Reducing elimination tree height for parallel LU factorization of sparse unsymmetric matrices

Enver Kayaaslan, Bora Uçar

Reducing elimination tree height for parallel LU factorization of sparse unsymmetric matrices

Enver Kayaaslan
INRIA and LIP (UMR 5668: ENS Lyon,
CNRS, UCBL, Université de Lyon, INRIA),
46 allee d'Italie, 69364, Lyon, France
Email: enver.kayaaslan@inria.fr

Bora Uçar
CNRS and LIP (UMR 5668: ENS Lyon,
CNRS, UCBL, Université de Lyon, INRIA),
46 allee d'Italie, 69364, Lyon, France
Email: bora.ucar@ens-lyon.fr

***Abstract*—The elimination tree for unsymmetric matrices is a recent model playing important roles in sparse LU factorization. This tree captures the dependencies between the tasks of some well-known variants of sparse LU factorization. Therefore, the height of the elimination tree corresponds to the critical path length of the task dependency graph in the corresponding parallel LU factorization methods. We investigate the problem of finding minimum height elimination trees to expose a maximum degree of parallelism by minimizing the critical path length. This problem has recently been shown to be NP-complete. Therefore, we propose heuristics, which generalize the most successful approaches used for symmetric matrices to unsymmetric ones. We test the proposed heuristics on a large set of real world matrices and report 28% reduction in the elimination tree heights with respect to a common method, which exploits the state of the art tools used in Cholesky factorization.**

***Keywords*-sparse LU factorization; critical path scheduling; elimination tree; unsymmetric matrices.**

I. Introduction

The standard elimination tree [37] has been used to expose parallelism in sparse Cholesky, LU, and QR factorizations [1], [3], [22], [31]. Roughly, a set of vertices without ancestor/descendant relations corresponds to a set of independent tasks that can be performed in parallel. Therefore, the total number of parallel steps, or the critical path length, is equal to the height of the tree on an unbounded number of processors [32], [38], [39]. Obtaining an elimination tree with the minimum height for a given matrix is NP-complete [35]. Therefore, heuristic approaches are used. One set of heuristic approaches is to content oneself with the graph partitioning based methods. These methods reduce some other important cost metrics in sparse Cholesky factorization, such as the fill-in and the operation count, while giving a shallow depth elimination tree [20]. When the matrix is unsymmetric, the elimination tree for LU factorization [15] would be useful to expose parallelism as well. In this respect, the height of the tree, again, corresponds to the number of parallel steps or the critical path length for certain factorization schemes. In this work, we develop heuristics to reduce the height of elimination trees for unsymmetric matrices. To the best of our knowledge no previous work looked at this problem on its own.

Let $\mathbf{A}$ be a square sparse unsymmetric matrix and $\mathrm{G}(\mathbf{A})$ be its standard directed graph model (more formal definitions are later in Section II). Then, the minimum height of an elimination tree of $\mathbf{A}$ corresponds to the directed graph parameter cycle-rank of $\mathrm{G}(\mathbf{A})$ [14]. Gruber [19] shows that computing the cycle-rank is NP-complete, justifying the need for heuristics for large problems. Consider the directed graph G_d obtained by replacing every edge of a given undirected graph G by two directed edges point at opposite directions. Then, the cycle-rank of G_d is closely related to the undirected graph parameter tree-depth [33, p.128], which corresponds to the minimum height of an elimination tree for a symmetric matrix. One can exploit this correspondence in the reverse sense in an attempt to reduce the height of the elimination tree while producing an ordering for the matrix. That is, one can use the standard undirected graph model corresponding to the matrix $\mathbf{A} + \mathbf{A}^T$, where the addition is pattern-wise. This approach of producing an undirected graph for a given directed graph by removing the direction of each edge is used in solvers such as MUMPS [3] and SuperLU [9]. This way, the state of the art graph partitioning based ordering methods (designed for Cholesky factorization) can be used. We will compare the proposed heuristics with such methods. One can also use some local ordering heuristics [2]; but as analogous with the case of symmetric matrices, these would not be very effective in reducing the height of the elimination tree (we show evidence on this later in Section V).

