Contents lists available at ScienceDirect

Journal of Computational and Applied Mathematics

journal homepage: www.elsevier.com/locate/cam

PSPIKE+: A family of parallel hybrid sparse linear system solvers

Yao Zhu[*], Ahmed H. Sameh

Purdue University, Department of Computer Science, 305 N. University Street, West Lafayette, IN 47907, USA

HIGHLIGHTS

- We present the PSPIKE+ family of parallel hybrid SPARSE linear system solvers.
- We propose the G-PAVER reordering for effective central banded preconditioners.
- We develop a hybrid MPI/OpenMP based parallel implementation of PSPIKE+.
- We demonstrate the robustness of PSPIKE+ on miscellaneous sparse linear systems.
- We show the good parallel scalability of PSPIKE+ on distributed-memory architectures.

ARTICLE INFO

Article history:
Received 22 May 2015
Received in revised form 1 April 2016

Keywords:
Parallel hybrid linear system solvers
Reordering
Graph partitioning
Vertex cover
Overlapping diagonal block preconditioner
Krylov subspace methods

ABSTRACT

We present PSPIKE+, as a family of parallel hybrid linear system solvers that are more robust than other available preconditioned iterative methods, and more scalable than parallel sparse direct solvers. A critical step in using PSPIKE+ is the extraction of a preconditioner that is in the form of a generalized central band which consists of a set of overlapping diagonal blocks that encapsulate as many of the heaviest elements of the coefficient matrix. To address this issue, we propose the G-PAVER reordering scheme that produces an effective preconditioner in the form of overlapping diagonal blocks while simultaneously accommodating load balancing on distributed-memory architectures and controlling overlap sizes. In each outer Krylov subspace iteration for solving the linear system, we need to solve linear systems involving the preconditioner. This is accomplished using a parallel tearing-based method, which may be regarded as an algebraic domain decomposition scheme. The tearing-based method requires the solution of a much smaller linear system (we call the balance system) whose size is equal to the sum of the overlaps. In this paper, we form the balance system explicitly and solve it directly. Our parallel implementation of PSPIKE+ based on G-PAVER, PARDISO 5.0.0, and solving the explicit balance system results in impressive robustness and good parallel scalability on computing platforms with a limited number of multi-core nodes.

1. Introduction

It has been shown that the parallel hybrid linear system solver, PSPIKE [1–3], achieves impressive robustness and high parallel scalability on modern distributed memory parallel architectures. PSPIKE depends on Fiedler reordering to enhance the dominance of either a narrow or medium central bands [1,2] that can be used as preconditioners to an outer Krylov

[*] Corresponding author.
E-mail addresses: yaozhu@purdue.edu (Y. Zhu), sameh@cs.purdue.edu (A.H. Sameh).

http://dx.doi.org/10.1016/j.cam.2016.08.033
0377-0427/© 2016 Elsevier B.V. All rights reserved.

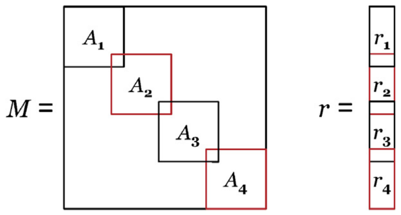


Fig. 1. Partitioning of the matrix and the RHS with overlapping, with $p = 4$.

subspace iteration. By medium-banded, we mean that the preconditioner consists of non-overlapping diagonal blocks plus small off-diagonal coupling blocks [3]. In this paper, we develop a new family of parallel hybrid linear system solvers, which we call PSPIKE+. In contrast to PSPIKE, PSPIKE+ adopts as preconditioner a generalized central band consisting of overlapping diagonal blocks that encapsulate as many of the heaviest elements as possible. In this sense, PSPIKE+ subsumes both the narrow and the medium-banded forms as special cases.

In Section 2, we describe the different stages of PSPIKE+. We propose a new sparse matrix reordering scheme G-PAVER that accommodates both load balancing on distributed-memory parallel architecture, as well as ease of controlling overlap sizes. G-PAVER relies on graph partitioning and vertex cover based overlapping (Section 3). The concept of reordering schemes in conjunction with algebraic domain decomposition and multilevel methods is not new. For example, in [4] the *ARMS reordering* based on breadth-first search with thresholding is proposed for finding block independent sets [5,4]. The ARMS reordering is utilized in [6] for constructing multilevel approximate inverses.

In solving linear systems involving the preconditioner produced by G-PAVER (in the form of overlapped diagonal blocks), we use the tearing-based method [7,8]. Unlike the approach used in [7,8] in which the balance system is not formed explicitly and solved iteratively, we form the balance system explicitly and solve it directly via block LU-factorization with diagonal boosting (Section 4). This direct approach is most suitable for computing platform with a limited number of multi-core nodes. On such architectures, this explicit approach to solving the balance system enhances the robustness of PSPIKE+ without necessarily incurring a much higher overhead, especially when the overlap sizes are relatively small. On platforms with many nodes, however, such an explicit approach is not ideal for achieving parallel scalability. The parallel implementation of PSPIKE+ based on G-PAVER, PARDISO 5.0.0, and the direct solution of the balance system is outlined in Section 5. The pivot perturbation strategy of PARDISO 5.0.0 helps to enhance the applicability of PSPIKE+ even in the case of having singular overlapping diagonal blocks. Further, we exploit the PARDISO 5.0.0 capability of extracting only the top and bottom tips of the solution to a linear system in which the right hand side is all zeros except an identical top and bottom tips. This has proven to be a very economical way of forming the balance system explicitly. Our numerical experiments demonstrate the utility of G-PAVER reordering, and the robustness and parallel scalability of PSPIKE+ on a distributed memory computing platform with 8 multi-core nodes (Section 6).

2. The PSPIKE+ algorithm

Consider the problem of solving the linear system of equations

$$\bar{A}\bar{x} = \bar{f} \tag{1}$$

where $\bar{A} \in \mathbb{R}^{n\times n}$ is nonsingular. The PSPIKE+ algorithm consists of three stages: (1) pre-processing the linear system; (2) constructing a preconditioner M as a set of *overlapping diagonal blocks* (ODB); and (3) outer iterations via a Krylov subspace method (e.g., BiCGstab) preconditioned by the ODB preconditioner, where in each iteration we need to solve systems of the form $Mz = r$. An example of the ODB form of the preconditioner M with 4 diagonal blocks is shown in Fig. 1.

2.1. Pre-processing stage

The purpose of the pre-processing stage is to reorder the sparse linear system (1) such that an effective and well conditioned ODB preconditioner could be constructed. In PSPIKE+, we adopt two reordering steps: column permutation, and symmetric reordering.

2.1.1. Column permutation

We use the HSL subroutine MC64 to permute the columns of $\bar{A}$ so as to maximize the product of the magnitudes of the diagonal elements, i.e. using the option of maximum product traversal [9]. MC64 also has the option of scaling the rows and columns of $\bar{A}$ such that the scaled and permuted matrix is then an I-matrix [10]. Let D_r be the diagonal row scaling matrix, D_c be the diagonal column scaling matrix, and Q be the column permutation matrix returned from MC64, the linear system resulting from applying MC64 is then given by

$$\bar{A}_s\bar{x}_s = \bar{f}_s \tag{2}$$

where

$$\bar{A}_s = \left(D_r \bar{A} D_c\right) Q^{\mathrm{T}} \tag{3}$$

and

$$\bar{f}_s = D_r \bar{f}.$$

Once the linear system (2) is solved, we can recover the solution to (1) via

$$\bar{x} = D_c Q^{\mathrm{T}} \bar{x}_s. \tag{4}$$

If we choose not to apply the MC64 scaling, then $D_r = D_c = I_n$, the $n \times n$ identity matrix.

2.1.2. Symmetric reordering

We further apply a symmetric reordering to $\bar{A}_s$. Let

$$A = P\bar{A}_s P^{\mathrm{T}}.$$

The symmetric reordering P is motivated by the following two objectives

- The sizes of the overlapping diagonal blocks in the ODB preconditioner do not vary significantly to assure adequate load-balancing.
- The ODB preconditioner extracted from A after the symmetric reordering P will encapsulate as many of the heaviest off-diagonal elements as possible.

The final linear system resulting from these two reordering schemes is given by

$$Ax = f \tag{5}$$

where

$$f = P\bar{f}_s$$
$$x = P\bar{x}_s. \tag{6}$$

Linear system (5) is then solved using a Krylov subspace method with the ODB preconditioner.

2.2. Constructing the ODB preconditioner

The coefficient matrix A resulting from the *pre-processing* stage (see Fig. 2(a) as an example) can be split as

$$A = M + E \tag{7}$$

where M is a central generalized band extracted from A in the ODB form, and E consists of the off-diagonal elements such that $\|E\|_F \ll \|M\|_F$. Furthermore, E is much sparser than A, and could be of a much lower rank. We denote the overlapping diagonal blocks by $A_1, \ldots, A_p$, where p is the number of blocks (see Fig. 1), and the size of the overlap between two consecutive overlapping diagonal blocks A_k and A_{k+1} by τ_k, for $k = 1, \ldots, (p-1)$.

As an example, in Fig. 2 we plot the sparsity patterns of A, M, and E of the coefficient matrix cage13 (with $n = 445{,}315$) from The University of Florida sparse matrix collection [11]. We can see that after the pre-processing stage, $\|E\|_F = 13.73 \ll \|M\|_F = 460.86$, and $nnz(E) = 462{,}024 \ll nnz(A) = 7{,}479{,}343$.

2.2.1. Factorization of the diagonal blocks

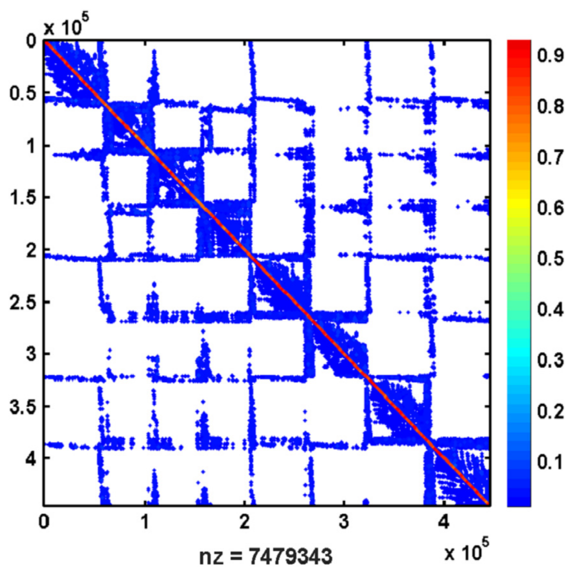
Once the ODB preconditioner M is extracted and torn into the p diagonal blocks A_k for $k = 1, \ldots, p$, we obtain the LU-factorization of each diagonal block using a sparse direct solver. In case a certain A_k is singular, we obtain the nonsingular L and U factors of its slight perturbation via diagonal pivoting with pivot perturbation, as is implemented in the sparse direct solver PARDISO 5.0.0. As will be detailed in the following, solving systems $Mz = r$ in each outer Krylov subspace iteration requires solving systems involving the LU factors of each A_k or its slight perturbation. The LU-factorization of the diagonal blocks is done only once and reused for all the following repeated solutions.

2.3. Krylov subspace method with the ODB preconditioner

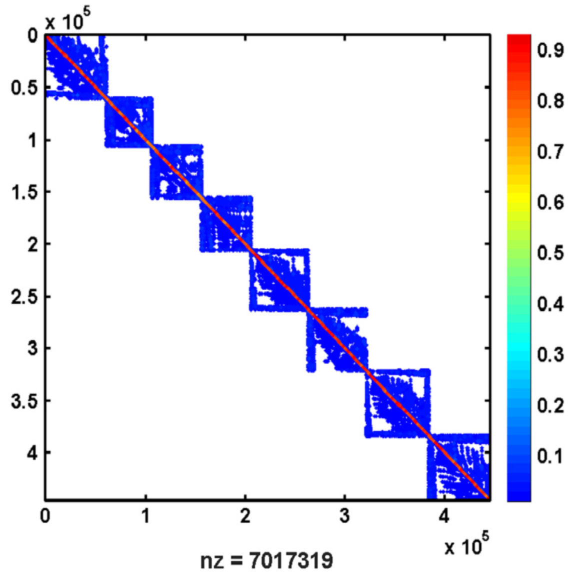
We solve the pre-processed linear system (5) iteratively using a Krylov subspace method (e.g., BiCGstab) preconditioned by the ODB preconditioner M. In the following, this is referred to as the *outer* Krylov subspace method. The preconditioning step

$$Mz = r \tag{8}$$

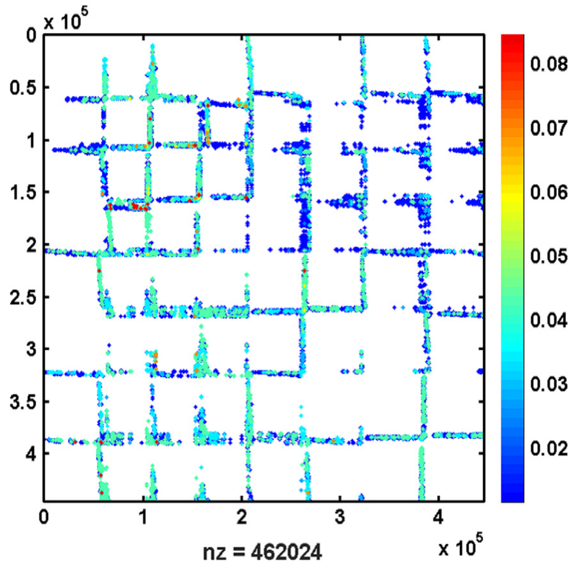
is solved using the tearing-based method proposed in [7]. We briefly outline the tearing-based method here. Suppose M is torn into p overlapping diagonal blocks $A_1, \ldots, A_p$, where



(a) $\|A\|_F = 461.07$.



