Parallel Hybrid Sparse Linear System Solvers

Murat Manguoğlu, Eric Polizzi, and Ahmed H. Sameh

1 Introduction

Some science and engineering applications give rise to large banded linear systems in which the bandwidth is a very small percentage of the system size. Often, these systems arise in the inner-most computational loop of these applications which indicates that these systems need to be solved efficiently as fast as possible on parallel computing platforms. This motivated the development of the earliest version of the SPIKE tridiagonal linear systems in the late 1970s, e.g. see [27] followed by an investigation of the communication complexity of this solver in [12] in 1984. In both of these studies this solver was not named "SPIKE" until it was further developed in [23, 24] in 2006. In this chapter, we also present an extension of this algorithm for solving sparse linear systems. This is done through reordering schemes that bring as many of the heaviest off-diagonal elements as closer to the main diagonal, followed by extracting effective preconditioners (that encapsulate as many of these heaviest elements as possible) for outer Krylov subspace methods for

M. Manguoğlu (✉)
Department of Computer Engineering, Middle East Technical University, Ankara, Turkey
e-mail: manguoglu@ceng.metu.edu.tr

E. Polizzi (✉)
Department of Electrical and Computer Engineering, University of Massachusetts at Amherst, Amherst, MA, USA

Department of Mathematics and Statistics, University of Massachusetts at Amherst, Amherst, MA, USA
e-mail: polizzi@ecs.umass.edu

A. H. Sameh
Department of Computer Science, Purdue University, West Lafayette, IN, USA
e-mail: sameh@cs.purdue.edu

A. Grama, A. H. Sameh (eds.), *Parallel Algorithms in Computational Science and Engineering*, Modeling and Simulation in Science, Engineering and Technology,
https://doi.org/10.1007/978-3-030-43736-7_4

solving these sparse systems. In each outer iteration variants of the SPIKE algorithm are used for solving linear systems involving these preconditioners. An extensive survey of the SPIKE algorithm and its extensions are given in [10].

2 The SPIKE for Banded Linear Systems (Dense Within the Band)

Consider the nonsingular banded linear system

$$Ax = f \tag{1}$$

shown in Fig. 1 with $A \in \mathbb{R}^{N \times N}$ being of bandwidth $\beta = 2m + 1$. Let N be an integer multiple of p (the number of partitions). In Fig. 1, p is chosen as 4. The off-diagonal blocks are given by

$$\bar{B}_j = \begin{pmatrix} 0 & 0 \\ B_j & 0 \end{pmatrix} \text{ and } \bar{C}_j = \begin{pmatrix} 0 & C_j \\ 0 & 0 \end{pmatrix} \tag{2}$$

for $j = 1, 2, \ldots, p-1$, where $B_j, C_j \in \mathbb{R}^{m \times m}$.

In what follows, we first describe “Spike” as a direct banded solver when A is diagonally dominant followed by the general case for which “Spike” becomes a hybrid (direct-iterative) banded solver for the linear system (1).

$$\underbrace{\begin{pmatrix} A_1 & \bar{B}_1 & & \\ \bar{C}_2 & A_2 & \bar{B}_2 & \\ & \bar{C}_3 & A_3 & \bar{B}_3 \\ & & \bar{C}_4 & A_4 \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{pmatrix}}_{f}$$

First, let A be a diagonally dominant matrix. Thus, each A_j is also diagonally dominant, $j = 1, 2, \ldots, p$, with the block diagonal matrix

$$\underbrace{\begin{pmatrix} A_1 & \bar{B}_1 & & \\ \bar{C}_2 & A_2 & \bar{B}_2 & \\ & \bar{C}_3 & A_3 & \bar{B}_3 \\ & & \bar{C}_4 & A_4 \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{pmatrix}}_{f}$$

Fig. 1 $A \in \mathbb{R}^{N \times N}$; bandwidth: $\beta = 2m + 1$; number of partitions: $p = 4$, $A_j \in \mathbb{R}^{n \times n}$, $j = 1, 2, \ldots, p$; $\bar{B}_j, \bar{C}_{j+1} \in \mathbb{R}^{n \times n}$, $j = 1, 2, \ldots, p-1$; $N = 4n$; $n = 3m$

Fig. 2 $p = 4, n = 3m,$ $N = 4n$

$$\underbrace{\begin{pmatrix} I & \bar{V}_1 & & \\ \bar{W}_2 & I & \bar{V}_2 & \\ & \bar{W}_3 & I & \bar{V}_3 \\ & & \bar{W}_4 & I \end{pmatrix}}_{S} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} g_1 \\ g_2 \\ g_3 \\ g_4 \end{pmatrix}}_{g}$$

$$D = \begin{pmatrix} A_1 & & & \\ & A_2 & & \\ & & \ddots & \\ & & & A_p \end{pmatrix} \tag{3}$$

nonsingular. Premultiplying both sides of (1) by D^{-1}, we obtain the modified system

$$Sx = g, \tag{4}$$

where the matrix $S = D^{-1}A$, and the updated right-hand side is given in which the bandwidth is very small percentage of the system size. Often, these updated right-hand side are given in Fig. 2. Here, the off-diagonal blocks, $\bar{V}_j$ and $\bar{W}_j$, are given by

$$\bar{V}_j = \left(V_j, 0\right) \text{ and } \bar{W}_{j+1} = \left(0, W_{j+1}\right), j = 1, 2, \ldots, p-1. \tag{5}$$

$$\underbrace{\begin{pmatrix} I & \bar{V}_1 & & \\ \bar{W}_2 & I & \bar{V}_2 & \\ & \bar{W}_3 & I & \bar{V}_3 \\ & & \bar{W}_4 & I \end{pmatrix}}_{S} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} g_1 \\ g_2 \\ g_3 \\ g_4 \end{pmatrix}}_{g}$$

The spikes,

$$V_j = \begin{pmatrix} V_3^{(j)} \\ V_2^{(j)} \\ V_1^{(j)} \end{pmatrix} \text{ and } W_j = \begin{pmatrix} W_1^{(j)} \\ W_2^{(j)} \\ W_3^{(j)} \end{pmatrix} \tag{6}$$

are obtained by solving the linear systems

$$A_j^{-1}[\hat{C}_j, A_j, \hat{B}_j] = [W_j, I_n, V_j] \tag{7}$$

and

$$g_j = A_j^{-1} f_j, j = 1, 2, \ldots, p \tag{8}$$

in which

$$\hat{B}_j = \begin{pmatrix} 0 \\ I_m \end{pmatrix} B_j; \hat{C}_j = \begin{pmatrix} I_m \\ 0 \end{pmatrix} C_j \tag{9}$$

with $V_j, W_j \in \mathbb{R}^{n \times m}$. Solving the linear system in (4), involving the "Spike matrix" S, reduces to solving a much smaller block-tridiagonal system of order $2m(p-1)$ of the form (See Fig. 2),

$$\begin{pmatrix} I_m & V_1^{(1)} & 0 & 0 & & \\ W_1^{(2)} & I_m & 0 & V_3^{(2)} & & \\ W_3^{(2)} & 0 & I_m & V_1^{(2)} & 0 & 0 \\ 0 & 0 & W_1^{(3)} & I_m & 0 & V_3^{(3)} \\ & & W_3^{(3)} & 0 & I_m & V_1^{(3)} \\ & & 0 & 0 & W_1^{(4)} & I_m \end{pmatrix} \begin{pmatrix} x_1^{(b)} \\ x_2^{(t)} \\ x_2^{(b)} \\ x_3^{(t)} \\ x_3^{(b)} \\ x_4^{(t)} \end{pmatrix} = \begin{pmatrix} g_1^{(b)} \\ g_2^{(t)} \\ g_2^{(b)} \\ g_3^{(t)} \\ g_3^{(b)} \\ g_4^{(t)} \end{pmatrix} \tag{10}$$

in which $x_i^{(t)} = (I_m, 0)x_i$, and $x_i^{(b)} = (0, I_m)x_i$, and similarly for $g_i^{(t)}$ and $g_i^{(b)}$. We refer to (10) as the reduced system,

$$Ry = h, \tag{11}$$

where R results from the symmetric permutation

$$PSP^T = \begin{pmatrix} I_\nu & G \\ 0 & R \end{pmatrix} \tag{12}$$

in which $\nu = pn - 2m(p-1)$. Note that since A is nonsingular, so is S as well as R. Solving the reduced system (10) for $x_i^{(b)}$ and $x_{i+1}^{(t)}$, $i = 1, 2, \ldots, p-1$, the solution x of system (4) is retrieved directly via

$$\begin{aligned} x_1 &= g_1 - V_1 x_2^{(t)} \\ x_i &= g_i - W_i x_{i-1}^{(b)} - V_i x_{i+1}^{(t)}, i = 2, 3, \ldots, p-1 \\ x_p &= g_p - W_p x_{p-1}^{(b)}. \end{aligned} \tag{13}$$

In summary, the SPIKE algorithm for solving the diagonally dominant banded system $Ax = f$ consist of the D-S factorization scheme in which D is block diagonal and S is the corresponding spike matrix. Consequently, solving system (1) consists of two phases:

(i) solve $Dg = f$ followed by
(ii) solve $Sx = g$ via the reduced system approach.