The height of the elimination tree, or the critical path length, is not the only metric for efficient parallel LU factorization of unsymmetric matrices. Depending on the factorization schemes, such as Gaussian elimination with static pivoting (implemented in GESP [29]) or multifrontal based solvers (implemented for example in WSMP [21]), other metrics come into play (such as, the fill-in, the size of blocking in GESP based solvers, e.g., SuperLU_MT [10], the maximum/minimum size of fronts for parallelism in multifrontal solvers, or the communication). Nonetheless, the elimination

tree helps to give an upper bound on the performance of the suitable parallel LU factorization schemes under a fairly standard PRAM model (assuming unit size computations, infinite number of processors, and no memory/communication or scheduling overhead). Furthermore, our overall approach is based on obtaining a desirable form for LU factorization, called bordered block triangular (or BBT for short, which is summarized in Section III) form [15]. This form simplifies many algorithmic details of different solvers [15], [17] and is likely to control the fill-in, as the nonzeros are constrained to be in the blocks of a BBT form. If one orders a matrix without a BBT form and then obtains this form, the fill-in can increase or decrease [16]. Hence, our BBT based approach is likely to be helpful for LU factorization schemes exploiting the BBT form as well in controlling the fill-in.

The organization of the paper is as follows. Section II contains basic definitions and background. In Section III, we present two common LU factorization schemes and show the relation between task dependencies and the elimination tree. We propose heuristics to reduce elimination tree height in Section IV. Section V gives the experimental results and Section VI concludes the paper.

II. Basics

A *graph* $\mathcal{G}$ is an ordered pair $(\mathcal{V}, \mathcal{E})$ of a set $\mathcal{V} = \{v_1, \ldots, v_n\}$ of n vertices and a set $\mathcal{E}$ of 2-element-subsets, called edges, of $\mathcal{V}$. A graph $\mathcal{G}$ is called *bipartite* if $\mathcal{V}$ can be divided into two disjoint sets U and V such that $\{u, v\} \in \mathcal{E}$ implies $u \in U$ and $v \in V$, or vice versa.

A *directed graph (digraph)* $\mathcal{G}$ is a graph where each edge $(u, v) \in \mathcal{E}$ is an ordered pair of vertices of $\mathcal{V}$. We define $\mathcal{G}^s = (\mathcal{V}, \mathcal{E}^s)$ as the graph associated with digraph $\mathcal{G}$ such that $\mathcal{E}^s = \{\{v_i, v_j\} : (v_i, v_j) \in \mathcal{E}\}$. A digraph $\mathcal{G}$ is called *complete* if $(u, v) \in \mathcal{E}$ for all vertex pairs (u, v).

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a digraph. A sequence of vertices $u_0, \ldots, u_k$ is called a *path* of length k from u_0 to u_k if $(u_i, u_{i+1}) \in \mathcal{E}$ for all $0 \leq i < k$. A vertex u is said to be connected to v if there is a path from u and v. $\mathcal{G}$ is called *strongly connected* if u is connected to v for each pair (u, v) of vertices. A *cycle* is a path that begins and ends at the same vertex. $\mathcal{G}$ is called *acyclic* if it contains no cycles, and we call such a digraph as *directed acyclic graph (dag)*.

A digraph $G' = (V', E')$ is said to be a *subgraph* of $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ if $V' \subseteq \mathcal{V}$ and $E' \subseteq \mathcal{E}$. We call G' as the subgraph of $\mathcal{G}$ *induced* by V', denoted as $\mathcal{G}[V']$, if $E' = \mathcal{E} \cap (V' \times V')$. For a vertex set $S \subseteq \mathcal{V}$, we define $\mathcal{G} - S$ as the subgraph of $\mathcal{G}$ induced by $\mathcal{V} - S$, i.e., $G[\mathcal{V} - S]$. A vertex set $C \subseteq \mathcal{V}$ is called a *strongly connected component* of $\mathcal{G}$ if the induced subgraph $\mathcal{G}[C]$ is strongly connected, and C is maximal in the sense that for all $C' \supset C$, $G[C']$ is not strongly connected. A transitive reduction $\mathcal{G}^0$ is a minimal subgraph of $\mathcal{G}$ such that if u is connected to v in $\mathcal{G}$ then u is connected to v in $\mathcal{G}^0$ as well. If $\mathcal{G}$ is acyclic, then its transitive reduction is unique.

A *rooted tree* $\mathcal{T}$ is an ordered triple $(\mathcal{V}, r, \rho)$ where $\mathcal{V}$ is the vertex set, $r \in \mathcal{V}$ is the root vertex, and ρ gives the ancestor/descendant relation. For two vertices v_i, v_j such that $\rho(v_j) = v_i$, we call v_i as the *parent* vertex of v_j, and call v_j as a *child* vertex of v_i. With this definition the root r is the ancestor of every other node in T. The children vertices of a vertex are called *sibling* vertices to each other. The *height* of $\mathcal{T}$, denoted as $h(\mathcal{T})$, is the number of vertices in the longest path from a leaf to the root r.

Let $\mathcal{T} = (\mathcal{V}, r, \rho)$ be a rooted tree. An *ordering* $\sigma : \mathcal{V} \leftrightarrow \{1, \ldots, n\}$ is a bijective function. An ordering σ is called *topological* if $\sigma(\rho(v_j)) > \sigma(v_j)$ for all $v_j \in \mathcal{V}$. Finally, a *postordering* is a special form of topological ordering in which the vertices of each subtree are ordered consecutively.

