
	Fault_639	638,802	27,245,944	SSAI	662	1	5.76e+2	1.10e+2
	Queen_4147	4,147,110	316,548,962	SSAI	1508	1	1.21e+4	2.02e+3
Undirected	pdb1HYS	36,417	4,344,765	iChol	276	-	1.26e−1	7.05e+0
Graph				\		-	-	9.77e−1
				SSAI	512	2	7.75e+1	9.80e+0

TABLE 3
Results on StocF-1465 with varying `lfil` *and* `itmax`

lfil	itmax	nnz(M)	nnz(M)+nnz(A)	**Itns**	**R**	**Obj**
20	40	11,033,497	32,038,886	7691	1	1.03
30	60	13,342,987	34,348,376	6767	1	0.97
40	80	15,282,153	36,287,542	6072	1	0.92
50	100	16,981,205	37,986,594	5487	1	0.87
60	120	18,526,847	39,532,236	5263	1	0.87
70	140	19,950,933	40,956,322	5156	1	0.88
80	160	21,293,187	42,298,576	4660	2	0.82
90	180	22,566,173	43,571,562	4631	1	0.84
100	200	23,774,337	447,797,26	4315	1	0.81

to compute $e_1^T A^{-1} e_1$. SuiteSparse contains cases $n = 2,000$ and 20,000. We generated two larger matrices of the same form. They remind us that iterative methods are essential for systems that have substantial fill-in in their Cholesky factors. For iChol and SSAI, T_2 scales linearly with `nnz` (with SSAI being twice as fast as iChol). On these problems, '\' scales poorly because the bandwidth grows linearly with n. It fails on even the moderately sized Trefethen_200000. The PCG iterations remain roughly constant because the matrix becomes more diagonally dominant near the bottom right corner, and the solution vector is also concentrated at the bottom. We checked all methods on the original Trefethen problem by solving with $b = e_1$. Their performance was similar. For each n we recovered the first ten digits of the solution: $0.7250188326, 0.7250783462, 0.7250809785, 0.725081256$ (found by all methods that worked). The $n = 20,000$ value agrees with [28].

3.3. Scale-up for finite-element linear elasticity matrices. Table 5 shows scale-up results for a structural mechanics problem that results from discretizing a cube using quadratic hexahedral finite elements. All elements are cubes, leading to a relatively well-conditioned A. We show results for $4, 8, 16, 32$ elements per direction. As in Table 4, '\' fails, though later because the bandwidth of A grows less rapidly with n compared to the Trefethen problems. Both iterative methods work well. SSAI needs more PCG iterations but less PCG time. The PCG iterations roughly double as the number of elements increases—a small price for 2X the resolution in all three directions.

TABLE 4
Combinatorics problem scale-up

Matrix	n	nnz	**Method**	**Itns**	T_1	T_2
Trefethen_2000	2,000	41,906	iChol	5	5.82e−4	4.95e−3
			\	-	-	2.15e−2
			SSAI	4	4.04e−1	2.53e−3
Trefethen_20000	20,000	554,466	iChol	5	5.73e−3	3.12e−2
			\	-	-	5.76e+0
			SSAI	3	5.27e+0	1.30e−2
Trefethen_200000	200,000	6,875,714	iChol	4	9.38e−2	3.13e−1
			SSAI	3	7.79e+1	1.60e−1
Trefethen_2000000	2,000,000	81,805,698	iChol	3	1.71e+0	3.35e+0
			SSAI	2	2.06e+3	1.49e+0

TABLE 5
Ordered grid structural mechanics scale-up

Matrix	n	nnz	**Method**	**Itns**	T_1	T_2
FEM3D_2	1,029	71,445	iChol	11	4.03e−3	1.08e−2
			\	-	-	6.16e−3
			SSAI	13	1.32e+0	8.64e−3
FEM3D_3	10,125	1,021,365	iChol	19	4.79e−2	1.31e−1
			\	-	-	1.23e+0
			SSAI	25	2.00e+1	1.18e−1
FEM3D_4	89,373	10,525,557	iChol	35	5.00e−1	2.40e+0
			\	-	-	9.65e+1
			SSAI	45	2.29e+2	2.09e+0
FEM3D_5	750,141	94,852,341	iChol	67	4.49e+0	4.18e+1
			SSAI	81	2.42e+3	3.40e+1