In (i) each system $A_j g_j = f_j$ is solved via the classical LU-factorization as implemented in Lapack [2].

Observe that if we assign one processor (or one multicore node) to each partition, then solving $Dg = f$ realizes maximum parallelism with no interprocessor communications. Solving the reduced system (10), however, requires interprocessor communications which increases as the number of partitions p increases. The retrieval process (13) again achieves almost perfect parallelism.

Variations of this basic form of the SPIKE algorithm are given in [24]. Also note that this SPIKE algorithm requires a larger number of arithmetic operations than those required by the classical banded LU-factorization scheme. In spite of this higher arithmetic operation count, this direct form of the SPIKE algorithm realizes higher parallel performance than ScaLapack on an 8-core Intel processor, see Fig. 3 [17], due to enhanced data locality.

The reason for the superior performance of Spike is illustrated in Fig. 4 showing that the total number of off-chip data accessed (in bytes) for Spike (red color) is less than that required by ScaLapack. Also, Fig. 5 shows that the number of instructions executed by Spike is almost half that required by ScaLapack. For more details about the measurements shown in Figs. 4 and 5, see Liu et al. [13].

Second, if the banded linear system (1) is not diagonally dominant, there is no guarantee that any of the diagonal blocks A_j, $j = 1, 2, \ldots, p$, see Fig. 1, is nonsingular. In this case, the banded linear system (1) is solved via a preconditioned Krylov subspace method such as GMRES or BiCGStab, e.g. see Saad [25]. Here the preconditioner M is chosen as $M = \hat{D}\hat{S}$ where $\hat{D} = diag(\hat{A}_1, \hat{A}_2, \ldots, \hat{A}_p)$ with

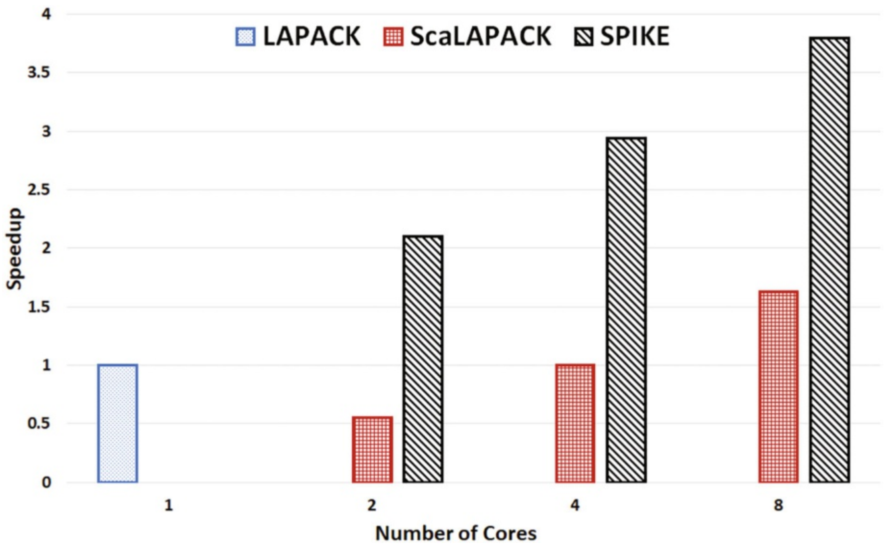

Fig. 3 Speedup of SPIKE and ScaLapack compared to the sequential Lapack for solving a linear system of size 960,000 with a bandwidth of 201

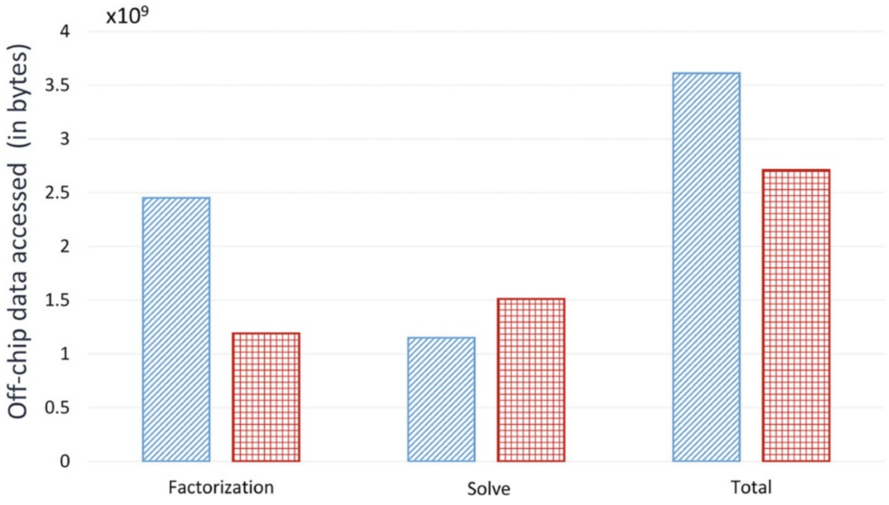

Fig. 4 Off-chip data being accessed for Spike (red) and ScaLapack (blue), using 4 cores

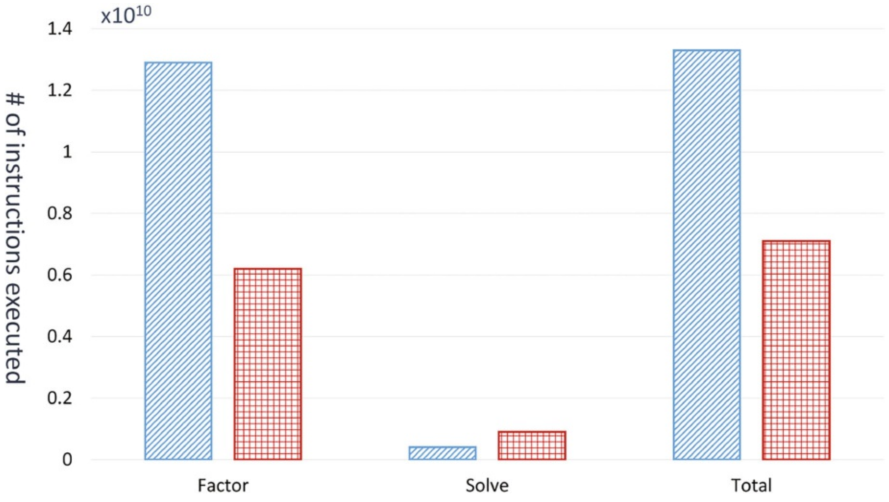

Fig. 5 Number of instructions executed by Spike (red) and ScaLapack (blue), using 4 cores

$\hat{A}_j = \hat{L}_j \hat{U}_j$, in which the factors $\hat{L}_j$ and $\hat{U}_j$ are obtained via the LU=factorization of each A_j using diagonal pivoting with a "boosting" strategy. In other words, if a diagonal element α during the factorization satisfies $|\alpha| \leq \varepsilon||A||_1$ where ε is a multiple of the unit roundoff, then α is modified as follows:

$$\begin{aligned} \alpha &:= \alpha + \theta||A_j||_1 \text{ if } \alpha \geq 0 \\ \alpha &:= \alpha - \theta||A_j||_1 \text{ if } \alpha < 0, \end{aligned} \tag{14}$$

where $\theta \sim \sqrt{\varepsilon}$. $\hat{S}$ is then of a form identical to that of S , see Fig. 2, except that the spikes V_j and W_j are obtained as follows:

$$(\hat{L}_j \hat{U}_j)^{-1}[\hat{C}_j, \hat{B}_j] = [V_j, W_j], \; j = 1, 2, \ldots, p \tag{15}$$

which entails a block forward sweep followed by a block backsweep. In each iteration of GMRES, for example, one needs to solve a system of the form $Mv = r$. This is accomplished in two steps: (1) solve $\hat{D}u = r$, and (2) solve $\hat{S}v = u$. As outlined above, the first system is solved via two triangular solvers: $\hat{L}_j \dot{u}_j = r_j$, and $\hat{U}_j u_j = \dot{u}_j$, $j = 1, 2, \ldots, p$. The second system, $\hat{S}v = u$, is solved via the reduced system approach, see (10), with retrieving the rest of the solution vector v via (13).