(b) $\|M\|_F = 460.86$.



(c) $\|E\|_F = 13.73$.

Fig. 2. Sparsity patterns of A, M, and E on `cage13`. The nonzeros are colored with respect to the magnitudes. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

$$A_1 = \begin{pmatrix} A_{22}^{(1)} & A_{23}^{(1)} \\ A_{32}^{(1)} & A_{33}^{(1)} \end{pmatrix} \qquad A_p = \begin{pmatrix} A_{11}^{(p)} & A_{12}^{(p)} \\ A_{21}^{(p)} & A_{22}^{(p)} \end{pmatrix}$$

$$A_k = \begin{pmatrix} A_{11}^{(k)} & A_{12}^{(k)} & A_{13}^{(k)} \\ A_{21}^{(k)} & A_{22}^{(k)} & A_{23}^{(k)} \\ A_{31}^{(k)} & A_{32}^{(k)} & A_{33}^{(k)} \end{pmatrix} \quad \text{for } k = 2, \ldots, (p-1)$$

in which $A_{33}^{(k)} \in \mathbb{R}^{\tau_k \times \tau_k}$ and $A_{11}^{(k+1)} \in \mathbb{R}^{\tau_k \times \tau_k}$ are overlapping for $k = 1, \ldots, (p-1)$. The main step of the tearing-based method is to solve the *balance system*

$$By = g \tag{9}$$

which assures the solution of the torn system is the same as the solution to system (8). For the sake of illustration, we examine the case of $p = 2$ for deriving the balance system. The linear system (8) is torn into 2 linear systems

$$\begin{pmatrix} A_{22}^{(1)} & A_{23}^{(1)} \\ A_{32}^{(1)} & A_{33}^{(1)} \end{pmatrix} \begin{pmatrix} z_2^{(1)} \\ z_3^{(1)} \end{pmatrix} = \begin{pmatrix} r_2^{(1)} \\ r_3^{(1)} + y \end{pmatrix} \tag{10}$$

and

$$\begin{pmatrix} A_{11}^{(2)} & A_{12}^{(2)} \\ A_{21}^{(2)} & A_{22}^{(2)} \end{pmatrix} \begin{pmatrix} z_1^{(2)} \\ z_2^{(2)} \end{pmatrix} = \begin{pmatrix} r_1^{(2)} - y \\ r_2^{(2)} \end{pmatrix} \tag{11}$$

where we need find the subvector y such that

$$z_3^{(1)} = z_1^{(2)}. \tag{12}$$

Let the inverse of the 2×2 block matrices in (10) and (11) be

$$\begin{pmatrix} A_{22}^{(1)} & A_{23}^{(1)} \\ A_{32}^{(1)} & A_{33}^{(1)} \end{pmatrix}^{-1} = \begin{pmatrix} B_{22}^{(1)} & B_{23}^{(1)} \\ B_{32}^{(1)} & B_{33}^{(1)} \end{pmatrix} \qquad \begin{pmatrix} A_{11}^{(2)} & A_{12}^{(2)} \\ A_{21}^{(2)} & A_{22}^{(2)} \end{pmatrix}^{-1} = \begin{pmatrix} B_{11}^{(2)} & B_{12}^{(2)} \\ B_{21}^{(2)} & B_{22}^{(2)} \end{pmatrix}. \tag{13}$$

After solving for $z_3^{(1)}$ in (10) and $z_1^{(2)}$ in (11) using (13) and then substituting into (12), we obtain the balance system of $p = 2$ as follows

$$(B_{33}^{(1)} + B_{11}^{(2)})y = (B_{11}^{(2)} r_1^{(2)} + B_{12}^{(2)} r_2^{(2)}) - (B_{32}^{(1)} r_2^{(1)} + B_{33}^{(1)} r_3^{(1)}).$$

For a general p, the coefficient matrix B is in block tridiagonal form [7]

$$B = \begin{pmatrix} B_{33}^{(1)} + B_{11}^{(2)} & -B_{13}^{(2)} & & & \\ -B_{31}^{(2)} & B_{33}^{(2)} + B_{11}^{(3)} & -B_{13}^{(3)} & & \\ & & \ddots & & \\ & & -B_{31}^{(p-2)} & B_{33}^{(p-2)} + B_{11}^{(p-1)} & -B_{13}^{(p-1)} \\ & & & -B_{31}^{(p-1)} & B_{33}^{(p-1)} + B_{11}^{(p)} \end{pmatrix} \tag{14}$$

of which the size is $\sum_{k=1}^{p-1} \tau_k$. The blocks $B_{11}^{(k)} \in \mathbb{R}^{\tau_{k-1} \times \tau_{k-1}}$, $B_{31}^{(k)} \in \mathbb{R}^{\tau_k \times \tau_{k-1}}$, $B_{13}^{(k)} \in \mathbb{R}^{\tau_{k-1} \times \tau_k}$, and $B_{33}^{(k)} \in \mathbb{R}^{\tau_k \times \tau_k}$ are defined by solving the linear systems

$$A_k \begin{pmatrix} B_{11}^{(k)} \\ * \\ B_{31}^{(k)} \end{pmatrix} = \begin{pmatrix} I_{\tau_{k-1}} \\ 0 \\ 0 \end{pmatrix} \qquad A_k \begin{pmatrix} B_{13}^{(k)} \\ * \\ B_{33}^{(k)} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ I_{\tau_k} \end{pmatrix}. \tag{15}$$

We enforce $\tau_k \ll n$, for $k = 1, \dots, (p-1)$. Thus the size of the balance system (9) is $\sum_{k=1}^{p-1} \tau_k \ll n$ when p is not large. The RHS of the balance system (9) is defined by

$$g = \begin{pmatrix} h_1^{(2)} - h_3^{(1)} \\ \vdots \\ h_1^{(p)} - h_3^{(p-1)} \end{pmatrix}$$

where the subvectors $h_1^{(k)} \in \mathbb{R}^{\tau_{k-1} \times 1}$, $h_3^{(k)} \in \mathbb{R}^{\tau_k \times 1}$ are defined by solving the linear systems

$$A_k \begin{pmatrix} h_1^{(k)} \\ h_2^{(k)} \\ h_3^{(k)} \end{pmatrix} = \begin{pmatrix} r_1^{(k)} \\ r_2^{(k)} \\ r_3^{(k)} \end{pmatrix} \tag{16}$$

for $k = 1, \dots, p$.

Theorem 2.1 (*Theorem 3 of* [7]). *If M is nonsingular with partitions A_k for $k = 1, \dots, p$ that are also nonsingular, then the coefficient matrix B in (14) is nonsingular.*

Let the solution to the balance system (9) be given by

$$y = \begin{pmatrix} y_1 \\ \vdots \\ y_{p-1} \end{pmatrix} \tag{17}$$

the tearing-based method then decomposes the linear system (8) into p independent linear systems [7]

$$A_k \begin{pmatrix} z_1^{(k)} \\ z_2^{(k)} \\ z_3^{(k)} \end{pmatrix} = \begin{pmatrix} r_1^{(k)} - y_{k-1} \\ r_2^{(k)} \\ r_3^{(k)} + y_k \end{pmatrix} \tag{18}$$

for $k = 1, \ldots, p$ with $y_0 = y_p = 0$.

2.4. PSPIKE+ as a family of parallel hybrid linear system solvers

In each stage of the PSPIKE+ algorithm, different options are available for a robust and efficient implementation on parallel architectures. The choices depend on the properties of the linear systems and the parallel architectures.

- Symmetric reordering P. Possible choices of P include the reverse Cuthill–McKee ordering [12], and the weighted spectral ordering [13,14]. Also, recent work [15] aims at permuting the entire matrix into the ODB form. In this paper, we propose a reordering method that aims at producing an effective generalized central band preconditioner in the ODB form. As a side effect, this reordering method also turns out to enhance the success of block Jacobi preconditioners.
- Factorization of diagonal blocks. When the overlapping diagonal blocks A_k, $k = 1, \ldots, p$ are general sparse matrices, we obtain the LU-factorization of them (or their slight perturbations) using a sparse direct solver, e.g., [16–19].
- Solution scheme of the balance system (9). In this paper, we propose to explicitly form the coefficient matrix B (14) and solve the balance system (9) using block LU-factorization with a "diagonal boosting" strategy.

3. The G-PAVER reordering

In this section, we propose a new reordering strategy that facilitates the extraction of an ODB preconditioner M so as to assure load balance. This reordering is done through two steps: (1) graph partitioning; and (2) vertex cover based overlapping. Because of its use of Graph PArtitioning and VERtex cover, we name it G-PAVER. In the following, we let $\bar{A}_s$ be the coefficient matrix on which the G-PAVER reordering is to be applied.

3.1. Graph partitioning

The purpose of graph partitioning is to extract from $\bar{A}_s$ a sequence of p non-overlapping diagonal blocks $C_1, \ldots, C_p$ such that

- The non-overlapping diagonal blocks are of almost equal size and sparsity.
- The non-overlapping diagonal blocks include as many of the heaviest elements of $\bar{A}_s$ as possible.

We first consider the load balancing issue. For a parallel hybrid solver, an accurate estimate of the workload involving a diagonal block C_k should consider

1. The work of factorizing C_k, which not only depends on the number-of-nonzeros (*nnz*) of C_k, but also the fill-ins incurred during factorization.
2. The work involved in matrix–vector multiplication, which is determined by the *nnz* of C_k.

As pointed in [20], in domain decomposition methods it is difficult to estimate the fill-ins without doing actual fill-reduce ordering followed by symbolic factorization. In [20], a predictor–corrector framework is proposed to iteratively refine an initial graph partitioning in order to balance such fill-ins. In [21] a different approach based on recursive bisection is taken, where the partial partition information from one level is used to set up constraints for the bisection at the next level. Although the methods proposed in [20,21] are insightful, they are expensive to implement and deploy in practice. In this paper, we propose to use the *nnz* of C_k as a coarse approximation of the work required for factorizing it.

We then consider the issue of extracting non-overlapping diagonal blocks $C_1, \ldots, C_p$ containing elements of large magnitude. Effectively this is a graph partitioning problem because C_k and C_l do not overlap for $k \neq l$. Moreover, we want to maximize the total magnitude of elements belonging to C_k for $k = 1, \ldots, p$. This objective together with the load balancing requirement on C_k ($k = 1, \ldots, p$) as described above is then formalized as the following *p-way graph partitioning by edge separators* problem.

Definition 3.1 (*p-GPES*)**.** Given an undirected and weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with weight function $w(i, j) : \mathcal{E} \to \mathbb{R}^+$, the p-way graph partitioning by edge separators (p-GPES) problem is to partition $\mathcal{V}$ into p disjoint subsets $\mathcal{V} = \mathcal{V}_1 \cup \cdots \cup \mathcal{V}_p$, such

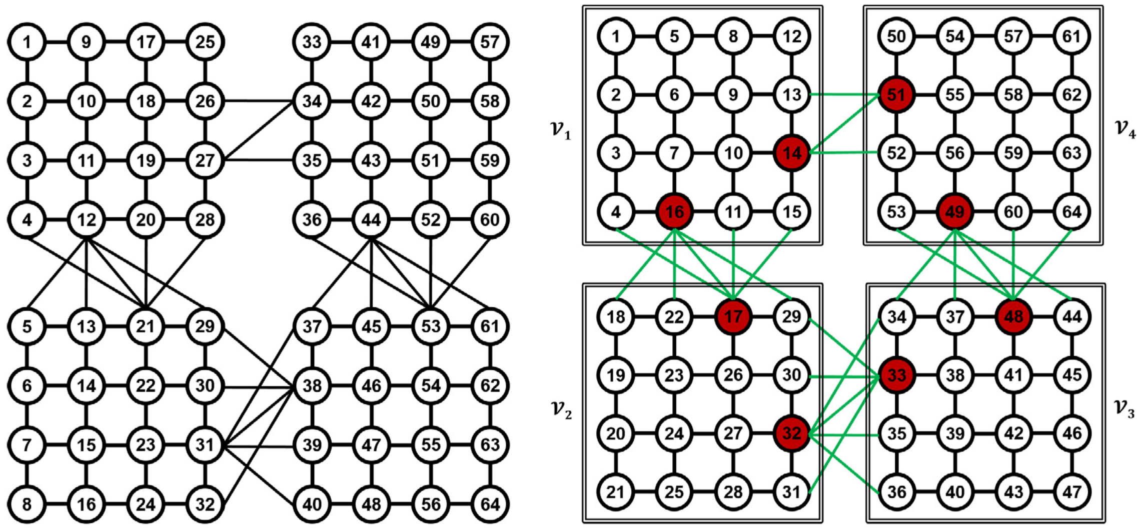


Fig. 3. An example of G-PAVER reordering, with $p = 4$. In 3(a), the nodes are numbered according to a column-major order. In 3(b), the graph is partitioned into 4 parts $\mathcal{V}_1$, $\mathcal{V}_2$, $\mathcal{V}_3$ and $\mathcal{V}_4$. All the cut edges are in green. The nodes forming the vertex cover of the cut edges are in red. The parts $\mathcal{V}_1$, $\mathcal{V}_2$, $\mathcal{V}_3$ and $\mathcal{V}_4$ are linear ordered, and the nodes are numbered according to a G-PAVER reordering. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

that the total weight of edges crossing different subsets is minimized, while the volumes of the subsets $\text{vol}(\mathcal{V}_k)$, $k = 1, \dots, p$ are balanced.