Let $\mathbf{A}$ be an $n \times n$ unsymmetric matrix with m off-diagonal nonzero entries. The standard digraph $\mathrm{G}(\mathbf{A}) = (V, E)$ of $\mathbf{A}$ consists of a vertex set $V = \{1, \ldots, n\}$ and an edge set $E = \{(i, j) : a_{ij} \neq 0\}$ of m edges. We may use $\mathrm{D}(\mathbf{A})$ whenever the digraph is acyclic. Moreover, we may adopt $G_{\mathbf{A}}$ and $D_{\mathbf{A}}$ instead of $\mathrm{G}(\mathbf{A})$ and $\mathrm{D}(\mathbf{A})$ when a subgraph notion is required, respectively.

An *elimination digraph* $G_k(\mathbf{A}) = (V_k, E_k)$ is defined for $0 \leq k < n$ with the vertex set $V_k = \{k+1, \ldots, n\}$ and the edge set

$$E_k = \begin{cases} E & : k = 0 \\ E_{k-1} \cup \{(i, j) : (i, k), (k, j) \in E_{k-1}\} & : k > 0 \end{cases} .$$

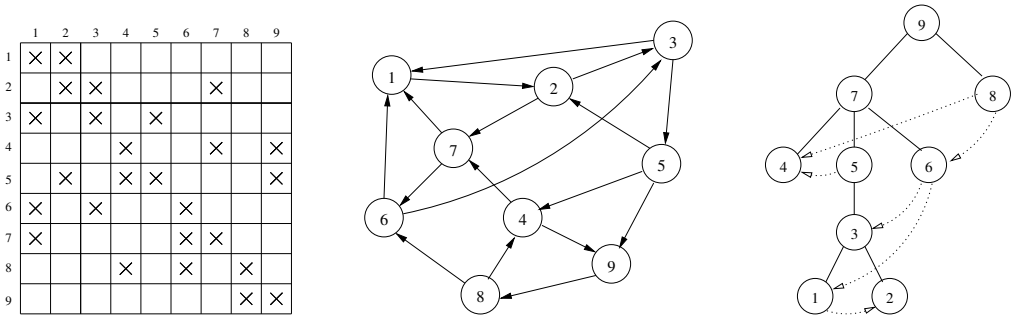
We also define $\overline{V}_k = \{1, \ldots, k\}$, for each $0 \leq k < n$.

We assume that $\mathbf{A}$ is irreducible and the LU-factorization $\mathbf{A} = \mathbf{LU}$ exists where $\mathbf{L}$ and $\mathbf{U}$ are unit lower and upper triangular, respectively. Since the factors $\mathbf{L}$ and $\mathbf{U}$ are triangular, we refer their standard digraph models as $\mathrm{D}(\mathbf{L})$ and $\mathrm{D}(\mathbf{U})$, respectively. Eisenstat and Liu [15] define the *elimination tree* $\mathrm{T}(\mathbf{A})$ as follows. Let

$$\textsc{Parent}(i) = \min\{j : j > i \text{ and } j \overset{\mathrm{D}(\mathbf{L})}{\Longrightarrow} i \overset{\mathrm{D}(\mathbf{U})}{\Longrightarrow} j\} ,$$

then $\textsc{Parent}(i)$ corresponds to the parent of vertex i in $\mathrm{T}(\mathbf{A})$ for $i < n$, and for the root n, $\textsc{Parent}(n) = \infty$. The authors also give an equivalent formulation that is defined over $\mathrm{G}(\mathbf{A})$. In this alternative, for any $i \neq n$, $\textsc{Parent}(i) = j$ whenever $j > i$ is the minimum such that the vertices i and j lie in the same strongly connected component of subgraph $G_{\mathbf{A}}[\overline{V}_j]$. Currently the algorithm with the best worst-case time complexity has a running time of $O(m \log n)$ [26]—another algorithm with the worst-case time complexity of $O(mn)$ [16] performs well in practice as well.