Since the elements of the inverse of a banded matrix decay as they move away from the main diagonal, the elements of the spikes V, W decay as they move away from the main diagonal as well. Such decay becomes more pronounced as the degree of diagonal dominance increases. We define the degree of diagonal dominance of A by

$$\tau = \min_{1 \le k \le N} [|a_{kk}| / \sum_{k \ne j} |a_{kj}|]. \tag{16}$$

Even for system (1) for which $\tau \ge 0.25$, one can take advantage of the decay in the spikes V_j, $(||V_q^{(j)}|| \ll ||V_1^{(j)}||)$, and W_j, $(||W_q^{(j)}|| \ll ||W_1^{(j)}||)$ (In our example above $q = 3$). Taking advantage from such a property by replacing $V_q^{(j)}$ and $W_q^{(j)}$ by zero, the reduced system (10) becomes a block diagonal system that requires only obtaining $V_1^{(j)}$ and $W_1^{(j)}$, $j = 1, 2, \ldots, p$. In other words, we need only to obtain the bottom $(m \times m)$ tip of each right spike V_j, and the top $(m \times m)$ tip of each left spike W_j, $1 \le j \le p$. Consequently, if we assign enough processors to obtain the LU-factorization of slightly perturbed $A_1, A_2, \ldots, A_{p-1}$ using the diagonal boosting strategy, and the UL-factorization of similarly perturbed $A_2, A_3, \ldots, A_p$, we need not obtain the whole spikes V_j and W_j. The LU-factorizations will obtain the bottom tips of V_j, while the UL-factorizations will enable obtaining the top tips of W_j resulting in significant savings for computing the coefficient matrix R of the reduced system. Further, since R in this case is block diagonal, solving the reduced system achieves maximum parallelism. This "truncated" version of the SPIKE algorithm was compared with Lapack and MKL-ScaLapack (i.e., Intel's Math Kernel Library) on an Intel multicore processor for solving 8 banded linear systems with coefficient matrices obtained from Matrix-Market (see Table 1). Table 2 shows the ratios:

$$\frac{\text{Average time(MKL-2 cores*)}}{\text{Average time (MKL-1 core)}}, \tag{17}$$

and

Table 1 A Matrix-Market collection of banded systems ($n > 10{,}000$) where kl,ku, N, ~Cond are the lower, upper bandwidths, matrix size, and the condition number estimate, respectively

Matrix name	kl	ku	N	~Cond
s3dkq4m2	614	614	90,449	N/A
s3dkt3m2	614	614	90,449	N/A
fidap035	244	247	19,716	4.3×10^{12}
e40r0000	451	451	17,281	2.2×10^{8}
e40r5000	451	451	17,281	2.2×10^{10}
bcsstk25	292	292	15,439	1.3×10^{13}
bcsstk18	1243	1243	11,948	6.5×10^{11}
bcsstk17	521	521	10,974	2.0×10^{10}

Table 2 Time ratios for Spike and MKL-ScaLapack

	MKL 1-core	MKL 2-cores	Spike 2-cores
Avg. time (ratio)	1.0	8.5	0.4
Rel. res. (norm)	$O(10^{-1})$–$O(10^{-10})$	$O(10^{-2})$–$O(10^{-10})$	$O(10^{-5})$–$O(10^{-11})$

$$\frac{\text{Average time(Spike-2 cores}^*)}{\text{Average time (MKL-1 core)}}, \tag{18}$$

together with the lowest and highest relative residual for each solver for the 8 benchmarks. *Note that for the 2-core entries each core belongs to a different node.

2.1 Multithreaded SPIKE

In shared memory systems, the parallelism in LAPACK LU algorithms can directly benefit from the threaded implementation of the low-level BLAS routines. In order to achieve further scalability improvement, however, it is necessary to move to a higher level of parallelism based on divide-and-conquer techniques. As a result, the OpenMP implementation of SPIKE on multithreaded systems [19, 31], is inherently better suited for parallelism than the traditional LAPACK banded LU solver. A recent stand-alone SPIKE-OpenMP solver (v1.0) [1] has been developed and released to the community.

Among the large number of variants available for SPIKE, the OpenMP solver was implemented using the recursive SPIKE algorithm [23, 24]. The latter consists of solving the reduced system (10) using SPIKE again but where the number of partitions had been divided by two. This process is repeated recursively until only two partitions are left (making the problem straightforward to solve). The SPIKE algorithm applied to two partitions is actually the kernel of recursive SPIKE, and from Fig. 3, we note the efficiency of 2×2 SPIKE which reaches a speedup of two using two partitions with two processors. The recursive SPIKE technique demonstrates parallel efficiency and is applicable to both diagonally and non-diagonally dominant systems. However, it was originally known for its lack of flexibility on distributed architectures since its application was essentially limited to a power of two number of processors. The scheme was then prone to potential waste of parallel resources when applied to shared memory systems using OpenMP [19]; for instance, if 63 cores were available, then only 32 would be effectively used by recursive SPIKE (i.e., the lowest nearest power of two). This limitation was overcome in [31] with the introduction of a new flexible threading scheme that can consider any number of threads. If the number of threads is not a power of two, some partitions are given two threads which, in turn, would benefit from the 2×2 SPIKE kernel. Load balancing is achieved by changing the size of each partition so that the computational costs of the large matrix operations on each partition are matched. This multithreaded SPIKE approach is then ideally suited for shared memory systems since optimized ratios between partition sizes can be tuned for a given system matrix and architecture, independently from user input [1]. Figure 6 demonstrates the efficiency of the scheme. The results show that the speedup performance of the new threaded recursive SPIKE is not limited to a power of two number of threads since the scalability keeps increasing with the number of threads. For example, at 30 threads the overall speed improvement increases from roughly $\times 6$ to roughly $\times 9$, as a result of the increased overall utilization of resources. The results also show that the SPIKE computation time is significantly superior to LAPACK Intel-MKL. We note that the two solvers' scaling performance are similar until 10 threads are reached, at which point SPIKE begins pulling away. Unlike SPIKE, parallelism performance of the inherently recursive serial LU approach used by MKL mainly relies on parallelism available via the BLAS which is rather poor for this matrix.

The SPIKE-openMP solver has been designed as an easy to use, "black-box" replacement to the standard LAPACK banded solver. In order to achieve near feature-parity with the standard LAPACK banded matrix solver, we add to SPIKE the feature known as transpose option, i.e. solve $A^T x = f$. Transpose solve

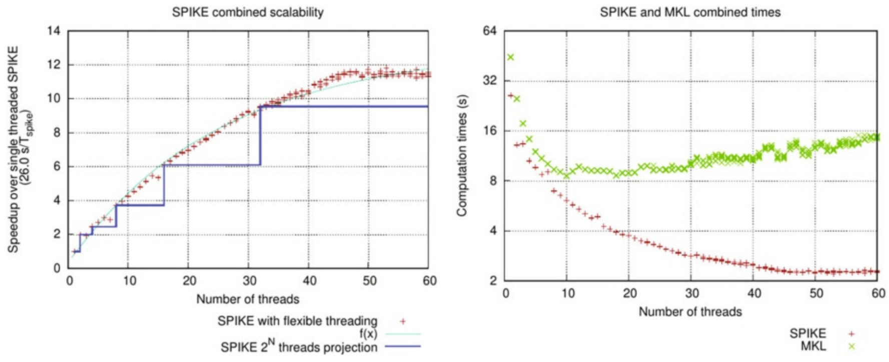

Fig. 6 SPIKE scalability and computation time compared to MKL-LAPACK for a system matrix of size N=1M, bandwidth 321, and with 160 right-hand sides

operation allows improved algorithmic flexibility and efficiency by eliminating the need for an explicit factorization of the matrix transpose when solving:

$$A^T x = f. \tag{19}$$

As a result, if the factorization $A = DS$ is already available, it can now be used to address the new SPIKE solve stages, which are now swapped:

1. solve $S^T y = f$ via the transposed reduced system approach, followed by
2. solve $D^T x = y$.

A transpose version of the recursive reduced system solver which has been proposed in [31] achieves near performance parity with the non-transpose solver.

3 Hybrid Methods for General Sparse Linear Systems

In large-scale computational science and engineering application one is often faced with solving large general sparse linear systems that cannot be reordered into a narrow banded form. Therefore, we use nonsymmetric and symmetric reorderings to maximize the magnitude of the product of the diagonal elements, move as many of the largest off-diagonal elements as possible close to the main diagonal, and extract a generalized banded preconditioner. In the next subsection we describe such a reordering process after which an effective preconditioner can be extracted where linear systems involving the such a preconditioner is solved using a variant of the SPIKE algorithm.

3.1 Weighted Nonsymmetric and Symmetric Reorderings for Sparse Matrices

As the first step of the reordering scheme we apply a nonsymmetric permutation and scaling (if needed) to make the diagonal of the coefficient matrix as large as possible. Such nonsymmetric permutation and scaling techniques are already available in the Harwell Subroutine Library (HSL) and is called MC64 [8] which (without scaling) creates permutations Π_1 and Π_2 such that the magnitude of the product of the diagonal elements of $B = \Pi_1 A \Pi_2$ is maximize, where A is the original coefficient matrix. This is followed by obtaining a symmetric permutation of B, $C = PBP^T$, where P is determined by the Fiedler vector [9] derived from B. The Fiedler vector is the eigenvector corresponding to the second smallest eigenvalue of the "weighted Laplacian" matrix based on B. This eigenvalue is sometimes called the algebraic connectivity of the graph. Note that the smallest eigenvalue is zero. As a result, many of the heaviest off-diagonal elements of C are much closer to the main diagonal.