When using p-GPES for extracting non-overlapping diagonal blocks $C_1, \dots, C_p$ from a matrix $\bar{A}_s$, we let the undirected and weighted graph be denoted by $\mathcal{G}[\bar{A}_s]$ whose adjacency matrix is $(|\bar{A}_s| + |\bar{A}_s^{\mathsf{T}}|)/2$. Thus, we have the weight function

$$w(i,j) = \frac{|\bar{A}_s|_{ij} + |\bar{A}_s|_{ji}}{2}.$$

Because we approximate the workload involving C_k using its nnz, we define the volume function on $\mathcal{V}_k$ as follows

$$\text{vol}(\mathcal{V}_k) = \sum_{i \in \mathcal{V}_k} \left| \left\{ j \neq i : |\bar{A}_s|_{ij} + |\bar{A}_s|_{ji} \neq 0 \right\} \right|.$$

Then from the p-GPES result, we define the non-overlapping diagonal blocks C_k to be that submatrix of $\bar{A}_s$ with row and column indices $i \in \mathcal{V}_k$, denoted as $\bar{A}_s(\mathcal{V}_k, \mathcal{V}_k)$, for $k = 1, \dots, p$. A p-GPES result induces a family of reorderings under which the indices in the same subset $\mathcal{V}_k$ are numbered contiguously.

Definition 3.2 (*p-GPES Reordering*)**.** Let P be a reordering matrix. We call P a p-GPES reordering that respects the graph partitioning result $\mathcal{V} = \mathcal{V}_1 \cup \dots \cup \mathcal{V}_p$, if for any $k \neq l$ either $\pi_P(i) < \pi_P(j)$ for all $i \in \mathcal{V}_k$ and all $j \in \mathcal{V}_l$; or $\pi_P(i) > \pi_P(j)$ for all $i \in \mathcal{V}_k$ and all $j \in \mathcal{V}_l$. π_P is the permutation represented by the reordering matrix P.

In Fig. 3(a) we show a graph with 64 nodes where the nodes are column-major ordered. In Fig. 3(b), the graph is partitioned into 4 parts $\mathcal{V}_1$, $\mathcal{V}_2$, $\mathcal{V}_3$, and $\mathcal{V}_4$. The nodes are then numbered according to a p-GPES reordering, where all the nodes in the same subset $\mathcal{V}_k$ are numbered in a contiguous way.

3.2. Vertex cover based overlapping

Suppose the graph partitioning step applied to the coefficient matrix $\bar{A}_s$ produces a partition $\mathcal{V} = \cup_{k=1}^{p} \mathcal{V}_k$ and an induced p-GPES reordering P_1. Then after reordering $\bar{A}_s$ with P_1, the p non-overlapping blocks on the diagonal would be $C_k = \bar{A}_s(\mathcal{V}_k, \mathcal{V}_k)$. In order to get a submatrix M in ODB form, we consider the problem of adding overlaps to the non-overlapping diagonal block form (nODB) composed of $C_1, \dots, C_p$. An example with $p = 4$ is shown in Fig. 4. In the following, we let A_k denote the overlapping diagonal block resulted from adding overlaps to the non-overlapping diagonal block C_k for $k = 1, \dots, p$. And the size of overlap between A_k and A_{k+1} is denoted by τ_k for $k = 1, \dots, p-1$. It is worth mentioning that since the p-GPES step already takes care of the load balancing issue among the blocks $C_1, \dots, C_p$ approximately as discussed in Section 3.1, adding overlaps with sizes $\tau_k \ll n$ $(k = 1, \dots, p-1)$ will not lead to load imbalance. Thus, the load balance among $A_1, \dots, A_p$ is derived from the load balance among $C_1, \dots, C_p$, no matter how the overlaps are determined.

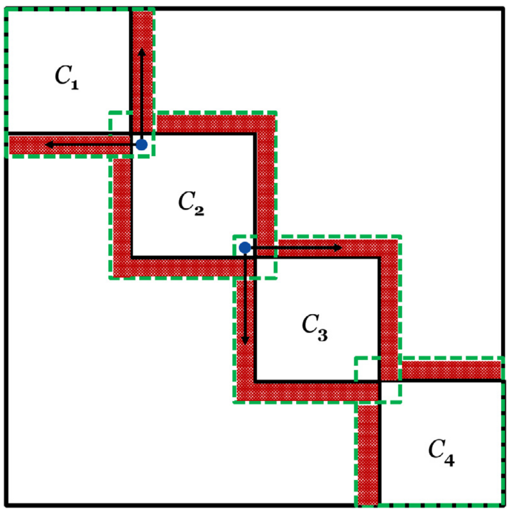


Fig. 4. Adding overlaps to a non-overlapping diagonal block form, with $p = 4$. We want to maximize the total magnitude of elements in the extended area that is shaded in red. A node $i \in \mathcal{V}_2$ that is in the vertex cover of the cut edges is represented by a blue dot. In the G-PAVER reordering, whether i is permuted to the top-left corner or bottom-right corner of C_2 is determined by comparing the total magnitude of edges connecting i to $\mathcal{V}_1$ versus that connecting i to $\mathcal{V}_3$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

When adding overlaps to a nODB form, we would like to achieve the following two goals simultaneously

- Minimizing the total size of required overlaps $\sum_{k=1}^{p-1} \tau_k$;
- Maximizing the total magnitude of elements in the *extended area*. In Fig. 4, the extended area is shaded in red.

We approach this problem using the idea of *vertex cover based overlapping*. In a nODB form induced by a p-GPES, the off-diagonal elements correspond to cut edges. In Fig. 3(b), the graph is partitioned into 4 parts $\mathcal{V}_1$, $\mathcal{V}_2$, $\mathcal{V}_3$ and $\mathcal{V}_4$. All the cut edges are colored in green. The nodes colored in red form the vertex cover of the cut edges. Now consider adding overlaps to C_1 and C_2 such that A_1 and A_2 overlap on nodes $\{16, 17\}$, then all the cut edges between $\mathcal{V}_1$ and $\mathcal{V}_2$ will be covered by the ODB form. Similarly, if we let A_2 and A_3 overlap on $\{32, 33\}$, and A_3 and A_4 overlap on $\{48, 49\}$, then all the cut edges except those between $\mathcal{V}_1$ and $\mathcal{V}_4$ will be covered by the ODB form.

Let the set of cut edges of a p-GPES outcome $\mathcal{V} = \cup_{k=1}^{p} \mathcal{V}_k$ be

$$\mathcal{E}_{\text{cut}} = \left\{ (i,j) \in \mathcal{E} : i \in \mathcal{V}_k,\ j \in \mathcal{V}_l,\ k \neq l \right\}. \tag{19}$$

Ideally, we would like to find a minimum vertex cover of $\mathcal{E}_{\text{cut}}$. Although the minimum vertex cover problem is NP-hard, there are efficient heuristic algorithms that work very well in practice and with guaranteed approximation ratios [22–24]. In our implementation of G-PAVER, we adopt the algorithm [23] because of its superior performance [25].

Notice that the sequence $C_1, \ldots, C_p$ is effectively a linear ordering of the non-overlapping diagonal blocks. The cut edges between $\mathcal{V}_k$ and $\mathcal{V}_l$ will never be included in the added overlaps unless $l = k \pm 1$ because of the restriction of this linear ordering.

3.2.1. *Spectral reordering of non-overlapping diagonal blocks*

We want to optimize a reordering $C_{\pi(1)}, \ldots, C_{\pi(p)}$ so that the total magnitude of off-diagonal elements between adjacent non-overlapping diagonal blocks C_{k-1} and C_k for $k = 1, \ldots, p-1$ is maximized. We approximately solve this problem using spectral reordering on the *quotient graph*, which is defined as follows.

Definition 3.3 (*Quotient Graph*). Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected and weighted graph, and $\mathcal{V} = \mathcal{V}_1 \cup \cdots \cup \mathcal{V}_p$ be an outcome of a p-GPES. The quotient graph $\widetilde{\mathcal{G}} = (\widetilde{\mathcal{V}}, \widetilde{\mathcal{E}})$ with the weight function $\widetilde{w}(\cdot, \cdot)$ for $k \neq l$ is defined as

$$\begin{aligned}
\widetilde{\mathcal{V}} &= \left\{ \mathcal{V}_1, \ldots, \mathcal{V}_p \right\} \\
\widetilde{w}(\mathcal{V}_k, \mathcal{V}_l) &= \begin{cases} \sum_{i \in \mathcal{V}_k} \sum_{j \in \mathcal{V}_l} w(i,j) & \text{if } k \neq l \\ 0 & \text{otherwise} \end{cases} \\
\widetilde{\mathcal{E}} &= \left\{ (\mathcal{V}_k, \mathcal{V}_l) : \widetilde{w}(\mathcal{V}_k, \mathcal{V}_l) > 0 \right\}.
\end{aligned} \tag{20}$$

We then use the Fiedler vector on the quotient graph $\widetilde{\mathcal{G}}$ to reorder the non-overlapping diagonal blocks as $C_{\pi_{\widetilde{P}}(1)}, \ldots, C_{\pi_{\widetilde{P}}(p)}$, where $\widetilde{P}$ is the corresponding block permutation matrix. Note that p is usually a small number, thus the Fiedler vector on $\widetilde{\mathcal{G}}$ can be computed cheaply using LAPACK subroutines, e.g., `dsyevr`. In Fig. 3(b), it is obvious that $\mathcal{V}_1$, $\mathcal{V}_2$, $\mathcal{V}_3$, $\mathcal{V}_4$ already follow a spectral reordering on the quotient graph because the number of cut edges between $\mathcal{V}_1$ and $\mathcal{V}_4$ is the smallest.

3.2.2. *Permuting vertex cover nodes*

In the following we assume the sequence of non-overlapping diagonal blocks $C_1, \ldots, C_p$ has already been subjected to spectral reordering. Let an approximate minimum vertex cover of $\mathcal{E}_{\text{cut}}$ be $\mathcal{VC} = \mathcal{VC}^1 \cup \cdots \cup \mathcal{VC}^p$, where $\mathcal{VC}^k$ is the vertex cover nodes in $\mathcal{V}_k$. In Fig. 4, we represent a node $i \in \mathcal{VC}^2$ by a blue dot. If we permute i to the top-left corner of C_2, then the added overlap between A_1 and A_2 will include those edges connecting i to the nodes in $\mathcal{V}_1$ (as indicated by the arrows crossing the red shaded area). Similarly, if we permute i to the bottom-right corner of C_2, then the added overlap between A_2 and A_3 will include those edges connecting i to the nodes in $\mathcal{V}_3$. We choose the permutation of i with a larger total magnitude of edges included via the added overlap. Generally for $i \in \mathcal{VC}^k$, we define the following two scores (intuitively the *left score* and the *right score*)

$$s^k_{k-1}(i) = \begin{cases} \sum_{j \in \mathcal{V}_{k-1}} (|\bar{A}_s(i,j)| + |\bar{A}_s(j,i)|)/2 & \text{if } k > 1 \\ 0 & \text{o.w.} \end{cases}$$
$$s^k_{k+1}(i) = \begin{cases} \sum_{j \in \mathcal{V}_{k+1}} (|\bar{A}_s(i,j)| + |\bar{A}_s(j,i)|)/2 & \text{if } k < p \\ 0 & \text{o.w.} \end{cases} \tag{21}$$

Then i is permuted to the top-left corner of C_k if $s^k_{k-1}(i) > s^k_{k+1}(i)$. Otherwise, i is permuted to the bottom-right corner of C_k. Using this rule, we can partition each $\mathcal{VC}^k$ into two subsets $\mathcal{VC}^k = \mathcal{VC}^k_{k-1} \cup \mathcal{VC}^k_{k+1}$, where

$$\mathcal{VC}^k_{k-1} = \left\{ i \in \mathcal{VC}^k : s^k_{k-1}(i) > s^k_{k+1}(i) \right\}$$
$$\mathcal{VC}^k_{k+1} = \left\{ i \in \mathcal{VC}^k : s^k_{k-1}(i) \le s^k_{k+1}(i) \right\}. \tag{22}$$

This in turn partitions $\mathcal{V}_k$ into three subsets

$$\mathcal{V}_k = \mathcal{VC}^k_{k-1} \cup \left(\mathcal{V}_k \setminus \mathcal{VC}^k \right) \cup \mathcal{VC}^k_{k+1}. \tag{23}$$

Definition 3.4 (*Local Vertex Cover Reordering*). Let $\mathcal{V}_k$ be partitioned as $\mathcal{V}_k = \mathcal{VC}^k_{k-1} \cup \left(\mathcal{V}_k \setminus \mathcal{VC}^k \right) \cup \mathcal{VC}^k_{k+1}$ where $\mathcal{VC}^k$ is the vertex cover nodes in $\mathcal{V}_k$, with $\mathcal{VC}^k_{k-1}$ and $\mathcal{VC}^k_{k+1}$ as defined in (22). A reordering P^k on $\mathcal{V}_k$ is called a local vertex cover reordering if the following conditions are satisfied by the permutation $\pi_{P^k}(\cdot)$:

(i) for $i, j \in \mathcal{VC}^k_{k-1}$, $\pi_{P^k}(i) < \pi_{P^k}(j)$ only if $s^k_{k-1}(i) \ge s^k_{k-1}(j)$;
(ii) for all $i \in \mathcal{VC}^k_{k-1}$ and all $j \in \mathcal{V}_k \setminus \mathcal{VC}^k$, $\pi_{P^k}(i) < \pi_{P^k}(j)$;
(iii) for all $i \in \mathcal{VC}^k_{k+1}$ and all $j \in \mathcal{V}_k \setminus \mathcal{VC}^k$, $\pi_{P^k}(i) > \pi_{P^k}(j)$;
(iv) for $i, j \in \mathcal{VC}^k_{k+1}$, $\pi_{P^k}(i) > \pi_{P^k}(j)$ only if $s^k_{k+1}(i) \ge s^k_{k+1}(j)$.