We now illustrate some of the definitions. Figure 1a displays a sample matrix $\mathbf{A}$ with 9 rows/columns and 26 nonzeros. Figure 1b shows the standard digraph representation $\mathrm{G}(\mathbf{A})$ of this matrix. Upon elimination of vertex 1, the three edges $(3, 2)$,$(6, 2)$, and $(7, 2)$ are added, and vertex 1 is removed, to build the elimination digraph $G_1(\mathbf{A})$. On the right side, Figure 1c shows the elimination tree $\mathrm{T}(\mathbf{A})$ where height $h(\mathrm{T}(\mathbf{A})) = 5$. Here, the parent vertex of 1 is 3, as $G_{\mathbf{A}}[\{1, 2\}]$ is not strongly connected but $G_{\mathbf{A}}[\{1, 2, 3\}]$ is (thanks to the cycle $1 \rightarrow 2 \rightarrow 3 \rightarrow 1$). The cross edges of $\mathrm{G}(\mathbf{A})$ with respect to the elimination tree $\mathrm{T}(\mathbf{A})$ are represented by dashed directed arrows in Figure 1c. The initial order is topological,



(a) The matrix $\mathbf{A}$ (b) Standard digraph $\mathrm{G}(\mathbf{A})$ (c) Elimination tree $\mathrm{T}(\mathbf{A})$

Fig. 1: A sample 9×9 unsymmetric matrix $\mathbf{A}$, the corresponding digraph $\mathrm{G}(\mathbf{A})$ and elimination tree $\mathrm{T}(\mathbf{A})$.

Algorithm 1 ROW-LU($\mathbf{A}$)

for $i \leftarrow 1$ to n **do**
  TROW(i):
  for each $k < i$ **st** $a_{ik} \neq 0$ **do**
    for each $j > k$ **st** $a_{kj} \neq 0$ **do**
      $a_{ij} \leftarrow a_{ij} - a_{kj} \times (a_{ik}/a_{kk})$

Algorithm 2 COLUMN-LU($\mathbf{A}$)

for $j \leftarrow 1$ to n **do**
  TCOL(j):
  for each $k < j$ **st** $a_{kj} \neq 0$ **do**
    for each $i > k$ **st** $a_{ik} \neq 0$ **do**
      $a_{ij} \leftarrow a_{ij} - a_{ik} \times (a_{kj}/a_{kk})$

which would be true for any matrix by definition, but not postordered in this instance. However, $8, 1, 2, 3, 5, 4, 6, 7, 9$ is a postordering of $\mathrm{T}(\mathbf{A})$, and does not respect the cross edges.

III. Equivalence between Critical Path Length and Elimination Tree Height

In this section, we summarize rowwise and columnwise LU factorization schemes, also known as ikj and jik variants [11], here to be referred as *row-LU* and *column-LU*, respectively. Then, we show that the critical path lengths for the parallel row-LU and column-LU factorization schemes are equivalent to the elimination tree height whenever the matrix has a certain form.

Let $\mathbf{A}$ be an $n \times n$ irreducible unsymmetric matrix, and $\mathbf{A} = \mathbf{LU}$, where $\mathbf{L}$ and $\mathbf{U}$ are unit lower and upper triangular, respectively. Algorithms 1 and 2 display the row-LU and column-LU factorization schemes, respectively. Both schemes update the given matrix $\mathbf{A}$ so that the $\mathbf{U} = [u_{ij}]_{i\leq j}$ factor is formed by the upper triangular (including the diagonal) entries of the matrix $\mathbf{A}$ at the end. The $\mathbf{L} = [\ell_{ij}]_{i\geq j}$ factor is formed by the multipliers ($\ell_{ik} = a_{ik}/a_{kk}$ for $i > k$) and a unit diagonal. We assume a coarse-grain parallelization approach [23], [31]. In Algorithm 1, task i is defined as the task of computing rows i of $\mathbf{L}$ and $\mathbf{U}$, and denoted as TROW(i). Similarly, in Algorithm 2, task j is defined as the task of computing the columns j of both $\mathbf{L}$ and $\mathbf{U}$ factors, and denoted as TCOL(j). The dependencies between the tasks can be represented with the digraphs $\mathrm{D}(\mathbf{L}^{\mathrm{T}})$ and $\mathrm{D}(\mathbf{U})$, for row-LU and column-LU factorization of $\mathbf{A}$, respectively.

Figure 2 illustrates the mentioned dependencies. Figure 2a shows the matrix $\mathbf{L} + \mathbf{U}$ corresponding to the factors of the sample matrix $\mathbf{A}$ given in Figure 1a. In this figure, the blue hexagons and green circles are the fill-ins in $\mathbf{L}$ and $\mathbf{U}$, respectively. Subsequently, Figures 2b and 2c show the task dependency digraphs for row-LU and column-LU factorizations, respectively. The blue dashed directed edges in Figure 2b, and the green dashed directed edges in Figure 2c correspond to the fill-ins. All edges in Figures 2b and 2c show dependencies. For example, in the column-LU, the second column depends on the first one, as the values in the first column of $\mathbf{L}$ are needed while computing the second column. This is shown in Figure 2c with a directed edge from the first vertex to the second one.