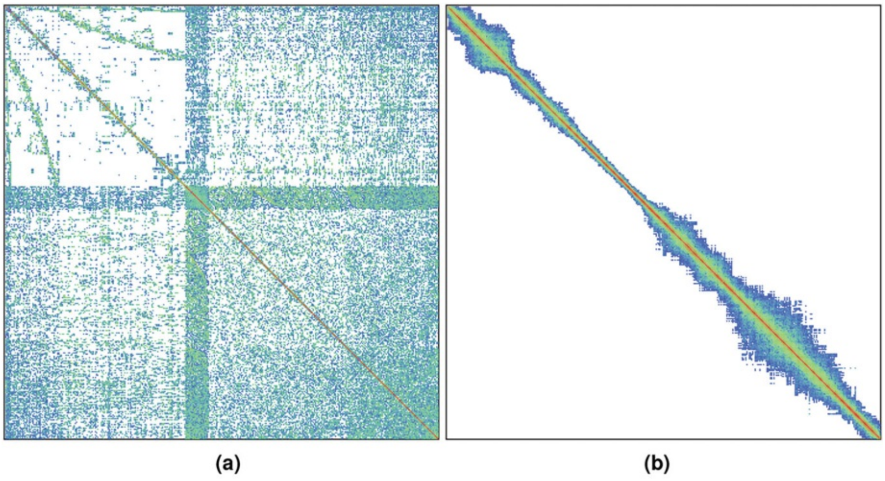

Fig. 7 The effect of the weighted spectral reordering using the Fiedler vector on *F2* matrix (colors indicate the magnitude of the absolute value of the elements. Red, green, and blue are the largest, intermediate, and smallest elements, respectively. (**a**) Original matrix. (**b**) Reordered matrix

Here, PSPIKE refers to a solver that is a hybrid of the sparse direct solver Pardiso [29] and the SPIKE algorithm. In Fig. 7, we illustrate the effect of such reordering on a symmetric stiffness matrix with 71,505 rows and columns, obtained from the SuiteSparse Matrix Collection [6].

From the original sparse linear system $Ax = f$, we obtain $By = g$, where $y = \Pi_2^T x$, and $g = \Pi_1 f$. If B is symmetric, one can form the "weighted Laplacian" matrix, $L_{ij}^w = -|b_{ij}|$, and

$$L_{jj}^w = \sum_k |b_{kj}| \tag{20}$$

as follows: Note that one can obtain the unweighted Laplacian by simply replacing each nonzero element of the matrix B by 1. In this subsection, we consider the weighted case as a preprocessing tool for the PSPIKE algorithm given in Sect. 3.3.

We assume that the corresponding graph is connected since the disconnected components can be easily identified and the Fiedler vector can be computed independently for each if the graph is disconnected. The eigenvalues of L^w are $0 = \lambda_1 < \lambda_2 \leq \lambda_3 \leq \ldots \leq \lambda_n$. The Fiedler vector, x_F, is the eigenvector corresponding to smallest nontrivial eigenvalue, λ_2. Since we assume a connected graph, the trivial eigenvector x_1 is a vector of all ones. If the coefficient matrix, B, is nonsymmetric, we simply construct L^w using the elements of $(|B| + |B^T|)/2$, instead of those of $|B|$.

A Trace Minimization [26, 28] based parallel algorithm for computing the Fiedler vector, TRACEMIN-Fiedler, has been proposed in [16]. We consider the standard symmetric eigenvalue problem,

$$L^w x = \lambda x. \tag{21}$$

The trace minimization eigensolver is based on the observation,

$$\min_{X \in \mathcal{X}_p} tr(X^T L^w X) = \sum_{i=1}^{p} \lambda_i, \tag{22}$$

where $\mathcal{X}_p$ is the set of all $n \times p$ matrices, X for which $X^T X = I$. The equality holds if and only if the columns of the matrix X span the eigenspace corresponding to the smallest p eigenvalues. At each iteration of the trace minimization algorithm an approximation $X_k \in \mathcal{X}_p$ which satisfies $X_k^T L^w X_k = \Theta_k$ for some diagonal Θ_k is obtained. The approximation X_k is corrected with Δ_k obtained by

$$\begin{array}{l} \text{minimizing } tr[(X_k - \Delta_k)^T L^w (X_k - \Delta_k)] \\ \text{subject to } X_k^T \Delta_k = 0. \end{array} . \tag{23}$$

The solution of the (23) can be obtained by solving the following saddle point problem:

$$\begin{pmatrix} L^w & X_k \\ X_k^T & 0 \end{pmatrix} \begin{pmatrix} \Delta_k \\ L_k \end{pmatrix} = \begin{pmatrix} L^w X_k \\ 0 \end{pmatrix}. \tag{24}$$

Once Δ_k is known, X_{k+1} is obtained by computing $(X_k - \Delta_k)$ which forms the section $X_{k+1}^T L^w X_{k+1} = \Theta_{k+1}$, $X_{k+1}^T X_{k+1} = I$. In [16], we solve those saddle point systems by computing the block LU-factorization of the coefficient matrix in (24), i.e. by forming the Schur complement matrix explicitly since we are only interested in the second smallest eigenvector and hence p is small. Then, the main computational cost is solving sparse linear systems of equations with a few right-hand side vectors where the coefficient matrix, L^w, is a large sparse and symmetric positive semi-definite matrix. The details of the TRACEMIN-Fiedler algorithm are given in [16]. This algorithm proved (see Fig. 8) to be more suitable for implementation on parallel architectures compared to the eigensolver used in HSL for (21). Table 3 shows the dimension, number of nonzeros, and symmetry properties of four large matrices obtained from the SuiteSparse Matrix Collection [7].

Table 3 Properties of matrices from the SuiteSparse matrix collection

Matrix group/name	n	nnz	Symmetry
Rajat/rajat31	4,690,002	20,316,253	No
Schenk/nlpkkt120	3,542,400	95,117,792	Yes
Freescale/freescale1	3,428,755	17,052,626	No
Zaoui/kkt_power	2,063,494	12,771,361	Yes

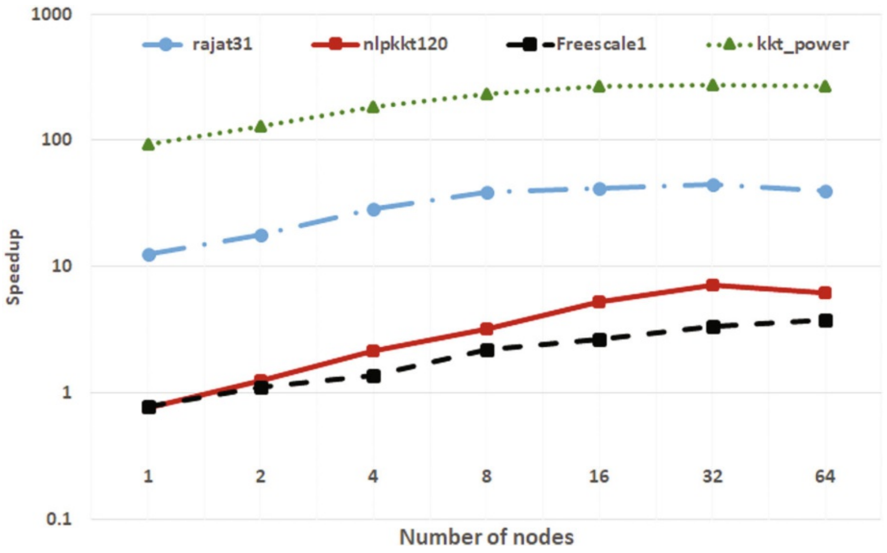

Fig. 8 Seedup of TRACEMIN-Fiedler reordering (using 8 cores per node) compared to the sequential HSL_MC73

After these two reordering steps, the resulting sparse linear system is of the form $Cz = h$, where $C = PBP^T$, $z = Py$, and $h = Pg$, with C having its heaviest off-diagonal elements as close to the main diagonal as possible. Choosing a central "band" of bandwidth $(2\beta + 1)$ as a preconditioner M of a Krylov subspace method with β chosen such that

$$||M||_F \simeq (1 - \epsilon)||C||_F. \tag{25}$$

Here $|| \cdot ||_F$ denotes the Frobenius norm, and ϵ chosen in the interval [0.001, 0.05]. Assuming C is of sufficiently large order n, say $n = 10^6$, then if $\beta \leq 10$, we call M a Narrow banded Preconditioner (NBP). If $\beta > 10$, we choose M as a block-tridiagonal preconditioner in which the diagonal blocks are sparse with relatively large "bandwidth," and the interconnecting off-diagonal blocks are dense square matrices of small dimensions. We call such M as Medium banded Preconditioner (MBP). When β becomes much larger than 10 in order to encapsulate as many off-diagonal as possible, we construct the preconditioner M as overlapped block diagonal sparse matrices. In this case, M is referred to as Wide banded Preconditoner (WBP). In each outer Krylov subspace iteration, one needs to solve linear systems involving M. For the cases of "MBP" and "WBP," one needs to use a sparse linear system solver. In Fig. 9 we show the classical computational loop that arises in many science and engineering applications. Solving linear systems occurs in the inner-most loop where the solution of such systems is needed to yield only modest relative residuals. For this purpose, we created a family of solvers that generalizes SPIKE for solving sparse linear systems $Ax = f$ using hybrid

Fig. 9 Target computational loop

```
Loop: Integration
    Loop: Nonlinear iteration
        Loop: Linear system solvers
            Implemented on parallel computing platforms;
        End $\eta_k$
    End $\epsilon_l$
End $\Delta t$
```

schemes, i.e. a combination of the direct sparse linear system solver Pardiso [29] and SPIKE. Even though SPIKE, rather than Pardiso is used for the case M being narrow banded, we refer to our family of hybrid solvers as PSpike_NBP, PSpike_MBP, and PSpike_WBP, respectively. In Fig. 10, we illustrate the structure of each of those preconditioners obtained from the reordered matrix C. Next, we describe and present some results illustrating the performance of each of Narrow Banded, Medium Banded, and Wide Banded preconditioners.