Conditions (ii) and (iii) guarantee that nodes in $\mathcal{VC}^k_{k-1}$ ($\mathcal{VC}^k_{k+1}$) will be permuted to the top-left (bottom-right) corner of C_k; while conditions (i) and (iv) further sort the nodes in $\mathcal{VC}^k_{k-1}$ ($\mathcal{VC}^k_{k+1}$) such that more important nodes are closer to the block C_{k-1} (C_{k+1}). From the local vertex cover reorderings P^k for $k = 1, \ldots, p$, we define the global vertex cover reordering

$$P_2 = \text{diag}\left(P^1, \ldots, P^p \right). \tag{24}$$

We summarize the algorithm of G-PAVER reordering in Algorithm 1. In Algorithm 1, the G-PAVER reordering P is defined by the concatenation

$$P = P_2 \widetilde{P} P_1. \tag{25}$$

When we apply G-PAVER to reorder the coefficient matrix $\bar{A}_s$ in order to extract a preconditioner in ODB form, we need to control the total overlap size $\sum_{k=1}^{p-1} \tau_k$ which is the dimension of the balance system (9). Thus, in Algorithm 1 we have an extra parameter τ that is used to control the overlap sizes such that $\tau_k \le \tau$ for $k = 1, \ldots, p-1$. The **for-loop** in lines 14–24 of Algorithm 1 determines the number of vertex cover nodes to include when adding overlaps to C_k and C_{k+1}. When the number of vertex cover nodes connecting C_k and C_{k+1} are no larger than τ, i.e., $|\mathcal{VC}^k_{k+1}| + |\mathcal{VC}^{k+1}_k| \le \tau$, then all these vertex cover nodes can be included by the overlaps added to C_k and C_{k+1}. This feature is especially useful when the cut edges $\mathcal{E}_{\text{cut}}$ have a very small vertex cover. In this case, the effective total overlap size $\sum_{k=1}^{p-1} \tau_k \ll (p-1)\tau$ even though the user might have specified a large τ.

The idea of adding overlaps to a graph partitioning result is also considered in [26]. Instead of directly adding overlaps to a graph partitioning, the contribution of G-PAVER is that it locally permutes $\mathcal{V}_k$ such that the vertex cover nodes in $\mathcal{V}_k$ are closer to the neighboring $\mathcal{V}_{k-1}$ or $\mathcal{V}_{k+1}$. In fact, the vertex cover nodes belong to the so-called *first level set* of $\mathcal{V}_k$ that is used for adding overlaps in [26]. By only adding the vertex cover nodes, G-PAVER reduces the required sizes of overlaps while still making it possible to cover as many cut edges as possible. In a recent work [15], the authors aim at permuting the entire matrix $\bar{A}_s$ to ODB form. However, even for moderate values of p, permuting the entire $\bar{A}_s$ into ODB form often leads to very large total overlap size $\sum_{k=1}^{p-1} \tau_k$ [15],[1] which makes the balance system (9) hard to solve. Since our target is to extract a preconditioner in ODB, we impose a threshold τ on the overlap sizes, while using G-PAVER reordering to enhance the quality of the ODB preconditioner.

Algorithm 1 G-PAVER $(\bar{A}_s, p, \tau)$

```
Input: $\bar{A}_s \in \mathbb{R}^{n\times n}$; the number of blocks $p$; the threshold on overlap sizes $\tau$.
Output: The G-PAVER reordering $P$; the sizes of the added overlap to the left $\mu_k^{k-1}$ and to the right $\mu_k^{k+1}$ such that
    $\tau_k = \mu_k^{k+1} + \mu_{k+1}^k \le \tau$ for $k = 1, \dots, p-1$.
 1: Construct the graph $\mathcal{G}[\bar{A}_s]$ whose adjacency matrix is $(|\bar{A}_s| + |\bar{A}_s^{\mathrm{T}}|)/2$.
 2: Apply $p$-GPES to $\mathcal{G}[\bar{A}_s]$ and let the result be $\mathcal{V} = \mathcal{V}_1 \cup \cdots \cup \mathcal{V}_p$.
    Let an induced $p$-GPES reordering be $P_1$.
 3: Construct the quotient graph $\widetilde{\mathcal{G}[\bar{A}_s]}$ of the $p$-GPES result according to (20). Compute the Fiedler vector on $\widetilde{\mathcal{G}[\bar{A}_s]}$ and then
    sort it to get the block spectral reordering $\widetilde{P}$.
 4: Compute an approximate minimum vertex cover of the cut edges $\mathcal{E}_{\text{cut}}$ defined in (19). Denote the approximate minimum
    vertex cover by $\mathcal{VC} = \mathcal{VC}^1 \cup \cdots \cup \mathcal{VC}^p$, where $\mathcal{VC}^k$ is the vertex cover nodes in $\mathcal{V}_k$.
 5: for $k = 1, \dots, p$ do
 6:     for $i \in \mathcal{VC}^k$ do
 7:         Compute the left score $s_{k-1}^k(i)$ and right score $s_{k+1}^k(i)$ according to (21) with respect to the block spectral reordering
            $\widetilde{P}$.
 8:     end for
 9:     Partition $\mathcal{VC}^k$ to $\mathcal{VC}^k_{k-1}$ and $\mathcal{VC}^k_{k+1}$ according to (22). Note that by definition $\mathcal{VC}^1_0 = \emptyset$, and $\mathcal{VC}^p_{p+1} = \emptyset$.
10:     Derive a local vertex cover reordering $P^k$ on $\mathcal{V}_k$ according to the partition (23) as well as by sorting $\mathcal{VC}^k_{k-1}$ ($\mathcal{VC}^k_{k+1}$) in
        descending (ascending) order according to $s^k_{k-1}$ ($s^k_{k+1}$).
11: end for
12: Compute the global vertex cover reordering $P_2$ according to (24).
13: Compute the G-PAVER reordering $P = P_2\widetilde{P}P_1$.
14: for $k = 1, \dots, p-1$ do
15:     if $|\mathcal{VC}^k_{k+1}| + |\mathcal{VC}^{k+1}_k| \le \tau$ then
16:         $\mu_k^{k+1} = |\mathcal{VC}^{k+1}_k|$, and $\mu_{k+1}^k = |\mathcal{VC}^k_{k+1}|$.
17:     else if $|\mathcal{VC}^k_{k+1}| \le \tau$ then
18:         $\mu_{k+1}^k = |\mathcal{VC}^k_{k+1}|$, and $\mu_k^{k+1} = \tau - \mu_{k+1}^k$.
19:     else if $|\mathcal{VC}^{k+1}_k| \le \tau$ then
20:         $\mu_k^{k+1} = |\mathcal{VC}^{k+1}_k|$, and $\mu_{k+1}^k = \tau - \mu_k^{k+1}$.
21:     else
22:         $\mu_k^{k+1} = \lceil \tau/2 \rceil$, and $\mu_{k+1}^k = \lfloor \tau/2 \rfloor$.
23:     end if
24: end for
25: Return the G-PAVER reordering $P$; the sizes of added overlaps $\mu_k^{k-1}, \mu_k^{k+1}$, for $k = 1, \dots, p-1$.
```

4. Solve the explicit balance system

In this section, we describe our solution scheme of the balance system (9). We explicitly form the coefficient matrix B in (14) by solving the linear systems (15). Given the block tridiagonal form of B in (14), we consider its block LU factorization,

[1] Although the reported ratios $\sum_{k=1}^{p-1} \tau_k/n$ are not large for most cases, $\sum_{k=1}^{p-1} \tau_k$ could be unacceptably large for large n.

i.e.,

$$B = \begin{pmatrix} F_1 & & & & \\ G_2 & F_2 & & & \\ & & \ddots & & \\ & & G_{p-2} & F_{p-2} & \\ & & & G_{p-1} & F_{p-1} \end{pmatrix} \begin{pmatrix} I & H_2 & & & \\ & I & H_3 & & \\ & & \ddots & & \\ & & & I & H_{p-1} \\ & & & & I \end{pmatrix} \tag{26}$$

where

$$F_1 = B_{33}^{(1)} + B_{11}^{(2)}$$

$$k = 2, \ldots, (p-1) \begin{cases} G_k = -B_{31}^{(k)} \\ F_k H_{k+1} = -B_{13}^{(k+1)} \\ F_k + G_k H_k = B_{33}^{(k)} + B_{11}^{(k+1)}. \end{cases} \tag{27}$$

In order to solve the linear systems in (27), we factorize the blocks F_k for $k = 1, \ldots, p$ with partial pivoting. It is still possible, however, to encounter pivots close to zero. We overcome this problem with a "diagonal boosting" strategy as follows. Let *pivot* be the current pivot. If $|pivot| < \epsilon$, then its magnitude is boosted by

$$pivot = pivot + \delta \quad \text{if } pivot \geq 0$$
$$pivot = pivot - \delta \quad \text{if } pivot < 0$$

where ϵ is a "machine zero" parameter, and δ is the boosting parameter. We implement this "diagonal boosting" strategy by modifying the LAPACK primitives DGETRF and DGETF2. Because of the diagonal boosting, we effectively compute an approximate factorization of F_k with low rank perturbation, i.e.,

$$\widetilde{P}_k F_k \approx \widetilde{L}_k \widetilde{U}_k.$$

Thus, we obtain the following approximate block LU-factorization of B

$$\widetilde{P}_B B \approx \begin{pmatrix} \widetilde{L}_1\widetilde{U}_1 & & & & \\ G_2 & \widetilde{L}_2\widetilde{U}_2 & & & \\ & & \ddots & & \\ & & G_{p-2} & \widetilde{L}_{p-2}\widetilde{U}_{p-2} & \\ & & & G_{p-1} & \widetilde{L}_{p-1}\widetilde{U}_{p-1} \end{pmatrix} \begin{pmatrix} I & H_2 & & & \\ & I & H_3 & & \\ & & \ddots & & \\ & & & I & H_{p-1} \\ & & & & I \end{pmatrix}$$
$$= \widetilde{L}_B \widetilde{U}_B. \tag{28}$$

We then solve the balance system (9) using an LU-preconditioned Krylov subspace method (e.g., BiCGstab). In the following, this Krylov subspace method for solving the balance system (9) is referred to as the *inner* Krylov subspace method. Because the perturbation $(\widetilde{P}_B B - \widetilde{L}_B \widetilde{U}_B)$ is of low rank, the preconditioned inner Krylov subspace method will converge rather quickly. It is also worth pointing out that because B is explicitly available, its matrix–vector multiplication can be computed using its block tridiagonal form (14), instead of using the *implicit* matrix–vector multiplication described in [7,8].

5. The parallel implementation of PSPIKE+

In this section, we describe the parallelism in the PSPIKE+ algorithm, and its implementation on parallel architectures.

The first step of the G-PAVER reordering is graph partitioning. For this purpose, we use the state-of-the-art parallel graph partitioning tool ParMETIS[2] [27]. We acknowledge that in our current implementation of G-PAVER, the step of vertex cover based overlapping is serial. However, this is not a bottleneck because the time of the G-PAVER reordering is usually a tiny percentage of the total time needed by PSPIKE+ for solving large sparse linear systems.

5.1. Parallel sparse direct solvers

After tearing M into p overlapping diagonal blocks $A_1, \ldots, A_p$, they can be factored in parallel. Furthermore, all the linear systems (15), (16), and (18) can be solved simultaneously in parallel for $k = 1, \ldots, p$. In our PSPIKE+ implementation, we use the state-of-the-art parallel sparse direct solver PARDISO 5.0.0[3] [18]. Specifically, we assign each block A_k to a single node of a cluster, and then use PARDISO 5.0.0 in each node for parallel factorization and triangular solve of A_k through multithreading that exploits the multi-core architecture of a single node. Two other outstanding characteristics of PARDISO 5.0.0 that makes it a preferred sparse direct solver for PSPIKE+ are:

2 http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview.

3 http://www.pardiso-project.org/.

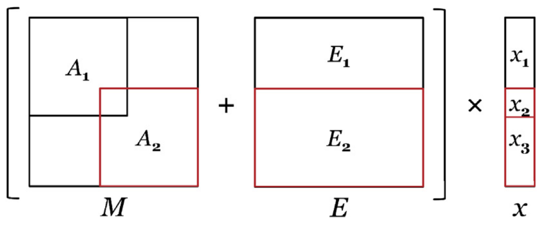


Fig. 5. Partitions of M, E, and x for parallel matrix–vector multiplication in the outer Krylov subspace method.