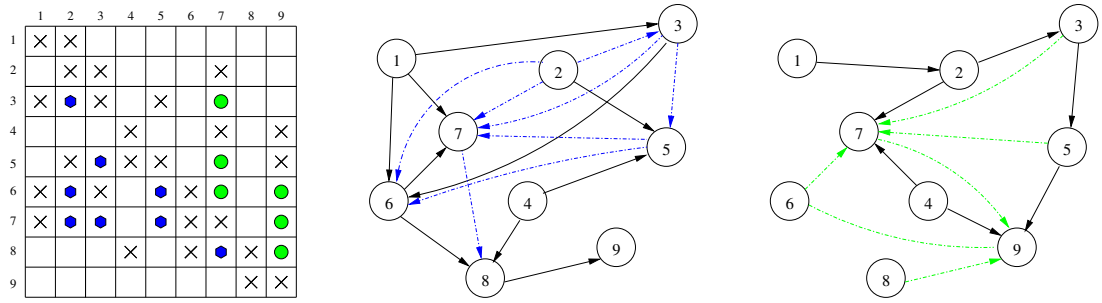
We now consider postorderings of the elimination tree $\mathrm{T}(\mathbf{A})$. Following the existing notation [15], let $\mathcal{T}[k]$ denote the set of vertices in the subtree of $\mathrm{T}(\mathbf{A})$ rooted at k. For a postordering, the order of siblings is not specified in general sense. A postordering is called *upper bordered block triangular (BBT)* if the topological order among the sibling vertices is respected, that is, a vertex k is numbered before a sibling vertex ℓ if there is an edge in $\mathrm{G}(\mathbf{A})$ that is directed from $\mathcal{T}[k]$ to $\mathcal{T}[\ell]$. Similarly, we call a postordering a *lower* BBT if the sibling vertices are ordered in reverse topological. As an example, the initial order of matrix given in Figure 1 is neither upper nor lower BBT, however, the order $8, 6, 1, 2, 3, 5, 4, 7, 9$ would be an upper BBT (see Figure 1c). The following theorem shows the relation between a BBT postordering and the task dependencies in LU factorizations.

Theorem 1 ([15]): Assuming the edges are directed from child to parent, $\mathrm{T}(\mathbf{A})$ is the transitive reduction of $\mathrm{D}(\mathbf{L}^{\mathrm{T}})$ and $\mathrm{D}(\mathbf{U})$ when $\mathbf{A}$ is upper and lower BBT ordered, respectively.

We follow this theorem with a corollary which provides the motivation for reducing the elimination tree height.

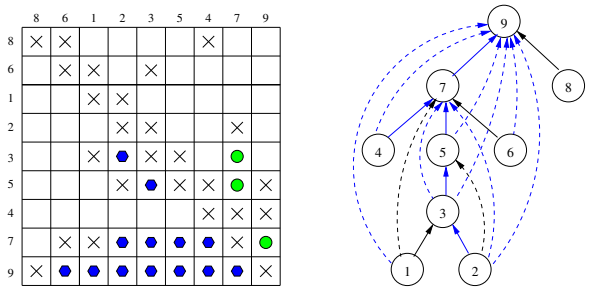
Corollary 1: The critical path length of task dependency digraph is equal to the elimination tree height $h(\mathrm{T}(\mathbf{A}))$ for row-LU factorization when $\mathbf{A}$ is upper BBT ordered, and for column-LU factorization when $\mathbf{A}$ is lower BBT ordered.

Figure 3 demonstrates the discussed points on a matrix



(a) The matrix $\mathbf{L}+\mathbf{U}$ (b) Task dependency digraph $\mathrm{D}(\mathbf{L}^{\mathrm{T}})$ (c) Task dependency digraph $\mathrm{D}(\mathbf{U})$

Fig. 2: The filled matrix $\mathbf{L}+\mathbf{U}$ (a), task dependency digraphs $\mathrm{D}(\mathbf{L}^{\mathrm{T}})$ for row-LU (b) and $\mathrm{D}(\mathbf{U})$ column-LU (c).



(a) The matrix $\widetilde{\mathbf{L}}+\widetilde{\mathbf{U}}$ (b) The digraph $\mathrm{D}(\widetilde{\mathbf{L}}^{\mathrm{T}})$

Fig. 3: The filled matrix of matrix $\widetilde{\mathbf{A}}=\widetilde{\mathbf{L}}\widetilde{\mathbf{U}}$ in BBT form (a) and the task dependency digraph $\mathrm{D}(\widetilde{\mathbf{L}}^{\mathrm{T}})$ for row-LU (b).