3.2 PSPIKE_NBP

Certain sparse linear systems $Ax = f$ yield, after the reordering procedures described in Sect. 3.1, effective narrow banded preconditioners to Krylov subspace methods like GMRES or BiCGStab.

Example 1
The first system $A_1x_1 = f_1$ considered here has the sparse coefficient matrix A_1 := "Rajat31" from the SuiteSparse Matrix Collection[6] of order ≃4.7M, see Fig. 11, which is in the form of an arrowhead. After reordering, and choosing $\epsilon = 0.05$, we extract a banded preconditioner M of bandwidth $2\beta + 1 = 11$, i.e. $\beta = 5$. Using an outer Krylov solver (BiCGStab) with a stopping criterion of relative residual $= 10^{-5}$, Fig. 12 shows that PSPIKE_NBP consumes ∼2.8 s on an Intel cluster of 32 nodes (8 cores/node). Here, solving linear systems of the form $Mz = r$ in each BiCGStab iteration is achieved by using the truncated version of the SPIKE algorithm outlined in Sect. 2. We compare the performance of PSPIKE_NBP with IBM's direct sparse linear system solver WSMP (implemented on the same Intel cluster). Figure 12 shows that while the factorization stage of WSMP is quite scalable, solving $A_1x_1 = f_1$ using WSMP on 16 nodes of this Intel cluster consumes ∼27 s (approximately 9.6 times slower than PSPIKE_NBP). This is due to solving the sparse triangular systems resulting from the LU-factorization of A_1. Note, however, that solving $A_1x_1 = f_1$ via WSMP yields a relative residual of order 10^{-10}.

Example 2
Here, we consider the sparse linear system $A_2x_2 = f_2$, where A_2 results from a Microelectromechanical System (MEMS) simulation—a mix of structural and

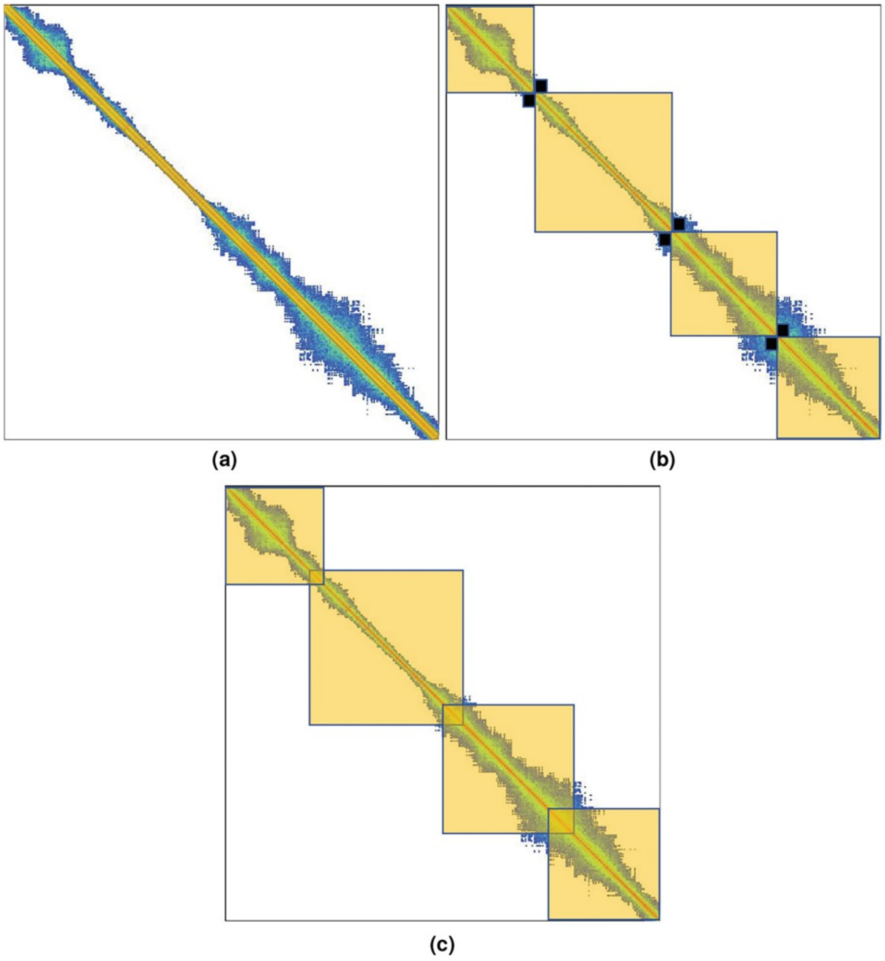

Fig. 10 Three forms of preconditioners based on the band structure and bandwidth, illustrated on $F2$ matrix after reordering. Yellow and Black colors indicate the preconditioners. (**a**) Narrow banded preconditioner. (**b**) Medium banded preconditioner. (**c**) Wide banded preconditioner

electromagnetic—with A_2 banded (sparse within the band) of order 11.0M and bandwidth of 0.3M, see Fig. 13. On an Intel cluster of 64 nodes (8 cores/node) PSPIKE_NBP with a preconditioner of bandwidth 11 consumes $\sim$2.4 s to obtain an approximation of x_2 with the required relative residual of 10^{-2}, see Fig. 14. WSMP could not be implemented on more than 32 nodes and requiring 86 s ($\sim$21.5 times slower than PSPIKE_NBP) to obtain a solution with relative residual of order 10^{-10}.

Example 3

Using the same linear system $A_2 x_2 = f_2$, we compare the performance of PSPIKE_NBP and the algebraic multigrid preconditioned Krylov subspace solver

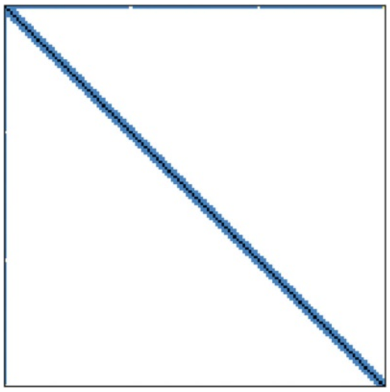
Fig. 11 Sparsity plot of Rajat31 (the figure is obtained from [6])