3.4. Scale-up for finite-volume petroleum engineering matrices. Table 6 shows scale-up results for SPE10, an important benchmark for testing solver methods in petroleum engineering [8]. Examples were constructed by Klockiewicz [15] using the methods in [18] for a petroleum reservoir simulation. Here again, '\' fails for the larger problems, while both iterative methods work well. As with the preceding finite-element matrices, iChol needs fewer PCG iterations but more PCG time. For this problem, $\texttt{nnz}(M) \approx 1.1\texttt{nnz}(A)$, almost reaching the relative bound for $nnz(M)$.

3.5. Scale-up for finite-element ice-sheet matrices. Table 7 shows scale-up results for a model of ice flow in Antarctica [5, 27]. SSAI is the only method that works on all cases, with '\' failing for the larger ones and iChol failing on all of them. For this problem, $\texttt{nnz}(M) \approx 0.3$–$0.4\texttt{nnz}(A)$, leaving room for a denser M.

4. Parallelism and complexity. Parallelism requires examination of run-time dependence on n, k, and number of processors p. We can assume that communication between processors is negligible for $p \ll n$, but the verification of this assumption and the following idealized analysis is the subject for another paper. For a nonsingular A, $n \le k \le n^2$. All lines in the outer loop of Algorithm 2 are trivially parallelizable on up to n machines, because each column is computed separately. Line 16 could be parallelized with greater effort, but it is only an $O(k)$ operation occurring once. Lines 6–12 are executed $O(k)$ times, requiring $O(k/n)$ arithmetic operations each time. Line 14 runs n times with $O(k/n)$ operations each time. The time to compute M is therefore $T(A) \equiv O(k^2/(np))$.

Algorithm 3 is dominated by matrix-vector products that require $O(k)$ arithmetic operations. In exact arithmetic, the loop runs at most n times and at least once. It must happen sequentially. On parallel machines, matrix-vector products can be done row by row. Therefore, with p machines they take $O(k/p)$ time. The norm on line 13 is an $O(n/p)$ operation because the contribution of each processor must be summed. Therefore, the run-time for PCG on p machines is $\tau(A) \equiv O(kn/p)$.

When $\tau \gtrsim T$ it could be advantageous to use larger `lfil` and `itmax` for computing M. This

TABLE 6
Finite-Volume problem scale-up

Matrix	n	nnz	**Method**	**Itns**	T_1	T_2
lin4E5	422,400	2,912,312	iChol	445	1.86e−2	6.11e+1
			\	-	-	8.02e+0
			SSAI	662	7.48e+1	3.47e+1
lin1E6	1,122,000	7,779,996	iChol	1139	6.88e−2	6.87e+2
			\	-	-	8.46e+1
			SSAI	1745	3.97e+2	2.40e+2
lin2E6	2,097,152	14,581,760	iChol	1795	1.35e−1	2.73e+3
			SSAI	2719	1.38e+3	7.35e+2
lin4E6	4,096,000	28,518,400	iChol	2008	2.42e−1	6.57e+3
			SSAI	2950	5.32e+3	1.55e+3
lin8E6	8,000,000	55,760,000	iChol	2960	5.53e−1	2.09e+4
			SSAI	4191	2.05e+4	5.79e+3
lin1E7	16,003,008	111,640,032	iChol	2950	1.11e+0	4.48e+4
			SSAI	4211	8.56e+4	1.18e+4

TABLE 7
Unordered grid structural mechanics scale-up

Matrix	n	nnz	**Method**	**Itns**	**R**	T_1	T_2
ant16_5	629,544	29,697,024	\	-	-	-	1.16e+1
			SSAI	5033	1	6.92e+2	6.75e+2
ant16_10	1,154,164	57,537,984	\	-	-	-	9.95e+1
			SSAI	8973	2	1.54e+3	2.22e+3
ant8_5	2,521,872	119,642,880	SSAI	6339	3	4.34e+3	3.64e+3
ant8_10	4,623,432	231,808,080	SSAI	12063	2	1.16e+4	1.54e+4

was true on `StocF-1465` (which needed the most PCG iterations among the SuiteSparse matrices).