(1) Pivot perturbation strategy. After the tearing, a certain block A_k could be close to a singular matrix. When factoring A_k using PARDISO 5.0.0, tiny pivots will be perturbed to ensure obtaining nonsingular factors of a slightly perturbed A_k. This makes it more robust compared to using some other sparse direct solvers. According to Theorem 2.1, if the factorization of all A_k for $k = 1, \ldots, p$ is successful, then the balance system (9) is well defined. Effectively, the pivot perturbation strategy changes the ODB preconditioner from M to $\widetilde{M}$. From the point of view of preconditioning the linear system (5), if the perturbation $\widetilde{M} - M$ is of low rank, which is usually the case, then $\widetilde{M}$ will also be an effective preconditioner.

(2) Extracting only the tips of the solution. When solving the linear systems (15) and (16) that define the balance system (9), essentially we only need the top and bottom tips of the solutions, i.e., $B_{11}^{(k)}$, $B_{31}^{(k)}$, $B_{13}^{(k)}$, $B_{33}^{(k)}$, $h_1^{(k)}$, and $h_3^{(k)}$. Because $\tau_{k-1}, \tau_k \ll n$, extracting only the top and bottom tips is much faster than complete triangular solves. This capability is also provided by PARDISO 5.0.0.

5.2. Parallel matrix–vector multiplication in the outer Krylov subspace method

According to the splitting of A in (7), we can compute the matrix–vector multiplication Ax in the outer Krylov subspace method as $Ax = Mx + Ex$. We describe how to take advantage of the ODB form of M to compute Mx in parallel. For simplicity, we consider the case of $p = 2$. But the same method generalizes to any value of p. In Fig. 5 we show the partitions of M, E, and x for parallel matrix–vector multiplication in the outer Krylov subspace method. Note that E is partitioned into non-overlapping block rows. A_1 and E_1 are stored on the first node, while A_2 and E_2 are stored on the second node. The vector x is partitioned into three blocks x_1, x_2 and x_3, with x_1 on the first node and x_2, x_3 on the second node. For each round of Mx, the second node first sends the block x_2 to the first node. Then the two nodes compute their local matrix–vector multiplications in parallel

$$\begin{pmatrix} v_1^{(1)} \\ v_2^{(1)} \end{pmatrix} = A_1 \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \qquad \begin{pmatrix} v_2^{(2)} \\ v_3^{(2)} \end{pmatrix} = A_2 \begin{pmatrix} x_2 \\ x_3 \end{pmatrix}.$$

After that, the first node sends $v_2^{(1)}$ to the second node and the resulting vector $v = Mx$ is then given by

$$v = \begin{pmatrix} v_1^{(1)} \\ v_2^{(1)} + v_2^{(2)} \\ v_3^{(2)} \end{pmatrix}.$$

For general values of p, the communications during the parallel computation of Mx only happen between a node and its immediate left and right neighbors, and the sizes of messages are $\tau_k \ll n$ for $k = 1, \ldots, p$.

The computation of Ex follows the usual parallel sparse matrix–vector multiplication using row-wise partitioning, where E_1x_1 is carried out on the first node, and $E_2 \begin{pmatrix} x_2 \\ x_3 \end{pmatrix}$ is carried out in the second node. Note that for a general value of p, the communication overhead of Ex is much less than that of Ax due to the fact that E is much sparser than A after the G-PAVER reordering.

5.3. Parallel solution of the explicit balance system

The block tridiagonal matrix B of the balance system (9) is generated and then stored in the first $(p-1)$ nodes, where the kth node stores the block row $\left(-B_{31}^{(k)}, B_{33}^{(k)} + B_{11}^{(k+1)}, -B_{13}^{(k+1)}\right)$. For the block LU factors, the kth node stores the blocks G_k, F_k, and H_{k+1}. During the parallel block LU factorization of B, the kth node needs to receive block H_k from its left neighbor, and then sends H_{k+1} to its right neighbor. The solution to the balance system (17) is also stored in the first $(p-1)$ nodes, where the kth node stores the subvector y_k. During the parallel matrix–vector multiplication of the inner Krylov subspace method, the kth node only needs to receive y_{k-1} (y_{k+1}) from its left (right) neighbor given the block tridiagonal form of B in (14). Thus, the parallel solution of the explicit balance system only involves nearest-neighbor communication with small-sized messages (because $\tau_{k-1}, \tau_k, \tau_{k+1} \ll n$), which is efficient on modern parallel architectures.

We summarize the PSPIKE+ algorithm in Algorithm 2.

Algorithm 2 PSPIKE+ ($\bar{A}, \bar{f}, p, \tau, \rho, maxit$)

```
Input: Nonsingular coefficient matrix $\bar{A} \in \mathbb{R}^{n\times n}$; the RHS $\bar{f}$; the number of blocks in the ODB form $p$; the threshold on
    overlap sizes $\tau$; the stopping criterion $\rho$; the maximum number of iterations in the outer Krylov subspace method $maxit$.
Output: Either an approximate solution $\bar{x}$ to the linear system (1), or report FAILURE.
 1: Apply MC64 column permutation $Q$ to $\bar{A}$ with optional row and column scaling $D_r$ and $D_c$. Let the transformed linear
    system be given by (2).
 2: Apply the G-PAVER symmetric reordering $P$ to $\bar{A}_s$ with the parameters $p$ and $\tau$. Let the transformed linear system be
    given by (5).
 3: Split $A$ into the ODB preconditioner $M$ and the matrix $E$ according to (7).
 4: Tear $M$ into a sequence of overlapping diagonal blocks $A_1, \dots, A_p$, where $A_k$ is stored on the $k$th node of a cluster. Each
    of the $p$ nodes factors $A_k$ using PARDISO 5.0.0 in parallel for $k = 1, \dots, p$.
 5: Explicitly form the coefficient matrix $B$ of the balance system (9) by extracting only the top and bottom tips of the
    solutions to the linear systems (15).
 6: Obtain the approximate block $LU$-factorization of $B$ according to (26).
 7: Solve the linear system (5) using parallel BiCGstab with the ODB preconditioner $M$, given the parameters $\rho$ and $maxit$.
    Either an approximate solution $x$ to (5) is found, or report FAILURE.
 8: if $x$ is found then
 9:     Compute the approximate solution $\bar{x}$ to (1) according to (6) and (4). Return $\bar{x}$.
10: else
11:     Report FAILURE.
12: end if
```

6. Numerical experiments

Our parallel implementation of PSPIKE+ is written in FORTRAN 90 and is based on the Intel MKL library. All the numerical experiments in this section are conducted on a distributed memory machine consisting of 8 nodes with each node having 24 cores (Intel Xeon E5-4617 processor) and 64 GB memory. For a given input coefficient matrix, we generate the RHS of the linear system by assuming the all-one vector as the solution. In all our experimental results, only in Section 6.1 we chose to activate the optional row and column scaling of MC64 in order to compare with the work [28] which assumes I-matrices as input.

6.1. G-PAVER reordering for block Jacobi preconditioners

In the G-PAVER reordering, if we set $\tau = 0$ (i.e., non-overlapping) then M becomes a block Jacobi preconditioner. In this section, we demonstrate that the p-GPES step of G-PAVER produces block Jacobi preconditioners that are robust and efficient. Unlike the study in [29] which only considers graph partitioning for preconditioning symmetric positive definite linear systems, we report experimental results of using G-PAVER without overlapping for preconditioning general sparse linear systems. Specifically, we compare G-PAVER to XPABLO [30] and SCPRE [28]. The matrices we use are those reported in Table 5.5 of [28]. Because [28] assumes I-matrices, we treat the matrix $\bar{A}_s$ in (3) after the MC64 column permutation with row and column scaling as the input coefficient matrix. In Table 1, for each linear system we report the number of outer BiCGstab iterations and the associated time to solve it using PSPIKE+ when G-PAVER is directed to produce a block Jacobi preconditioner. The numbers in boldface represent results that are superior to the best results reported in Table 5.5 of [28]. We see that G-PAVER can produce very competitive block Jacobi preconditioners as compared to XPABLO [30] and SCPRE [28]. In counting the total number of successes, PSPIKE+ with G-PAVER for block Jacobi preconditioners succeeds on 41 out of 49 linear systems (84%), while SCPRE-dec (the best in Table 5.5 of [28]) succeeds on 36 out of 49 (73%).

6.2. Effects of overlap sizes for preconditioning

In this section, we empirically investigate the effects of varying the overlap sizes of the ODB preconditioner M. We essentially achieve this by varying the value of τ, i.e., the threshold on the overlap sizes. Because M degenerates to a block Jacobi preconditioner when $\tau = 0$, the numerical experiments of this section will also shed light on the benefits of adding overlaps for preconditioning. The linear systems on which we have observed significant effects of overlap sizes are shown in Table 2 which are chosen from the University of Florida sparse matrix collection [11]. In Table 3 we demonstrate the effects of varying τ on the robustness and efficiency of the resulting ODB preconditioner. Especially, we have identified a subset of linear systems arising from Economic applications (mac_econ_fwd500, mark3jac120, mark3jac120sc, mark3jac140, mark3jac140sc) such that large enough overlap sizes (when $\tau = 400$) lead to the most robust and efficient ODB preconditioners, where the time of outer BiCGstab iterations is dramatically reduced compared to $\tau = 0, 100, 200$. As seen from Table 3 when $p = 4$, the block Jacobi preconditioner fails on all except one of these linear systems. When $p = 8$,

Table 1
Number of outer BiCGstab iterations and the associated time to solve it using PSPIKE+ when G-PAVER is directed to produce a block Jacobi preconditioner. When running PSPIKE+, we set $p = 8$ and $\tau = 0$ for block Jacobi preconditioners, the stopping criterion $\rho = 10^{-7}$ (in [28] the stopping criterion achievable on the unpreconditioned system is 10^{-7} although that on the preconditioned system from GMRES(50) is 10^{-8}), and the maximum number of BiCGstab iterations *maxit* = 500. The sign of '–' indicates a failure of PSPIKE+ to reach the stopping criterion.

Matrix	# BiCGstab iterations	BiCGstab time (s)
Hamrle2	–	–
rajat03	44.5	0.48
circuit_3	116.5	0.47
coupled	25	0.16
memplus	9.5	0.05
rajat22	240.5	0.80
onetone2	–	–
onetone1	–	–
rajat15	**138**	**0.56**
ckt11752_tr_0	59	**0.28**
circuit_4	–	–
bcircuit	171.5	**0.76**
rajat18	450	3.54
hcircuit	47.5	**0.35**
ASIC_100ks	**6.5**	**0.06**
ASIC_100k	**7**	**0.13**
ASIC_680ks	11.5	**0.54**
rajat23	**18.5**	**0.12**
twotone	–	–
trans5	57	0.58
dc2	**8**	**0.11**
G2_circuit	**68.5**	**0.68**
scircuit	**187**	**0.98**
transient	**173.5**	**1.48**
Raj1	**327.5**	**2.61**
ASIC_320ks	**0.5**	**0.04**
ASIC_320k	**2.5**	**0.16**
utm5940	**275.5**	**0.54**
dw4096	–	–
Zhao1	10.5	0.12
igbt3	139.5	0.45
wang3	**28**	**0.19**
wang4	**27**	**0.24**
ecl32	**68**	**0.78**
ibm_matrix_2	80	0.77
matrix-new_3	**0.5**	**0.09**
matrix_9	**49.5**	**1.24**
ASIC_680k	14	0.92
G3_circuit	**87.5**	**5.89**
rajat29	–	–
rajat30	24	**1.37**
Hamrle3	–	–
memchip	29.5	**2.86**
offshore	**6.5**	**0.37**
tmt_sym	**144**	**4.21**
t2em	**72.5**	**2.70**
tmt_unsym	**240.5**	**7.92**
para-4	**69.5**	**2.70**
ohne2	**1**	**0.12**

only $\tau = 200$ and $\tau = 400$ result in a successful PSPIKE+ on all of them. For the linear systems largebasis, scircuit, and TSOPF_RS_b39_c30, increasing τ makes no difference regarding the number of outer BiCGstab iterations. Recall that τ is just a threshold on the overlap sizes, it is possible that the actual sizes of overlaps as determined by G-PAVER on these matrices could be the same for different values of τ. This adaptivity of G-PAVER enables PSPIKE+ to control the overlap sizes without sacrificing the quality of the ODB preconditioner. We also see from Table 3 that when $p = 2$, all the linear systems enjoy improved efficiency from adding large enough overlaps. When $p = 4$ and $p = 8$, on TSOPF_RS_b39_c30, venkat25, and venkat50 although the number of outer BiCGstab iterations of the overlapping cases are still better than that of the block Jacobi preconditioner, the timing of the latter is more favorable because there is no need to solve the balance system when $\tau = 0$.

Table 2
Linear systems demonstrating the effects of overlap sizes for preconditioning.

Linear system	n	nnz	Application
largebasis	440,020	5,240,084	Optimization problem
mac_econ_fwd500	206,500	1,273,389	Economic problem
mark3jac120	54,929	322,483	
mark3jac120sc	54,929	322,483	
mark3jac140	64,089	376,395	
mark3jac140sc	64,089	376,395	
scircuit	170,998	958,936	Circuit simulation
TSOPF_RS_b39_c30	60,098	1,079,986	Power network problem
venkat25	62,424	1,717,763	Computational fluid dynamics
venkat50	62,424	1,717,777	

6.3. Robustness of PSPIKE+

We demonstrate the robustness of PSPIKE+ by comparing it with PARDISO 5.0.0. We chose from The University of Florida sparse matrix collection [11] 142 matrices of miscellaneous applications, as summarized in Table 4. We run PARDISO 5.0.0 under its default setting on one node with 16 threads. It succeeds on 120 out of 142 linear systems with an aggregated success rate of 85%. For PSPIKE+, we set the threshold on overlap sizes $\tau = 200$, the maximum number of outer BiCGstab iterations $maxit = 500$, and vary the number of overlapping diagonal blocks $p = 2$, 4, and 8. For both PARDISO 5.0.0 and PSPIKE+, we set the limit on the wall clock time for solving a linear system to be 600 s. In Table 5, we report the success rates of PSPIKE+ on the 142 linear systems for the stopping criterion $\rho = 10^{-4}$ and $\rho = 10^{-10}$ respectively.