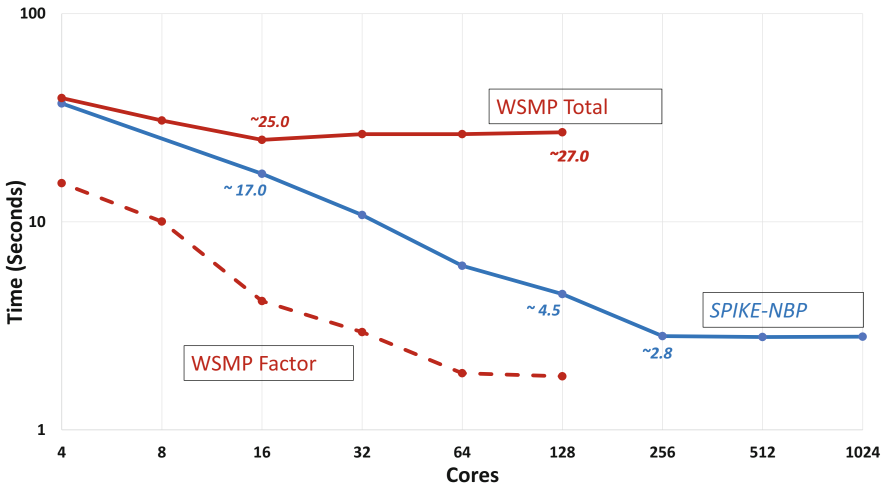

Fig. 12 SPIKE-NBP for Rajat31 system

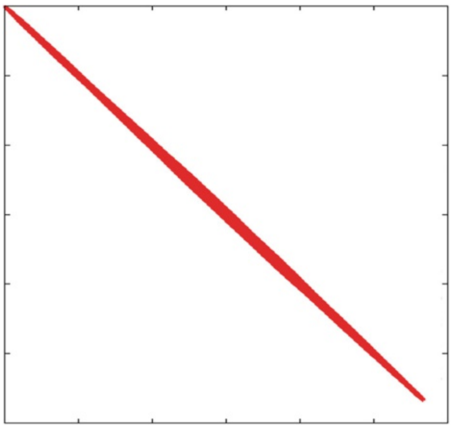
Fig. 13 Sparsity plot of MEMS matrix

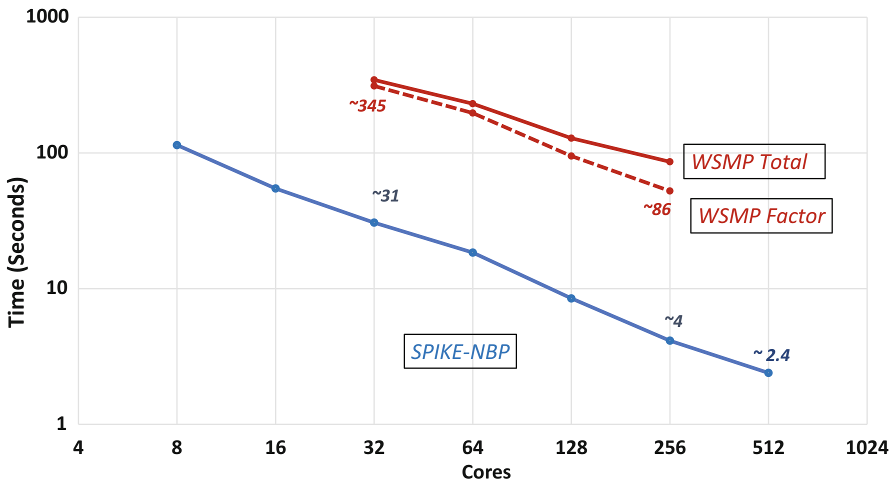

Fig. 14 SPIKE-NBP for MEMS system

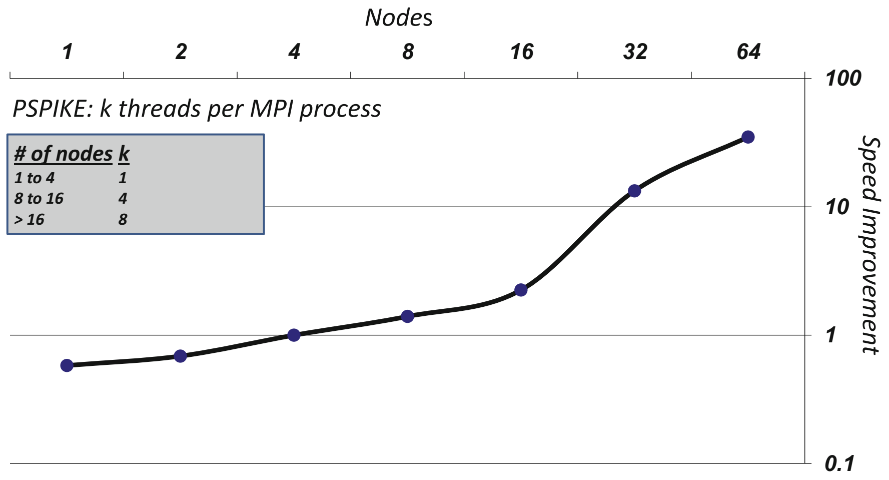

Fig. 15 Speed improvement: time (Trilinos-ML)/time (PSPIKE)

in Trilinos-ML developed at Sandia National Lab. on an Intel cluster of 64 nodes (8 cores/node). Using the Chebyshev smoother for Trilinos-ML, Fig. 15 shows the speed improvement realized by PSPIKE_NBP once we use more than 4 nodes. In PSPIKE we use a hybrid programming paradigm, OpenMP within each node (*k* threads per MPI process) and one node per MPI process with *k* depending on the number of nodes used to obtain a solution with relative residual of order 10^{-10}.

3.3 SPIKE_MBP

Here, we note how a system of the form $Mz = r$ is solved in each iteration of a Krylov subspace method, where $M \in \mathbb{R}^{n\times n}$ is of the form of a block-tridiagonal matrix

$$M = \begin{pmatrix} M_1 & \tilde{B}_1 & & & \\ \tilde{C}_2 & M_2 & \tilde{B}_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \tilde{C}_{k-1} & M_{k-1} & \tilde{B}_{k-1} \\ & & & \tilde{C}_k & M_k \end{pmatrix}, \tag{26}$$

where k is the number of partitions (often chosen as the number of nodes), where each M_j is a large sparse matrix of order $m = \lceil n/k \rceil$, and

$$\tilde{B}_j = \begin{pmatrix} 0 & 0 \\ B_j & 0 \end{pmatrix}, \tilde{C}_j = \begin{pmatrix} 0 & C_j \\ 0 & 0 \end{pmatrix} \tag{27}$$

in which B_j and C_j are dense matrices of order $\nu << m$. Now, $Mz = r$ is solved using the SPIKE algorithm by forming only the reduced system by solving

$$M_j \begin{pmatrix} V_j & W_j \\ * & * \\ \vdots & \vdots \\ * & * \\ V_j' & W_j' \end{pmatrix} = \begin{pmatrix} C_j & 0 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ 0 & B_j \end{pmatrix} \tag{28}$$

only for the tips of the spikes V_j, V_j' and W_j, W_j' via an interesting feature of the sparse direct solver Pardiso. Using the spike tips only, the reduced system is formed and solved via ScaLapack. Once this is achieved, the solution of $Mz = r$ is realized by employing the factors of each M_j obtained by Pardiso.

The description of PSPIKE_MBP is given in more detail with parallel scalability results for large-scale problems in [18] and its application to a PDE-constrained optimization problem in [30]. While Pardiso is primarily suitable for single node platforms, PSPIKE is scalable across multiple nodes. Furthermore, we would like to mention that PSPIKE is capable of using message passing-multithreaded hybrid parallelism. In Fig. 16, we present the required solution time of PSPIKE compared to Pardiso (on one node) for a medium size and large 3D PDE-constrained optimization problems with $75 \times 75 \times 75$ and $150 \times 150 \times 150$ meshes, respectively, using hybrid parallelism with 8 threads (cores) per node. Note that for the larger problem Pardiso runs out of memory due to fill-in. Further details of these problems and the results are given in [30].

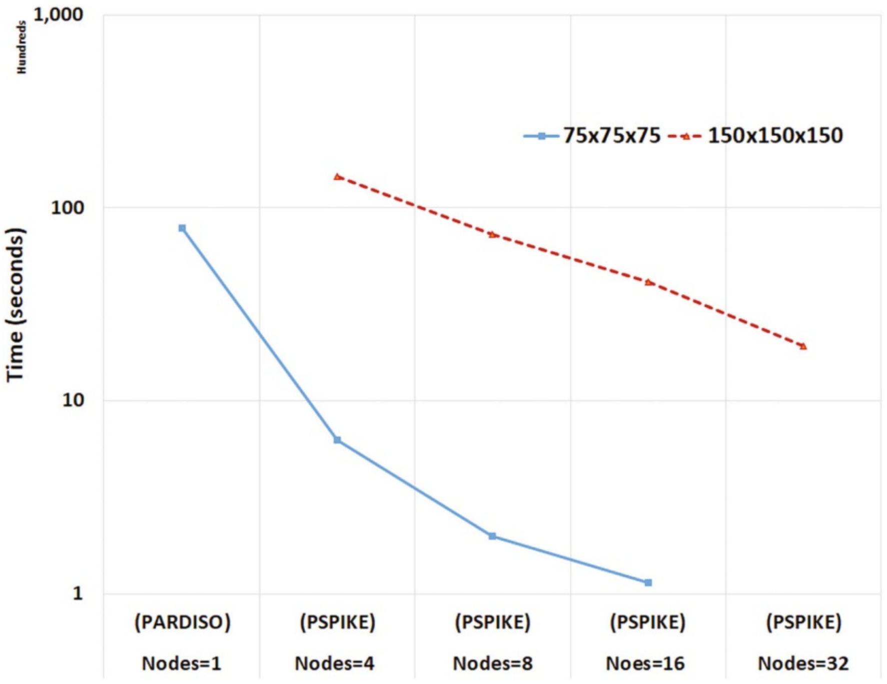

Fig. 16 SPIKE_MBP and Pardiso solution times for the optimization problem

3.4 SPIKE_WBP

For some applications it is not possible to obtain, after reordering, a narrow banded preconditioner, or a block-tridiagonal preconditioner in which the interconnecting off-diagonal blocks are of much smaller size than the diagonal blocks. An example of that is illustrated in Fig. 17. Note that, after reordering, the "heavy" off-diagonal elements (black color) cannot be contained in either of the two previous forms of the preconditioner $M \in \mathbb{R}^{n\times n}$. As an alternative, one way to encapsulate as many of the heavy elements in M is to create a preconditioner that consists of overlapped diagonal blocks, see Fig. 17, for M consisting of two overlapped blocks. In each outer Krylov subspace iteration we solve systems of the form $Mz = r$ via the algorithm given in [22]. Using the two overlapped blocks, $Mz = r$ becomes of the form

$$\begin{pmatrix} M_{11} & M_{12} & \\ M_{21} & M_{22} & M_{23} \\ & M_{32} & M_{33} \end{pmatrix} \tag{29}$$