Such run-time information should be taken into account when we choose a method to solve a system, or try to bound the size of the system being constructed. A comprehensive study such as [16] is needed to verify how these theoretical results for parallel machines hold up in practice.

5. Discussion. The numerical results show that even on serial machines, SSAI is competitive with iChol preconditioners and direct methods for large HPD linear systems whose Cholesky factors are denser than A. Moreover, SSAI + PCG is the most robust of the four methods, and guarantees a solution in a wider range of cases. For small systems, the overhead of computing the preconditioner is considerable, although this is partly due to inefficient implementation compared to built-in functions. The overhead for computing M is less significant when there are several right-hand sides or several similar matrices, such that the preconditioner M can be reused. Although there is no 'one size fits all' preconditioner, our results suggest that SSAI for HPD systems is suitable if some of the following criteria are met: n or k is large, A arises from finite element schemes, factors of A would be too dense, or parallelization is essential.

A future research direction is to develop a SSAI-type algorithm for more general matrices. For example, for square or rectangular systems the normal equation $A^H Ax = A^H b$ may suggest an SSAI-type preconditioner for LSQR and LSMR [22, 11]. Also, if we can form an HPD preconditioner for an indefinite system, it could be used with MINRES [21, 6].

Acknowledgements. We thank Eric Darve, Bazyli Klockiewicz, and Leopold Cambier for consultation and data regarding the SPE10 and Antarctica matrices. We also recognize the invaluable resource that the SuiteSparse collection [9] represents.

REFERENCES

[1] T. Andrews, *Computation time comparison between Matlab and C++ using launch windows*, 2012, https://pdfs.semanticscholar.org/ed2b/5f9a2ca37e55052eafbf5abc166245cf7995.pdf.

[2] O. Axelsson, *Iterative Solution Methods*, Cambridge University Press, Cambridge, 1994, http://dx.doi.org/10.1017/CBO9780511624100.

[3] M. Benzi, *Preconditioning techniques for large linear systems: A survey*, J. Comput. Phys., 182 (2002), pp. 418–447, http://dx.doi.org/10.1006/jcph.2002.7176.

[4] M. Benzi, C. D. Meyer, and M. Tuma, *A sparse approximate inverse preconditioner for the conjugate gradient method*, SIAM J. Sci. Comput., 17 (1996), pp. 1135–1149, http://dx.doi.org/10.1137/S1064827594271421.

[5] L. Cambier, C. Chen, E. G. Boman, S. Rajamanickam, R. S. Tuminaro, and E. Darve, *An algebraic sparsified nested dissection algorithm using low-rank approximations*. arXiv preprint arXiv:1901.02971, 2019.

[6] S. C. Choi, C. C. Paige, and M. A. Saunders, *MINRES-QLP: a Krylov subspace method for indefinite or singular symmetric systems*, SIAM J. Sci. Comput., 33 (2011), pp. 1810–1836, http://dx.doi.org/10.1137/100787921.

[7] E. Chow, *A priori sparsity patterns for parallel sparse approximate inverse preconditioners*, SIAM J. Sci. Comput., 21 (2000), pp. 1804–1822, http://dx.doi.org/10.1137/S106482759833913X.

[8] M. Christie, M. Blunt, and et al., *Tenth SPE comparative solution project: A comparison of upscaling techniques*, Soc. Petrol. Eng. J., 4 (2001), http://dx.doi.org/10.2118/72469-PA.

[9] T. A. Davis and Y. Hu, *The University of Florida sparse matrix collection*, ACM Trans. Math. Softw., 38 (2011), http://dx.doi.org/10.1145/2049662.2049663.

[10] S. Demko, W. F. Moss, and P. W. Smith, *Decay rates for inverses of band matrices*, Math. Comp., 43 (1984), pp. 491–499, http://dx.doi.org/10.2307/2008290.