$\widetilde{\mathbf{A}}$, which is the permuted from $\mathbf{A}$ of Figure 1a with the upper BBT postordering $8, 6, 1, 2, 3, 5, 4, 7, 9$. Figure 3a and 3b show the filled matrix $\widetilde{\mathbf{L}}+\widetilde{\mathbf{U}}$ such that $\widetilde{\mathbf{A}}=\widetilde{\mathbf{L}}\widetilde{\mathbf{U}}$, and the corresponding task dependency digraph $\mathrm{D}(\widetilde{\mathbf{L}}^{\mathrm{T}})$ for row-LU factorization, respectively. As seen in Figure 3b, there are only tree (solid) and back (dashed) edges with respect to elimination tree $\mathrm{T}(\widetilde{\mathbf{A}})$, due to the fact that $\mathrm{T}(\widetilde{\mathbf{A}})$ is the transitive reduction of $\mathrm{D}(\widetilde{\mathbf{L}}^{\mathrm{T}})$ (see Theorem 1). In the figure, the blue edges refer to fill-in nonzeros. As seen in the figure, there is a path from each vertex to the root, and a critical path $1\rightarrow 3\rightarrow 5\rightarrow 7\rightarrow 9$ has length 5 which coincides with the height of $\mathrm{T}(\widetilde{\mathbf{A}})$.

IV. Reducing Elimination Tree Height

We propose a top-down approach to reorder a given matrix leading to a short elimination tree. The bigger lines of the proposed approach form a generalization of the state of the art methods used in Cholesky factorization (and are based on nested dissection [18]). We give the algorithms and discussions for the upper BBT form; they can be easily adapted for the lower BBT form.

A. Theoretical basis

Let $\mathbf{A}$ be an $n\times n$ sparse unsymmetric irreducible matrix and $\mathrm{T}(\mathbf{A})$ be its elimination tree. In this case, a BBT decomposition of $\mathbf{A}$ can be represented as a permutation of $\mathbf{A}$ with a permutation matrix $\mathbf{P}$ such that

$$\mathbf{A}_{\mathrm{BBT}}=\begin{bmatrix} \mathrm{A}_{11} & \mathrm{A}_{12} & \dots & \mathrm{A}_{1\mathrm{K}} & \mathrm{A}_{1\mathrm{B}} \\ & \mathrm{A}_{22} & \dots & \mathrm{A}_{2\mathrm{K}} & \mathrm{A}_{2\mathrm{B}} \\ & & \ddots & \vdots & \vdots \\ & & & \mathrm{A}_{\mathrm{KK}} & \mathrm{A}_{\mathrm{KB}} \\ \mathrm{A}_{\mathrm{B1}} & \mathrm{A}_{\mathrm{B2}} & \dots & \mathrm{A}_{\mathrm{BK}} & \mathrm{A}_{\mathrm{BB}} \end{bmatrix}, \tag{1}$$

where $\mathbf{A}_{\mathrm{BBT}}=\mathbf{PAP}^{T}$, the number of diagonal blocks $K>1$, and $\mathbf{A}_{\mathrm{kk}}$ is irreducible for each $1\leq k\leq K$. The border $\mathbf{A}_{\mathrm{B}*}$ is called *minimal* if there is no permutation matrix $\mathbf{P}'\neq\mathbf{P}$ such that $\mathbf{P}'\mathbf{A}\mathbf{P}'^{\mathrm{T}}$ is a BBT decomposition and $\mathbf{A}'_{\mathrm{B}*}\subset\mathbf{A}_{\mathrm{B}*}$. That is, the border $\mathbf{A}_{\mathrm{B}*}$ is minimal if we cannot remove columns from $\mathbf{A}_{\mathrm{B}*}$ and the corresponding rows from $\mathbf{A}_{\mathrm{B}*}$, permute them together to a diagonal block, and still have a BBT form. We give the following proposition aimed to be used for Theorem 2.

Proposition 1: Let $\mathbf{A}$ be in upper BBT form and the border $\mathbf{A}_{\mathrm{B}*}$ be minimal. The elimination digraph $\mathcal{G}_{\kappa}(\mathbf{A})$ is complete where $\kappa=\sum_{i=1}^{K}|\mathbf{A}_{\mathrm{kk}}|$.