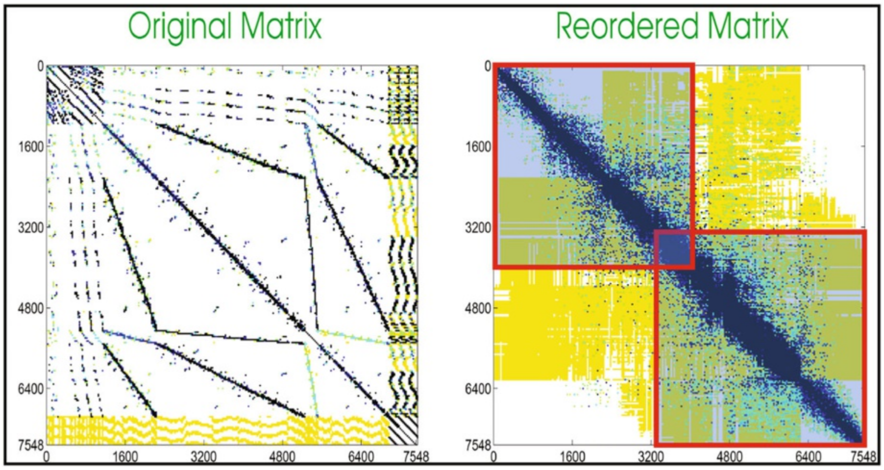

Fig. 17 WBP highlighted after reordering, see [21] for the tearing based parallel hybrid sparse solver

which can be "torn" into two linear systems

$$\begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{22}^{(1)} \end{pmatrix} \begin{pmatrix} z_1(y) \\ z_2^{(1)}(y) \end{pmatrix} = \begin{pmatrix} r_1 \\ \alpha r_2 + y \end{pmatrix} \tag{30}$$

$$\begin{pmatrix} M_{22}^{(2)} & M_{23} \\ M_{32} & M_{33}^{(1)} \end{pmatrix} \begin{pmatrix} z_2^{(2)}(y) \\ z_2^{(3)}(y) \end{pmatrix} = \begin{pmatrix} (1-\alpha) r_2 - y \\ \alpha r_3 \end{pmatrix}, \tag{31}$$

where the overlap matrix $M_{22} = M_{22}^{(1)} + M_{22}^{(2)}$, and $0 < \alpha < 1$. Clearly, we need to choose y so that $z_2^{(1)} = z_2^{(2)}$. Enforcing $z_2^{(1)}(y) = z_2^{(2)}(y)$ results in a linear system $Gy = g$ of size equal to that of overlap matrix M_{22}, $\nu << n$. In solving $Gy = g$ for the unknown y, using a Krylov subspace method, it is shown in [22] that one needs not generate either G or g explicitly, in fact the residual $r(p) = g - G * p$ is given by $[z_2^{(2)}(p) - z_2^{(1)}(p)]$, $r(0) = g = [z_2^{(2)}(0) - z_2^{(1)}(0)]$, and the matrix-vector product $G * g = r(0) - r(g)$. The case of more than two overlapped blocks is considered in detail in [22].

3.5 The General SPIKE

Now we describe the general case where the coefficient matrix has not been subjected to the reordering process described earlier. In other words it is a general sparse matrix and also there are multiple right-hand side vectors. Given a nonsingular linear system of equations,

$$AX = F, \tag{32}$$

where $A \in \mathbb{R}^{n \times n}$ is a general sparse matrix and assume we have m right-hand side vectors F, we can still apply the General SPIKE algorithm as follows. As in the banded case, let us assume A, X, and F are partitioned conformably into k block rows and A is also partitioned into k block columns. The Spike factorization can be described as the factorization of the coefficient matrix [14],

$$A = DS, \tag{33}$$

where D is the block diagonal of A and S is the "spike" matrix. Let $A = D + R$ where R is a matrix that contains elements except diagonal blocks. Assuming D is invertible and using (33) we obtain the spike matrix,

$$S = I + \bar{S}, \tag{34}$$

where $\bar{S} = D^{-1}R$. Note that the diagonal of S consists of ones and the off-diagonals are the spikes ($\bar{S}$). Going back to the original linear system in (32), if we multiply both sides of the equality with D^{-1} from left, we have the modified system

$$SX = G, \tag{35}$$

where $G = D^{-1}F$. The modified system in (35) has the same solution vector, X, as the original system in (32). Furthermore, let idx be the nonzero column indices of R which also correspond to nonzero column indices of $D^{-1}R$. Then, there is an independent subsystem corresponding to the unknowns with row indices idx, i.e. $X(idx, :)$ in (35) such that,

$$\hat{S}\hat{X} = \hat{G}, \tag{36}$$

where $\hat{S} = S(idx, idx)$, $\hat{X} = X(idx, :)$, and $\hat{G} = G(idx, :)$. Dimensions of the reduced system in (36) are $r \times r$ where $r = length(idx)$ with $r \leq n$. After solving the reduced system we can retrieve the remaining unknowns in parallel,

$$X = G - \bar{S}X. \tag{37}$$

Note that we only need a subset of unknowns, $\hat{X}$, to evaluate the right-hand side of the equality since the other columns in $\bar{S}$ are zeros. This approach requires $\bar{S}$ to be computed explicitly. Alternatively, one can obtain the solution by solving the following system in parallel,

$$DX = F - RX. \tag{38}$$

Again, the right-hand side can be evaluated once we obtain $\hat{X}$. In contrast to (37), (38) does not require the computation of $\bar{S}$ completely, even though it still requires the solution of the reduced system involving $\hat{S}$ which may be explicitly formed via partially computing $\bar{S}$. Alternatively, the reduced system can be solved iteratively without forming $\hat{S}$ explicitly. Some of these alternatives might be preferred in practice, depending on the current availability of efficient software tools to perform those operations.

In any case, the size of the reduced system depends on r. A smaller r not only enhances parallelism by enabling a smaller reduced system and less communication requirements, but also reduces the arithmetic complexity in computing $\hat{S}$ and $\bar{S}$ (if needed) as well as the complexity of (37) and (38).

In practice, we assume $r << n$. Ideally, $r = 0$ and some matrices can be reordered into a block diagonal form. In this case, there is no reduced system and the block diagonal systems are solved independently in parallel. Most applications, however, give rise to sparse linear system of equations that does not contain independent blocks, then the objective is to reorder and partition those matrices in such a way that the number of the nonzero columns in $\bar{R}$ is minimized [15].

The main difference between the sparse and the banded SPIKE algorithms is the dependence of the reduced system size (r) on the sparsity structure of the matrix (and hence on the corresponding graph or hypergraph representation of the sparse matrix). Therefore, sparse graph/hypergraph partitioning methods are key ingredients for the algorithm to be scalable and to perform efficiently. METIS [11] and PaToH [4], are suitable for graph and hypergraph partitioning, respectively, and they fit well to the objective of minimizing the reduced system dimension.

To illustrate the algorithm, we give a small (9×9) coefficient matrix, A, in Fig. 18a for simplicity we ignore the numerical values. Given $k = 3$, the coefficient matrix and right-hand side are comformably partitioned,

$$A = \begin{pmatrix} D_{11} & R_{12} & R_{13} \\ R_{21} & D_{22} & R_{23} \\ R_{31} & R_{32} & D_{33} \end{pmatrix} \text{ and } F = \begin{pmatrix} F_1 \\ F_2 \\ F_3 \end{pmatrix}. \tag{39}$$

The set of indices of nonzero columns of $\bar{R}$ are $idx = \{1, 5, 8\}$. After partitioning, S and G can be computed as follows:

$$S = \begin{pmatrix} I & D_{11}^{-1}R_{12} & D_{11}^{-1}R_{13} \\ D_{22}^{-1}R_{21} & I & D_{22}^{-1}R_{23} \\ D_{33}^{-1}R_{31} & D_{33}^{-1}R_{32} & I \end{pmatrix} \text{ and } G = \begin{pmatrix} D_{11}^{-1}F_1 \\ D_{22}^{-1}F_2 \\ D_{33}^{-1}F_3 \end{pmatrix}. \tag{40}$$

If the right-hand side vector is available immediately, the computation involved is the solution of independent linear systems with multiple right-hand sides,

$$D_{11}[S_{12}, S_{13}, G_1] = [R_{12}, R_{13}, F_1], \tag{41}$$

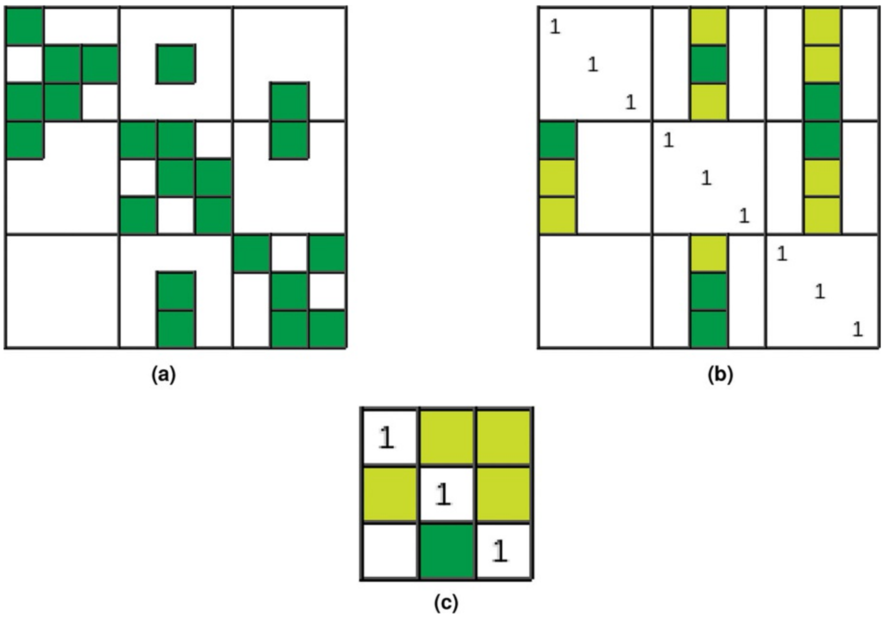

Fig. 18 A, S, and $\hat{S}$ for the small example. (**a**) Coefficient matrix (A). (**b**) Spike matrix (S). (**c**) Reduced system coefficient matrix ($\hat{S}$)

$$D_{22}[S_{21}, S_{23}, G_2] = [R_{21}, R_{23}, F_2], \tag{42}$$

$$D_{33}[S_{31}, S_{32}, G_3] = [R_{31}, R_{32}, F_3]. \tag{43}$$

Note that in (41),(42), and (43) only a few columns of $R_{ij,i \neq j}$ are nonzero and the rest are zeros. We do not store or perform operations with zero columns since the corresponding solution vector is already zero. The resulting S matrix is shown in Fig. 18b. Light green elements are fill-ins and some of them can be negligible as in the banded case [20, 24] if A is diagonally dominant or near diagonally dominant.