Clearly, smaller value of the number of blocks p leads to more robust ODB preconditioner M. Specifically for $p = 2$, the aggregated success rate of PSPIKE+ is 73% if only $\rho = 10^{-4}$ is desired for the stopping criterion. Intuitively this is because when p gets larger, fewer elements of the coefficient matrix A are included in the preconditioner M. It is worth mentioning that among the 142 linear systems, we observe two linear systems on which PSPIKE+ achieves more accurate solutions (as measured by the relative residual) than the direct solver PARDISO 5.0.0, as shown in Table 6.

6.4. Parallel scalability of PSPIKE+

In this section, we demonstrate the parallel scalability of PSPIKE+. For each linear system, we profile the total time of the entire PSPIKE+ pipeline. The total time ($T_{\text{PSPIKE+}}$) consists of the times consumed by: MC64 column permutation (T_{MC64}), G-PAVER symmetric reordering ($T_{\text{G-PAVER}}$), constructing the ODB preconditioner (T_{ODB}), and the outer BiCGstab iterations (T_{BiCGstab}). The MC64 column permutation and the G-PAVER symmetric reordering (the vertex cover based overlapping to be more specific) are the main serial part of the entire PSPIKE+ pipeline. We observe that most often T_{MC64} and $T_{\text{G-PAVER}}$ are negligible compared to T_{ODB} and T_{BiCGstab}. Thus the serial nature of MC64 and G-PAVER is not an impediment to the parallel scalability of PSPIKE+. We also note that the time of constructing the ODB preconditioner T_{ODB} can be amortized when solving for multiple RHS with the same coefficient matrix. Nevertheless in the following we use $T_{\text{PSPIKE+}}$ to measure the parallel scalability of PSPIKE+ when solving for just one RHS.

6.4.1. Strong scaling on a model problem

We demonstrate the strong scaling of PSPIKE+ on a linear system whose coefficient matrix is defined by the 5-point discretization of Poisson equation on a 2D regular grid, with each dimension having 1000 nodes (thus $n = 1,000,000$ and $nnz = 4,996,000$). We fix the number of blocks $p = 192$ (same as the number of MPI processes, with each MPI process being single threaded), while changing the number of available cores to be 24, 48, 96, and 192. As shown in Fig. 6, PSPIKE+ achieves almost linear speedup on this model problem.

6.4.2. PSPIKE+ speed improvement over PARDISO 5.0.0

We demonstrate the speed improvement of PSPIKE+ over PARDISO 5.0.0 on the linear systems in Table 7 that are chosen from The University of Florida sparse matrix collection [11]. In Figs. 7–11 we plot the sparsity patterns of the coefficient matrices before and after the MC64 and G-PAVER reordering. When running PSPIKE+ for this experiment on our 8-node distributed memory machine, each node is assigned at most 1 MPI process, with 16 OpenMP threads as well as 16 MKL threads. We fix the stopping criterion of PSPIKE+ to $\rho = 10^{-4}$, and the maximum number of BiCGstab iterations to $maxit = 500$. Each linear system is also solved using PARDISO 5.0.0 with the default parameter setting on one node with 16 threads with the time consumed denoted by T_{PARDISO}. We find that on cage13, PARDISO 5.0.0 runs out of memory.

In Fig. 12(a)–(d) we plot the speed improvement of PSPIKE+ vs. PARDISO 5.0.0 ($T_{\text{PARDISO}}/T_{\text{PSPIKE+}}$) for different number of nodes (same as the number of blocks) $p = 2$, 4, and 8. From Fig. 12(b) we see that even though H2O is of moderate size, PSPIKE+ achieves superlinear speed improvement over PARDISO 5.0.0 due to the excessive fill-ins incurred by PARDISO 5.0.0 factorization of the whole matrix. As seen from Fig. 12(a) and (c), PSPIKE+ also achieves desirable speed improvement

Table 3
Effects of overlap sizes for preconditioning, with $p = 2$, 4, and 8 respectively. For each value of p, we vary the threshold on the overlap sizes $\tau = 0$, 100, 200, and 400. When running PSPIKE+, we set the stopping criterion $\rho = 10^{-10}$, and the maximum number of BiCGstab iterations *maxit* = 500. The sign of '–' indicates a failure of PSPIKE+ to reach the stopping criterion. For each linear system and each parameter setting, we report the number of outer BiCGstab iterations and the associated time to solve it.

p	Linear system		$\tau = 0$	$\tau = 100$	$\tau = 200$	$\tau = 400$
2	`largebasis`	# BiCGstab iterations	4	**0.5**	**0.5**	**0.5**
		BiCGstab time (s)	0.28	**0.12**	**0.12**	**0.14**
	`mac_econ_fwd500`	# BiCGstab iterations	254	60.5	34.5	**1**
		BiCGstab time (s)	21.36	17.02	6.83	**0.32**
	`mark3jac120`	# BiCGstab iterations	37.5	27.5	14.5	**1**
		BiCGstab time (s)	1.05	2.48	1.01	**0.08**
	`mark3jac120sc`	# BiCGstab iterations	46.5	27	11	**1**
		BiCGstab time (s)	1.75	2.20	0.77	**0.08**
	`mark3jac140`	# BiCGstab iterations	35.5	30.5	13.5	**1**
		BiCGstab time (s)	1.35	2.91	1.14	**0.09**
	`mark3jac140sc`	# BiCGstab iterations	60.5	27.5	14.5	**1**
		BiCGstab time (s)	1.65	2.27	1.22	**0.10**
	`scircuit`	# BiCGstab iterations	25	**0.5**	**0.5**	**0.5**
		BiCGstab time (s)	0.72	**0.03**	**0.04**	**0.03**
	`TSOPF_RS_b39_c30`	# BiCGstab iterations	15	**0.5**	**0.5**	**0.5**
		BiCGstab time (s)	0.17	**0.04**	**0.07**	**0.02**
	`venkat25`	# BiCGstab iterations	28.5	22	16	**0.5**
		BiCGstab time (s)	0.50	1.41	0.97	**0.04**
	`venkat50`	# BiCGstab iterations	34.5	27.5	18.5	**0.5**
		BiCGstab time (s)	0.80	1.74	0.94	**0.04**
4	`largebasis`	# BiCGstab iterations	12.5	**0.5**	**0.5**	**0.5**
		BiCGstab time (s)	0.62	**0.06**	**0.09**	**0.07**
	`mac_econ_fwd500`	# BiCGstab iterations	–	257.5	55.5	**1**
		BiCGstab time (s)	–	45.77	10.96	**0.17**
	`mark3jac120`	# BiCGstab iterations	–	72.5	40.5	**1**
		BiCGstab time (s)	–	3.23	4.04	**0.07**
	`mark3jac120sc`	# BiCGstab iterations	–	43.5	25.5	**1**
		BiCGstab time (s)	–	1.92	1.08	**0.08**
	`mark3jac140`	# BiCGstab iterations	–	51	28	**1**
		BiCGstab time (s)	–	2.57	1.32	**0.07**
	`mark3jac140sc`	# BiCGstab iterations	311.5	47	30.5	**1**
		BiCGstab time (s)	9.24	2.26	1.53	**0.08**
	`scircuit`	# BiCGstab iterations	395.5	**21.5**	**21.5**	**21.5**
		BiCGstab time (s)	4.36	**0.70**	**0.53**	**0.71**
	`TSOPF_RS_b39_c30`	# BiCGstab iterations	12.5	**5.5**	**5.5**	**5.5**
		BiCGstab time (s)	**0.12**	**0.15**	**0.15**	**0.15**
	`venkat25`	# BiCGstab iterations	36.5	29.5	**24**	**23.5**
		BiCGstab time (s)	**0.47**	0.97	0.95	0.84
	`venkat50`	# BiCGstab iterations	49.5	37	31	**27**
		BiCGstab time (s)	**0.69**	1.25	1.00	1.04
8	`largebasis`	# BiCGstab iterations	114.5	**0.5**	**0.5**	**0.5**
		BiCGstab time (s)	1.42	**0.06**	**0.06**	**0.07**
	`mac_econ_fwd500`	# BiCGstab iterations	–	–	382	**1**
		BiCGstab time (s)	–	–	26.54	**0.14**
	`mark3jac120`	# BiCGstab iterations	–	85.5	5	**1**
		BiCGstab time (s)	–	1.22	1.02	**0.10**
	`mark3jac120sc`	# BiCGstab iterations	–	–	37.5	**1**
		BiCGstab time (s)	–	–	0.62	**0.17**
	`mark3jac140`	# BiCGstab iterations	–	125.5	99.5	**1**
		BiCGstab time (s)	–	2.34	1.77	**0.12**
	`mark3jac140sc`	# BiCGstab iterations	–	124.5	87.5	**1**
		BiCGstab time (s)	–	1.76	1.89	**0.11**
	`scircuit`	# BiCGstab iterations	–	–	–	–

(continued on next page)

Table 3 (*continued*)

p	Linear system		$\tau = 0$	$\tau = 100$	$\tau = 200$	$\tau = 400$
		BiCGstab time (s)	–	–	–	–
	`TSOPF_RS_b39_c30`	# BiCGstab iterations	**23.5**	**24**	**24**	**24**
		BiCGstab time (s)	**0.12**	0.50	0.29	0.28
	`venkat25`	# BiCGstab iterations	46	**32.5**	**32**	**32**
		BiCGstab time (s)	**0.32**	0.62	0.69	0.65
	`venkat50`	# BiCGstab iterations	52.5	**44.5**	**43**	**45.5**
		BiCGstab time (s)	**0.33**	0.74	0.76	0.80

Table 4
Summary of UFL sparse matrices by applications used for robustness test of PSPIKE+.

Application	# matrices	n	nnz
Computational fluid dynamics	19	60,740–2,017,169	329,762–283,073,458
Chemical processes	9	57,735–112,211	275,094–1,915,726
Circuit simulation	28	68,902–4,690,002	307,604–20,316,253
Density functional theory	13	61,349–268,096	2,216,736–18,488,476
Directed weighted graph	1	399,130	1,216,334
DNA electrophoresis	4	130,228–5,154,859	2,032,536–99,199,551
Economic problem	9	53,370–206,500	322,483–12,733,89
Electromagnetics	6	101,492–1,157,456	1,647,264–89,306,020
Frequency domain circuit simulation	4	36,057–659,033	222,596–5,834,044
Model reduction	15	66,917–986,703	553,921–47,851,783
Optimization problem	3	160,000–440,020	1,750,416–5,240,084
Power network problem	7	60,098–226,340	762,969–13,135,930
Semicon. Dev. Sim.	18	103,430–181,343	893,984–6,869,939
Structural problem	3	68,121–1,504,002	5,377,761–110,686,677
Thermal problem	3	84,617–204,316	463,625–3,489,300

Table 5
Success rates of PSPIKE+ per each application as well as aggregated over the total 142 linear systems for the stopping criterion $\rho = 10^{-4}$ and $\rho = 10^{-10}$ respectively, and with respect to $p = 2$, 4, and 8. Throughout the experiments we fix $\tau = 200$, and the maximum number of outer BiCGstab iterations $maxit = 500$. The aggregated success rate of PARDISO 5.0.0 is $120/142 = 85\%$.