Proof: Let us consider the elimination digraph $\mathcal{G}_{\kappa}(\mathbf{A})=(V_{\kappa},E_{\kappa})$ with $V_{\kappa}=\{\kappa+1,\dots,n\}$. Let $\mathcal{G}=\mathcal{G}_0=\mathrm{G}(\mathbf{A})$. Since $\mathbf{A}_{\mathrm{B}*}$ is minimal, $\mathcal{G}[\overline{V}_{\kappa}\cup\{b\}]$ is strongly connected for all $\kappa<b\leq n$. Take any vertex pair (b_1,b_2) such that $\kappa<b_1\neq b_2\leq n$, and any vertex $\overline{v}\in\overline{V}_{\kappa}$. Then, b_1 is connected to $\overline{v}$ in $\mathcal{G}[\overline{V}_{\kappa}\cup\{b_1\}]$ and $\overline{v}$ is connected to b_2 in $\mathcal{G}[\overline{V}_{\kappa}\cup\{b_2\}]$. Thus, b_1 is connected to b_2 in $\mathcal{G}[\overline{V}_{\kappa}\cup\{b_1,b_2\}]$. Due to the Fill Path Theorem [36], $(b_1,b_2)\in E_{\kappa}$. ■

The following theorem provides a basis for reducing the elimination tree height $h(\mathrm{T}(\mathbf{A}))$.

Theorem 2: Let $\mathbf{A}$ be in upper BBT form, and the border $\mathbf{A}_{\mathrm{B}*}$ be minimal. Then,

$$h(\mathrm{T}(\mathbf{A}))=|\mathbf{A}_{\mathrm{B}*}|+\max_{1\leq k\leq K}h(\mathrm{T}(\mathbf{A}_{\mathrm{kk}}))\ .$$

Proof: Let $\kappa_k=\sum_{i=1}^{k}|\mathbf{A}_{\mathrm{kk}}|$ for $1\leq k\leq K$, and κ be the total size of the diagonal blocks, i.e., $\kappa=\kappa_K$. For any $1\leq k\leq K$, $\textsc{Parent}(\kappa_k)=\kappa+1$. This is because of three facts: (i) each diagonal block is strongly connected; (ii) the last vertex in each block is the root of the corresponding subtree, (iii) and the border is minimal—if the vertex $\kappa+1$ does not form a strong component containing vertices from all blocks, we could have a smaller border. Due to Proposition 1, the elimination digraph

Algorithm 3 PERMUTE($\mathbf{A}$)

if $|\mathbf{A}| < \tau$ **then**
  PERMUTEBT($\mathbf{A}$) ▶ Recursion terminates
else
  $\mathbf{A}_{\mathrm{BBT}} \leftarrow$ PERMUTEBBT($\mathbf{A}$) ▶ As given in (1)
  for $k \leftarrow 1$ to K **do**
    PERMUTE($\mathbf{A}_{\mathrm{kk}}$) ▶ Subblocks of $\mathbf{A}_{\mathrm{BBT}}$

Algorithm 4 FINDSTRONGVERTEXSEPARATORES($\mathcal{G}$)

$\Pi_{ES} \leftarrow$ GRAPHBIPARTITEES($\mathcal{G}$) ▶ $\Pi_{ES} = \{V_1, V_2\}$
$\Pi_{1\mapsto 2} \leftarrow$ FINDMINCOVER($\mathcal{G}, E_{2\mapsto 1}$)
$\Pi_{2\mapsto 1} \leftarrow$ FINDMINCOVER($\mathcal{G}, E_{1\mapsto 2}$)
if $|S_{1\mapsto 2}| < |S_{2\mapsto 1}|$ **then**
  return $\Pi_{1\mapsto 2} = \{\widetilde{V}_1, \widetilde{V}_2, S_{1\mapsto 2}\}$ ▶ $\widetilde{V}_1 \mapsto \widetilde{V}_2$
else
  return $\Pi_{2\mapsto 1} = \{\widetilde{V}_1, \widetilde{V}_2, S_{2\mapsto 1}\}$ ▶ $\widetilde{V}_2 \mapsto \widetilde{V}_1$

$\mathcal{G}_\kappa(\mathbf{A})$ is complete, and hence PARENT$(b) = b + 1$, for $\kappa < b < n$. Thus, $h(\mathrm{T}(\mathbf{A}))$ is equal to size of the border $\mathbf{A}_{\mathrm{B}*}$ plus the maximum height of elimination trees of diagonal blocks $\mathbf{A}_{\mathrm{kk}}$ for $1 \leq k \leq K$. ■

The theorem says that the critical path length in the parallel LU factorization methods summarized in Section III can be expressed recursively in terms of the border and the critical path length of a block with the maximum size. This holds for row-LU scheme when the matrix $\mathbf{A}$ is in an upper BBT form, and for column-LU scheme, when $\mathbf{A}$ is in a lower BBT form. Hence, having a small border size and diagonal blocks of similar size is likely to lead reduced critical path length.