Further savings can be obtained, if a sparse solver with sparse right-hand side vectors is available and if it is capable of solving only for a few unknowns, one can compute only those components of vectors in S_{ij} that is required for forming the reduced system (defined by idx). One of the implementation of the General SPIKE algorithm in [3] performs partial solves via the sparse right-hand side feature of PARDISO [29]. Next, we can form the reduced system explicitly by selecting $\hat{S} = S(idx, idx)$ and $\hat{G} = G(idx, :)$ and solve the reduced system, (36), to obtain $\hat{X} = X(idx, :)$. $\hat{S}$ for the small example is shown in Fig. 18c. The complete solution is obtained in parallel via either:

$$X = G - \bar{S}X \tag{44}$$

Algorithm 1: General spike algorithm

```
1: procedure GeneralSpike($A$, $X$, $F$, $k$)    ▷ to solve $AX = F$ with $k$ partitions
2:     $D + R \leftarrow A$
3:     Identify nonzero columns of $R$ and store their indices in $idx$
4:     $D[S_{(:,idx)}, G] = [R_{(:,idx)}, F]$, solve for:
         • $[S_{(:,idx)}, G]$ (full solve) or
         • $[S_{(idx,idx)}, G_{(idx,:)}]$ (partial solve)
5:     $S_{(idx,idx)} X_{(idx,:)} = G_{(idx,:)}$ (Solve for $X_{(idx,:)}$)
6:     Retrieve the solution vector ($X$):
         • $X \leftarrow G - S_{(:,idx)} X_{(idx)}$ if $S_{(:,idx)]}$ is available
         • $DX = [F - R_{(:,idx)} x_{(idx)}]$ (Solve for $X$), otherwise
7: end procedure
```

or

$$DX = F - RX. \tag{45}$$

The former is preferred if the spikes are formed explicitly since the multiplication $\bar{S}X$ can be implemented using dense matrix-vector (BLAS Level 2) or matrix-matrix (BLAS Level 3) operations, for $m = 1$ and $m > 1$, respectively. The latter requires sparse matrix-dense matrix multiplications (RX), followed by the solution of independent sparse linear systems. It is preferred if the spikes are not explicitly available. The pseudocode of the algorithm is summarized in Algorithm 1.

Numerical results and the performance of this scheme are given in [14] in the context of a parallel solver for the preconditioned linear system and in [3] as a direct multithreaded recursive parallel sparse solver. Furthermore, a multithreaded general sparse triangular solver is proposed in [5].

4 Conclusions

The SPIKE algorithm for banded linear systems that are dense withing the band has been shown to be competitive in parallel scalability with the parallel banded solver in ScaLapack on a variety of parallel architectures. Also, the hybrid PSPIKE (Pardiso-SPIKE) algorithm for large sparse linear systems has proven to be equally competitive with: (1) direct sparse solvers such as Pardiso and WSMP if one requires only approximate solutions that correspond to modest relative residuals, and (2) black-box preconditioned Krylov subspace methods including algebraic multigrid preconditioners.

Acknowledgments The authors would like to thank Drs. Maxim Naumov and Faisal Saied for performing many of the numerical experiments reported in this chapter.

References

1. *SPIKE openMP package*. http://www.spike-solver.org/.
2. E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, third ed., 1999.
3. E. S. Bolukbasi and M. Manguoglu, *A multithreaded recursive and nonrecursive parallel sparse direct solver*, in Advances in Computational Fluid-Structure Interaction and Flow Simulation, Springer, 2016, pp. 283–292.
4. U. V. Catalyurek and C. Aykanat, *Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication*, IEEE Transactions on parallel and distributed systems, 10 (1999), pp. 673–693.
5. I. Cugu and M. Manguoglu, *A parallel multithreaded sparse triangular linear system solver*, Computers & Mathematics with Applications, (2019).
6. T. A. Davis and Y. Hu, *The University of Florida Sparse Matrix Collection*, ACM Transactions on Mathematical Software (TOMS), 38 (2011), p. 1.
7. T. A. Davis and Y. Hu, *The University of Florida Sparse Matrix Collection*, ACM Trans. Math. Softw., 38 (2011), pp. 1:1–1:25.
8. I. S. Duff and J. Koster, *The design and use of algorithms for permuting large entries to the diagonal of sparse matrices*, SIAM Journal on Matrix Analysis and Applications, 20 (1999), pp. 889–901.
9. M. Fiedler, *Algebraic connectivity of graphs*, Czechoslovak Mathematical Journal, 23 (1973), pp. 298–305.
10. E. Gallopoulos, B. Philippe, and A. H. Sameh, *Parallelism in matrix computations*, Springer, 2016.
11. G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM Journal on scientific Computing, 20 (1998), pp. 359–392.
12. D. H. Lawrie and A. H. Sameh, *The computation and communication complexity of a parallel banded system solver*, ACM Transactions on Mathematical Software (TOMS), 10 (1984), pp. 185–195.
13. L. Liu, Z. Li, and A. H. Sameh, *Analyzing memory access intensity in parallel programs on multicore*, in Proceedings of the 22nd annual international conference on Supercomputing, ACM, 2008, pp. 359–367.
14. M. Manguoglu, *A domain-decomposing parallel sparse linear system solver*, Journal of computational and applied mathematics, 236 (2011), pp. 319–325.
15. ——, *A parallel sparse solver and its relation to graphs*, in CEM'11 Computational Electromagnetics International Workshop, IEEE, 2011, pp. 91–94.
16. M. Manguoglu, E. Cox, F. Saied, and A. Sameh, *TRACEMIN-Fiedler: A Parallel Algorithm for Computing the Fiedler Vector*, High Performance Computing for Computational Science–VECPAR 2010, (2011), pp. 449–455.
17. M. Manguoglu, F. Saied, A. Sameh, and A. Grama, *Performance models for the spike banded linear system solver*, Scientific Programming, 19 (2011), pp. 13–25.
18. M. Manguoglu, A. H. Sameh, and O. Schenk, *Pspike: A parallel hybrid sparse linear system solver*, in European Conference on Parallel Processing, Springer, 2009, pp. 797–808.
19. K. Mendiratta and E. Polizzi, *A threaded spike algorithm for solving general banded systems*, Parallel Computing, 37 (2011), pp. 733–741. 6th International Workshop on Parallel Matrix Algorithms and Applications (PMAA'10).
20. C. C. K. Mikkelsen and M. Manguoglu, *Analysis of the truncated SPIKE algorithm*, SIAM Journal on Matrix Analysis and Applications, 30 (2008), pp. 1500–1519.
21. M. Naumov, M. Manguoglu, and A. H. Sameh, *A tearing-based hybrid parallel sparse linear system solver*, J. Computational Applied Mathematics, 234 (2010), pp. 3025–3038.

22. M. Naumov and A. H. Sameh, *A tearing-based hybrid parallel banded linear system solver*, Journal of Computational and Applied Mathematics, 226 (2009), pp. 306–318.
23. E. Polizzi and A. Sameh, *SPIKE: A parallel environment for solving banded linear systems*, Computers & Fluids, 36 (2007), pp. 113–120. Challenges and Advances in Flow Simulation and Modeling.
24. E. Polizzi and A. H. Sameh, *A parallel hybrid banded system solver: the SPIKE algorithm*, Parallel computing, 32 (2006), pp. 177–194.
25. Y. Saad, *Iterative methods for sparse linear systems*, vol. 82, siam, 2003.
26. A. Sameh and Z. Tong, *The trace minimization method for the symmetric generalized eigenvalue problem*, J. Comput. Appl. Math., 123 (2000), pp. 155–175.
27. A. H. Sameh and D. J. Kuck, *On stable parallel linear system solvers*, Journal of the ACM (JACM), 25 (1978), pp. 81–91.
28. A. H. Sameh and J. A. Wisniewski, *A trace minimization algorithm for the generalized eigenvalue problem*, SIAM Journal on Numerical Analysis, 19 (1982), pp. 1243–1259.
29. O. Schenk and K. Gärtner, *Solving unsymmetric sparse systems of linear equations with PARDISO*, Future Generation Computer Systems, 20 (2004), pp. 475–487.
30. O. Schenk, M. Manguoglu, A. Sameh, M. Christen, and M. Sathe, *Parallel scalable PDE-constrained optimization: antenna identification in hyperthermia cancer treatment planning*, Computer Science-Research and Development, 23 (2009), pp. 177–183.
31. B. S. Spring, E. Polizzi, and A. H. Sameh, *A feature complete SPIKE banded algorithm and solver*, CoRR, abs/1811.03559 (2018).