ρ	Application	$p = 2$	$p = 4$	$p = 8$
10^{-4}	Computational fluid dynamics	$12/19 = 63\%$	$11/19 = 58\%$	$11/19 = 58\%$
	Chemical processes	$8/9 = 89\%$	$8/9 = 89\%$	$8/9 = 89\%$
	Circuit simulation	$26/28 = 93\%$	$23/28 = 82\%$	$21/28 = 75\%$
	Density functional theory	$4/13 = 31\%$	$6/13 = 46\%$	$5/13 = 38\%$
	Directed weighted graph	$1/1 = 100\%$	$1/1 = 100\%$	$1/1 = 100\%$
	DNA electrophoresis	$2/4 = 50\%$	$2/4 = 50\%$	$3/4 = 75\%$
	Economic problem	$5/9 = 56\%$	$5/9 = 56\%$	$5/9 = 56\%$
	Electromagnetics	$2/6 = 33\%$	$2/6 = 33\%$	$2/6 = 33\%$
	Frequency domain circuit simulation	$3/4 = 75\%$	$2/4 = 50\%$	$0/4 = 0\%$
	Model reduction	$10/15 = 67\%$	$9/15 = 60\%$	$10/15 = 67\%$
	Optimization problem	$3/3 = 100\%$	$3/3 = 100\%$	$3/3 = 100\%$
	Power network problem	$5/7 = 71\%$	$4/7 = 57\%$	$4/7 = 57\%$
	Semicon. Dev. Sim.	$18/18 = 100\%$	$13/18 = 72\%$	$11/18 = 61\%$
	Structural problem	$2/3 = 67\%$	$2/3 = 67\%$	$1/3 = 33\%$
	Thermal problem	$3/3 = 100\%$	$3/3 = 100\%$	$3/3 = 100\%$
	Aggregated	$104/142 = 73\%$	$94/142 = 66\%$	$88/142 = 62\%$
10^{-10}	Computational fluid dynamics	$12/19 = 63\%$	$10/19 = 53\%$	$11/19 = 58\%$
	Chemical processes	$7/9 = 78\%$	$8/9 = 89\%$	$8/9 = 89\%$
	Circuit simulation	$22/28 = 79\%$	$20/28 = 71\%$	$15/28 = 54\%$
	Density functional theory	$3/13 = 23\%$	$4/13 = 31\%$	$4/13 = 31\%$
	Directed weighted graph	$1/1 = 100\%$	$1/1 = 100\%$	$1/1 = 100\%$
	DNA electrophoresis	$2/4 = 50\%$	$2/4 = 50\%$	$3/4 = 75\%$
	Economic problem	$5/9 = 56\%$	$5/9 = 56\%$	$5/9 = 56\%$
	Electromagnetics	$2/6 = 33\%$	$2/6 = 33\%$	$2/6 = 33\%$
	Frequency domain circuit simulation	$2/4 = 50\%$	$1/4 = 25\%$	$0/4 = 0\%$
	Model reduction	$8/15 = 53\%$	$7/15 = 47\%$	$7/15 = 47\%$
	Optimization problem	$3/3 = 100\%$	$3/3 = 100\%$	$3/3 = 100\%$
	Power network problem	$5/7 = 71\%$	$4/7 = 57\%$	$4/7 = 57\%$
	Semicon. Dev. Sim.	$11/18 = 61\%$	$6/18 = 33\%$	$6/18 = 33\%$
	Structural problem	$1/3 = 33\%$	$1/3 = 33\%$	$1/3 = 33\%$
	Thermal problem	$3/3 = 100\%$	$3/3 = 100\%$	$3/3 = 100\%$
	Aggregated	$87/142 = 61\%$	$77/142 = 54\%$	$73/142 = 51\%$

Table 6
Two linear systems on which PSPIKE+ achieves better accuracy than PARDISO 5.0.0, as measured by the relative residual.

Linear system	PARDISO 5.0.0	PSPIKE+		
		$p = 2$	$p = 4$	$p = 8$
`lhr71c`	5×10^{-6}	8×10^{-11}	8×10^{-12}	4×10^{-11}
`mac_econ_fwd500`	1×10^{-6}	7×10^{-11}	5×10^{-11}	1×10^{-10}

Table 7
Linear systems for demonstrating PSPIKE+ speed improvement over PARDISO 5.0.0.

Linear system	*n*	*nnz*	Application
`mac_econ_fwd500`	206,500	1,273,389	Economic problem
`H2O`	67,024	2,216,736	Density functional theory
`cage13`	445,315	7,479,343	Circuit simulation
`atmosmodl`	1,489,752	10,319,760	Computational fluid dynamics
`rajat31`	4,690,002	20,316,253	Circuit simulation

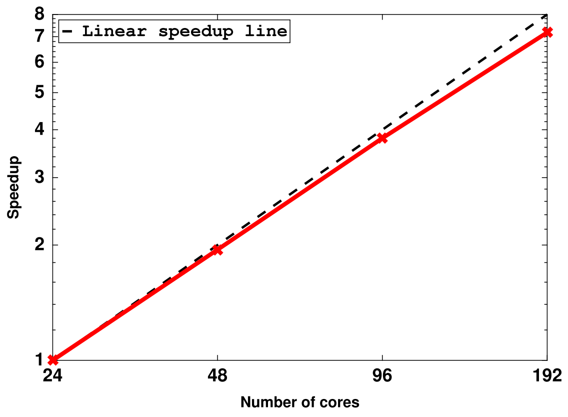
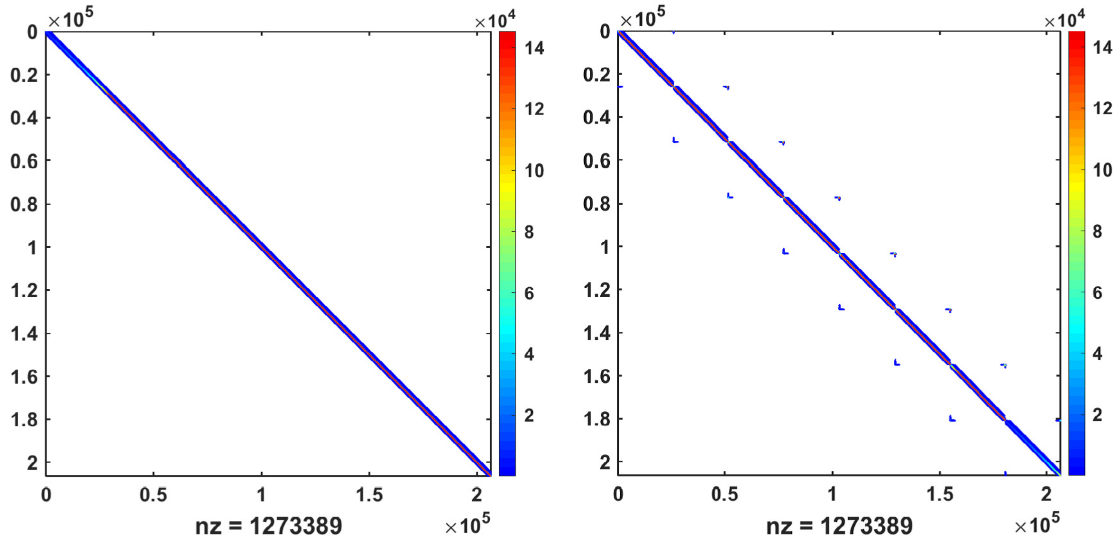


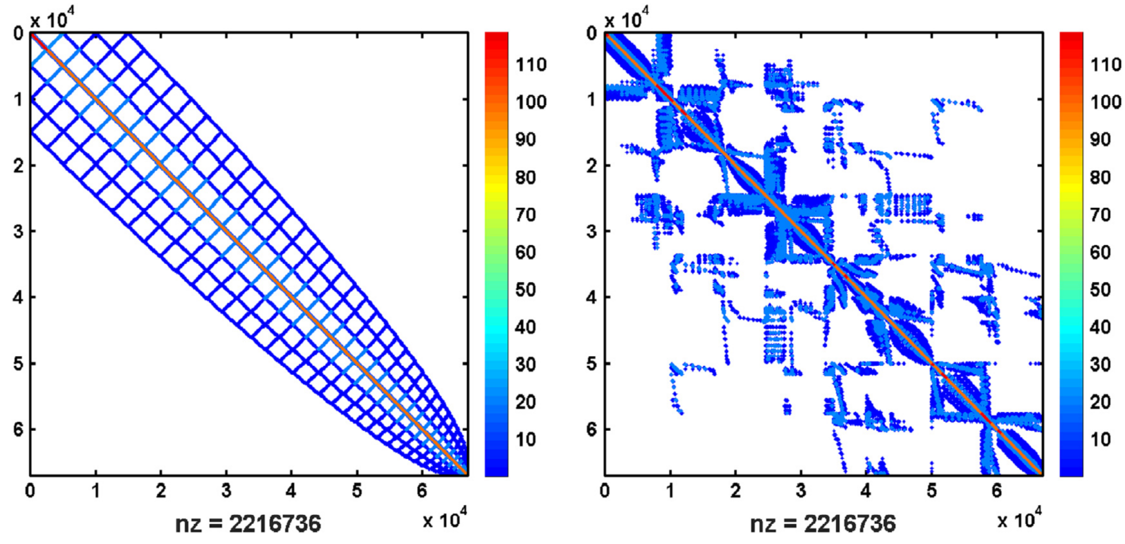
Fig. 6. Strong scaling of PSPIKE+ on a linear system defined by a Poisson 2D matrix. The number of blocks is fixed to be $p = 192$.



(a) Original.
(b) After MC64 and G-PAVER with $p = 8$.

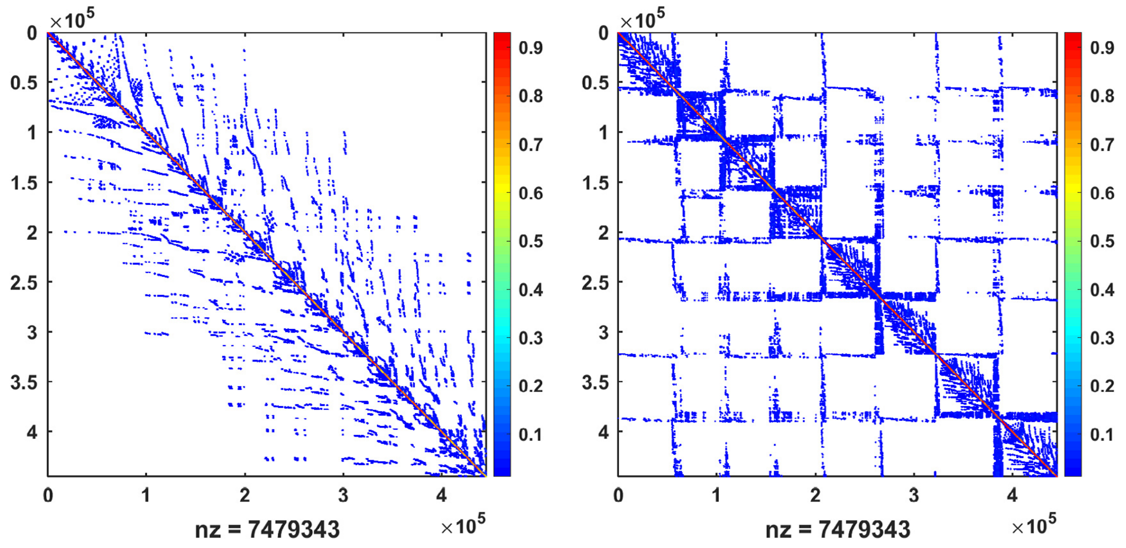
Fig. 7. Sparsity pattern of `mac_econ_fwd500`. The nonzeros are colored with respect to the magnitudes. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

on `mac_econ_fwd500` and `atmosmodl`. In contrast, although `rajat31` is of large size, PSPIKE+ barely matches the time consumed by PARDISO 5.0.0 when using $p = 8$ nodes (see Fig. 12(d)). This is due to the fact that the factorization on `rajat31` incurs very few fill-ins, which makes both the factorization and triangular solve in this case quite amenable to



(a) Original. (b) After MC64 and G-PAVER with $p = 8$.

Fig. 8. Sparsity pattern of H2O. The nonzeros are colored with respect to the magnitudes. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)



(a) Original. (b) After MC64 and G-PAVER with $p = 8$.

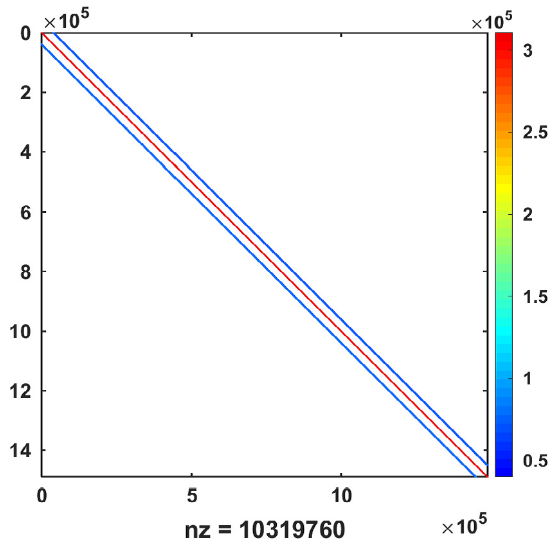
Fig. 9. Sparsity pattern of cage13. The nonzeros are colored with respect to the magnitudes. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

shared memory parallelism exploited by PARDISO 5.0.0. In addition, the communication overhead of PSPIKE+ is not offset by the reduction of the work in the factorization and triangular solve of the diagonal blocks.

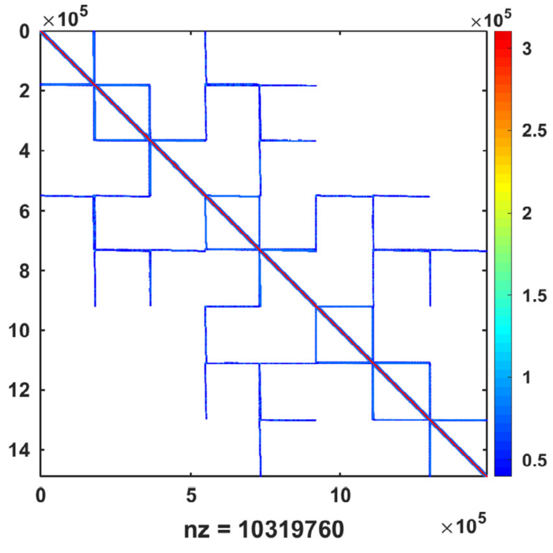
When solving cage13 directly using PARDISO 5.0.0 on a single node, the factorization of the whole matrix incurs prohibitively many fill-ins consuming all the available RAM without producing a solution. In contrast, PSPIKE+ is viable even for $p = 2$ blocks. In Fig. 13, we see that the speedup of PSPIKE+ with respect to a changing number of blocks $p = 2$, 4, and 8 is superlinear. This is because the total work in factorization and triangular solve is reduced dramatically when the number of blocks increases.