[11] D. C.-L. Fong and M. Saunders, *LSMR: An iterative algorithm for least-squares problems*, SIAM J. Sci. Comput., 33 (2011), pp. 2950–2971, http://dx.doi.org/10.1137/10079687X.

[12] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins, Baltimore, MD, 1983.

[13] M. J. Grote and T. Huckle, *Parallel preconditioning with sparse approximate inverses*, SIAM J. Sci. Comput., 18 (1997), pp. 838–853, http://dx.doi.org/10.1137/S1064827594276552.

[14] M. R. Hestenes and E. L. Stiefel, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bureau Standards, 49 (1952), pp. 409–436, http://dx.doi.org/10.6028/jres.049.044.

[15] B. Klockiewicz and E. Darve, *Sparse hierarchical preconditioners using piecewise smooth approximations of eigenvectors*. arXiv preprint arXiv:1907.03406v1, 2019.

[16] L. Y. Kolotilina and A. Y. Yeremin, *Factorized sparse approximate inverse preconditioning II: Solution of 3D FE systems on massively parallel computers*, Int. J. High Speed Comput., 7 (1995), pp. 191–215, http://dx.doi.org/10.1142/S0129053395000117.

[17] J. Kouatchou, *Basic Comparison of Python, Julia, R, MATLAB and IDL*, 2016, https://modelingguru.nasa.gov/docs/DOC-2625.

[18] A. M. Manea, J. Sewall, H. A. Tchelepi, and et al., *Parallel multiscale linear solver for highly detailed reservoir models*, Soc. Petrol. Eng. J., 21 (2016), pp. 2–62, http://dx.doi.org/10.2118/173259-PA.

[19] T. A. Manteufel, *An incomplete factorization technique for positive definite linear systems*, Math. Comp., 34 (1980), pp. 473–497, http://dx.doi.org/10.1090/s0025-5718-1980-0559197-0.

[20] J. A. Meijerink and H. A. van der Vorst, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Math. Comp., 31 (1977), pp. 148–162, http://dx.doi.org/10.2307/2005786.

[21] C. C. Paige and M. A. Saunders, *Solution of sparse indefinite systems of linear equations*, SIAM J. Numer. Anal, 12 (1975), pp. 617–629, http://dx.doi.org/10.1137/0712047.

[22] C. C. Paige and M. A. Saunders, *LSQR: An algorithm for sparse linear equations and sparse least squares*, ACM Trans. Math. Softw., 8 (1982), pp. 43–71, http://dx.doi.org/10.1145/355984.355989.

[23] S. Regev, 2019, https://drive.google.com/drive/u/2/folders/1N6nhNpe6fotNO4D38B0VPuusOvE2w1Yi.

[24] S. Regev, 2019, https://github.com/shakedregev/SRP_with_michael.git.

[25] Y. Saad and M. H. Schultz, *GMRES: A generalized minimal residual algorithm for nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869, http://dx.doi.org/10.1137/0907058.

[26] D. K. Salkuyeh and F. Toutounian, *A new approach to compute sparse approximate inverse of an SPD matrix*, IUST - Int. J. Eng. Sci., 15 (2004), http://dx.doi.org/10.1016/j.amc.2005.06.011.

[27] I. K. Tezaur, M. Perego, A. G. Salinger, R. S. Tuminaro, and S. F. Price, *Albany/felix: a parallel, scalable and robust, finite element, first-order stokes approximation ice sheet solver built for advanced analysis*, Geosci Model Dev, 8 (2015), pp. 1197–1220, http://dx.doi.org/10.5194/gmd-8-1197-2015.

[28] L. N. Trefethen, *A hundred-dollar, hundred-digit challenge*, SIAM News, 35 (2002), p. 65.

[29] L. N. Trefethen, *The SIAM 100-Dollar, 100-Digit Challenge*, 2002, https://people.maths.ox.ac.uk/trefethen/hundred.html.

[30] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*, SIAM, Philadelphia, PA, 1983.