7. Conclusions

In this paper we present PSPIKE+, a family of parallel hybrid linear system solvers that could be instantiated by specific choices of the symmetric reordering, the factorization of diagonal blocks, and the solution scheme of the balance system. We propose the G-PAVER reordering algorithm that produces effective preconditioners where both the issues of load balancing

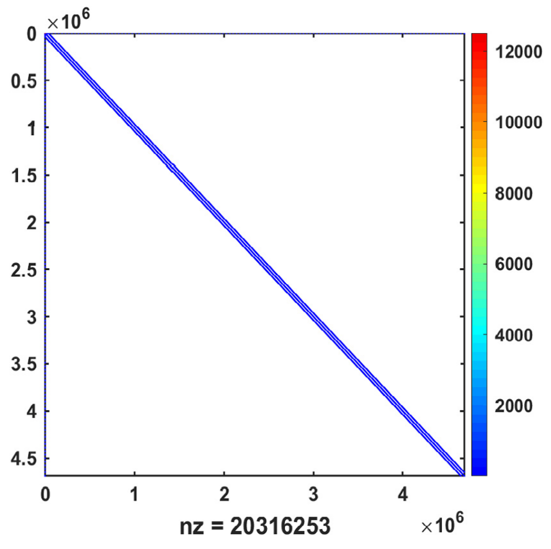


(a) Original.

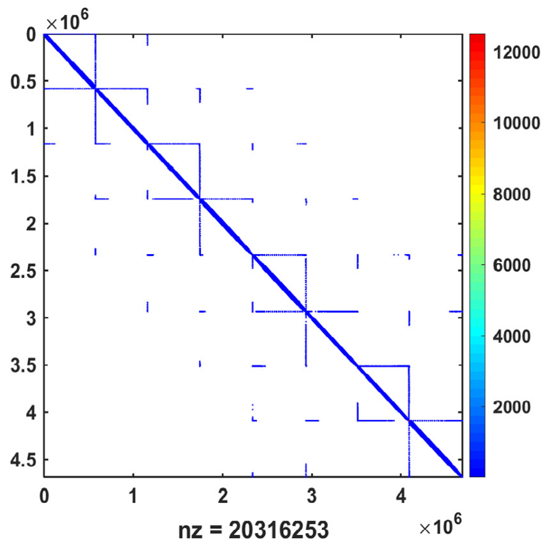


(b) After MC64 and G-PAVER with $p = 8$.

Fig. 10. Sparsity pattern of `atmosmodl`. The nonzeros are colored with respect to the magnitudes. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)



(a) Original.



(b) After MC64 and G-PAVER with $p = 8$.

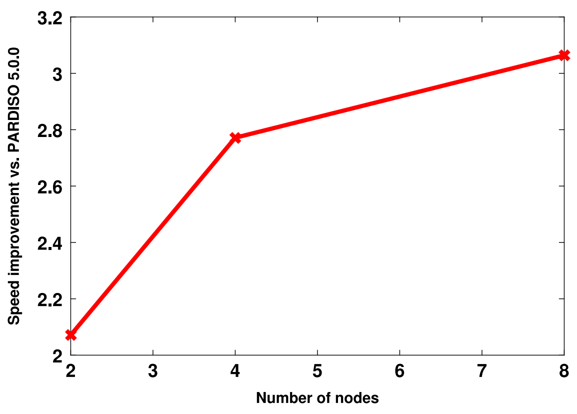
Fig. 11. Sparsity pattern of `rajat31`. The nonzeros are colored with respect to the magnitudes. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

and control of the overlap sizes are taken into account. Our parallel implementation of PSPIKE+ based on G-PAVER, PARDISO 5.0.0, and solving the explicit balance system demonstrates both robustness and parallel scalability on distributed memory architectures.

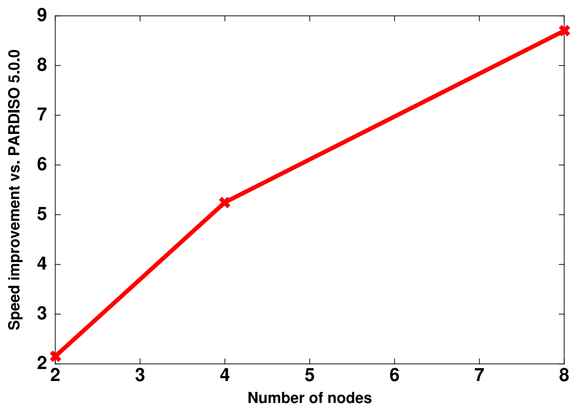
While explicitly forming and solving the balance system enhances the robustness of PSPIKE+, this direct method will not be scalable on parallel architectures with many nodes. For our future work, we will study iterative methods for solving the balance system, in which the main technical question is the construction of effective inner preconditioners to the balance system that will lead to both robustness and parallel scalability.

Acknowledgment

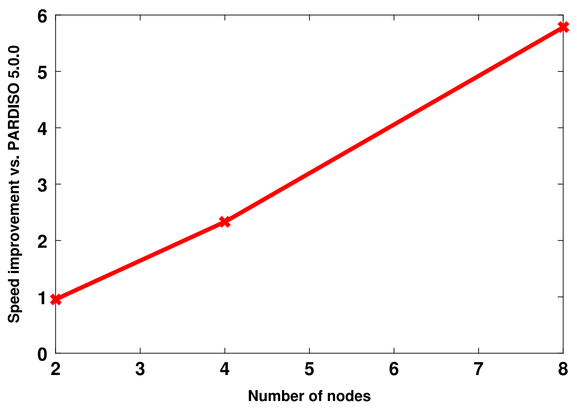
This work was supported by the Army Research Office, ARO grant number 7W911NF-11-1-0401.



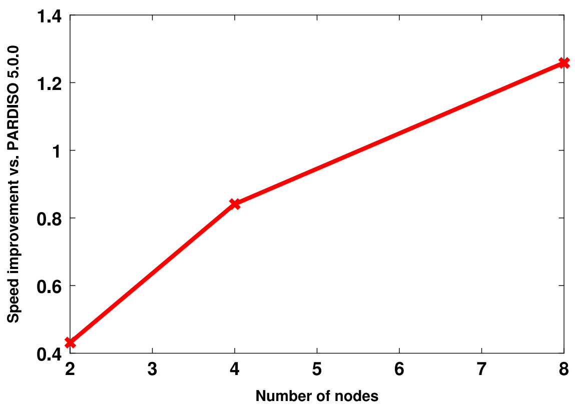
(a) `mac_econ_fwd500`, with $\tau = 400$.



(b) `H2O`, with $\tau = 200$.



(c) `atmosmodl`, with $\tau = 200$.



(d) `rajat31`, with $\tau = 200$.

Fig. 12. PSPIKE+ speed improvement vs. PARDISO 5.0.0.

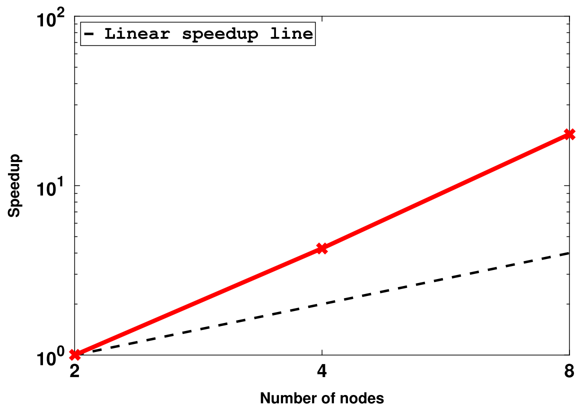


Fig. 13. Superlinear speedup of PSPIKE+ on `cage13` with respect to a changing number of blocks $p = 2, 4, 8$.

References

[1] M. Manguoglu, A.H. Sameh, O. Schenk, PSPIKE: A parallel hybrid sparse linear system solver, in: H.J. Sips, D.H.J. Epema, H. Lin (Eds.), Euro-Par, in: Lecture Notes in Computer Science, vol. 5704, Springer, 2009, pp. 797–808.

[2] O. Schenk, M. Manguoglu, A.H. Sameh, M. Christen, M. Sathe, Parallel scalable PDE-constrained optimization: antenna identification in hyperthermia cancer treatment planning, Comput. Sci.—R&D 23 (3–4) (2009) 177–183.

[3] M. Sathe, O. Schenk, B. Ucar, A. Sameh, A scalable hybrid linear solver based on combinatorial algorithms, in: U. Naumann, O. Schenk (Eds.), Combinatorial Scientific Computing, CRC Press, 2012, pp. 95–127.

[4] Y. Saad, B. Suchomel, ARMS: an algebraic recursive multilevel solver for general sparse linear systems, Numer. Linear Algebra Appl. 9 (5) (2002) 359–378.

[5] Y. Saad, J. Zhang, BILUTM: a domain-based multilevel block ILUT preconditioner for general sparse matrices, SIAM J. Matrix Anal. Appl. 21 (1) (1999) 279–299.
[6] C.K. Filelis-Papadopoulos, G.A. Gravvanis, A class of generic factored and multi-level recursive approximate inverse techniques for solving general sparse systems, Eng. Comput. 33 (1) (2016) 74–99.
[7] M. Naumov, A.H. Sameh, A tearing-based hybrid parallel banded linear system solver, J. Comput. Appl. Math. 226 (2) (2009) 306–318.
[8] M. Naumov, M. Manguoglu, A.H. Sameh, A tearing-based hybrid parallel sparse linear system solver, J. Comput. Appl. Math. 234 (10) (2010) 3025–3038.
[9] I.S. Duff, J. Koster, On algorithms for permuting large entries to the diagonal of a sparse matrix, SIAM J. Matrix Anal. Appl. 22 (2001) 973–996.
[10] M. Olschowka, A. Neumaier, A new pivoting strategy for Gaussian elimination, Linear Algebra Appl. 240 (1996) 131–151.
[11] T.A. Davis, Y. Hu, The university of Florida sparse matrix collection, ACM Trans. Math. Softw. 38 (1) (2011) 1–25.
[12] A. George, J.W. Liu, Computer Solution of Large Sparse Positive Definite, Prentice Hall Professional Technical Reference, 1981.
[13] Y.F. Hu, J.A. Scott, HSL_MC73: A fast multilevel Fiedler and profile reduction code, Tech. Rep., RAL, Oxfordshire, England, 2003.
[14] M. Manguoglu, E. Cox, F. Saied, A.H. Sameh, TRACEMIN-Fiedler: a parallel algorithm for computing the Fiedler vector, in: J.M.L.M. Palma, M.J. Dayd, O. Marques, J.C. Lopes (Eds.), VECPAR, in: Lecture Notes in Computer Science, vol. 6449, Springer, 2010, pp. 449–455.
[15] S. Acer, E. Kayaaslan, C. Aykanat, A recursive bipartitioning algorithm for permuting sparse square matrices into block diagonal form with overlap, SIAM J. Sci. Comput., 35, (1).
[16] P.R. Amestoy, I.S. Duff, J. Koster, J.-Y. L'Excellent, A fully asynchronous multifrontal solver using distributed dynamic scheduling, SIAM J. Matrix Anal. Appl. 23 (1) (2001) 15–41.
[17] T.A.T.A. Davis, Algorithm 832: UMFPACK V4.3 - an unsymmetric-pattern multifrontal method, ACM Trans. Math. Software 30 (2) (2004) 196–199.
[18] O. Schenk, K. Gärtner, Solving unsymmetric sparse systems of linear equations with PARDISO, Future Gener. Comput. Syst. 20 (3) (2004) 475–487.
[19] X.S. Li, An overview of SuperLU: algorithms, implementation, and user interface, ACM Trans. Math. Software 31 (3) (2005) 302–325.
[20] I. Moulitsas, G. Karypis, Algorithms for graph partitioning and fill reducing ordering for domain decomposition methods, Tech. Rep., RAL, Oxfordshire, England and ENSEEIHT-IRIT, Toulouse, France, 2006.
[21] K. Kaya, F.-H. Rouet, B. Uar, On partitioning problems with complex objectives, in: Euro-Par 2011: Parallel Processing Workshops, in: Lecture Notes in Computer Science, vol. 7155, Springer, 2012, pp. 334–344.
[22] D. Avis, T. Imamura, A list heuristic for vertex cover, Oper. Res. Lett. 35 (2) (2007) 201–204.
[23] F. Delbot, C. Laforest, A better list heuristic for vertex cover, Inform. Process. Lett. 107 (3–4) (2008) 125–127.
[24] D.P. Williamson, D.B. Shmoys, The Design of Approximation Algorithms, Cambridge University Press, 2011.
[25] E. Angel, R. Campigotto, C. Laforest, Implementation and comparison of heuristics for the vertex cover problem on huge graphs, in: R. Klasing (Ed.), SEA, in: Lecture Notes in Computer Science, vol. 7276, Springer, 2012, pp. 39–50.
[26] D. Fritzsche, A. Frommer, D.B. Szyld, Fast graph partitioning and applications to preconditioning, in: Fourth SIAM Workshop on Combinatorial Scientific Computing, 2009.
[27] G. Karypis, V. Kumar, A parallel algorithm for multilevel graph partitioning and sparse matrix ordering, J. Parallel Distrib. Comput. 48 (1) (1998) 71–95.
[28] I.S. Duff, K. Kaya, Preconditioners based on strong subgraphs, Electron. Trans. Numer. Anal. 40 (2013) 225–248.
[29] E. Vecharynski, Y. Saad, M. Sosonkina, Graph partitioning using matrix values for preconditioning symmetric positive definite systems, SIAM J. Sci. Comput. 36 (1) (2014) A63–A87.
[30] D. Fritzsche, A. Frommer, D.B. Szyld, Extensions of certain graph-based algorithms for preconditioning, SIAM J. Sci. Comput. 29 (2007) 2144–2161.