B. Recursive approach: PERMUTE

We propose a recursive approach to reorder a given matrix so that the elimination tree is reduced. Algorithm 3 gives the overview of the solution framework. The main procedure PERMUTE takes an irreducible unsymmetric matrix $\mathbf{A}$ as its input. It calls PERMUTEBBT to decompose $\mathbf{A}$ into a BBT form, $\mathbf{A}_{\mathrm{BBT}}$. Upon obtaining such a decomposition, the main procedure calls itself on each diagonal block $\mathbf{A}_{\mathrm{kk}}$, recursively. Whenever the matrix becomes sufficiently small (its size gets smaller than τ), the procedure PERMUTEBT is called in order to compute a fine BBT decomposition in which the diagonal blocks are of unit size, here to be referred as *bordered triangular* (BT) decomposition, and the recursion terminates.

C. BBT decomposition: PERMUTEBBT

Permuting $\mathbf{A}$ into a BBT form translates into finding a *strong (vertex) separator* of $\mathrm{G}(\mathbf{A})$, where the strong separator itself corresponds to the border $\mathbf{A}_{\mathrm{B}*}$. Regarding to Theorem 2, we are particularly interested in finding minimal borders.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a strongly connected directed graph. A strong separator (also called *directed vertex separator* [27]) is defined as a vertex set $S \subset \mathcal{V}$ such that $\mathcal{G} - S$ is not strongly connected. Moreover, S is said to be *minimal* if $\mathcal{G} - S'$ is strongly connected for any proper subset $S' \subset S$.

The algorithm that we propose to find a strong separator is based on bipartitioning of digraphs. In case of undirected graphs, typically, there are two kinds of graph partitioning methods which differ by their separators: *edge separators* and *vertex separators*. An edge (or vertex) separator is a set of edges (or vertices) whose removal leaves the graph disconnected.

A BBT decomposition of a matrix may have three or more subblocks (as seen in (1)). However, the algorithms we use to find a strong separator utilize 2-way partitioning, which in turn constructs two parts, each of which may potentially have several components. For the sake of simplicity in the presentation, we assume that both parts are strongly connected.

We introduce some notation in order to ease the discussion. For a pair of vertex subsets (U, W) of digraph $\mathcal{G}$, we write $U \mapsto W$, if there *may* be some edges from U to W, but there is *no* edge from W to U. Besides, $U \leftrightarrow W$ and $U \not\leftrightarrow W$ are used in a more natural way, that is, $U \leftrightarrow W$ implies that there *may* be edges in both directions, whereas $U \not\leftrightarrow W$ refers to absence of such an edge between U and W.

We cast our problem of finding strong separators as follows: For a given digraph $\mathcal{G}$, find a vertex partition $\Pi^* = \{V_1, V_2, S\}$, where S refers to a strong separator, and V_1, V_2 refer to vertex parts, so that S has small cardinality, the part sizes are close to each other, and either $V_1 \mapsto V_2$ or $V_2 \mapsto V_1$.

1) Edge-separator-based method: The first step of the method is to obtain a 2-way partition $\Pi_{ES} = \{V_1, V_2\}$ of $\mathcal{G}$ by edge separator. We build two strong separators upon Π_{ES} and choose the one with the smaller cardinality.

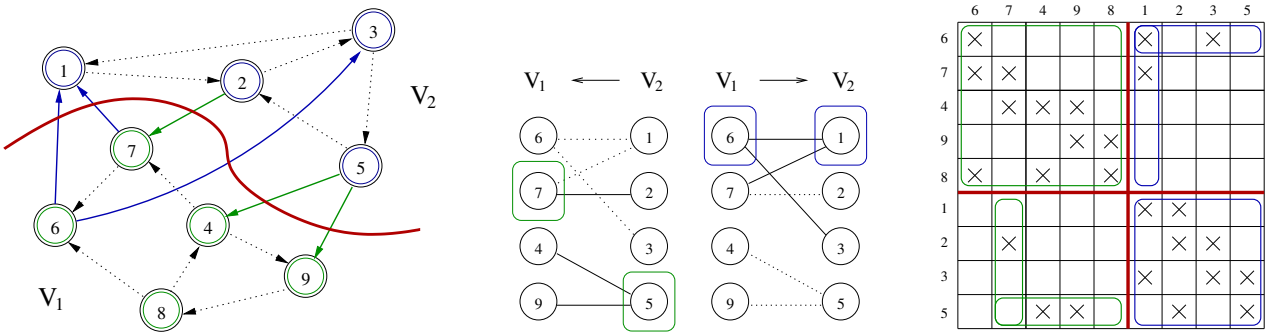
For an edge separator $\Pi_{ES} = \{V_1, V_2\}$, the *directed cut* $E_{i\mapsto j}$ is defined as the set of edges directed from V_i to V_j, i.e., $E_{i\mapsto j} = \{(u, v) \in E : u \in V_i,\ v \in V_j\}$, and we say that a vertex set S *covers* $E_{i\mapsto j}$ if for any edge $(u, v) \in E_{i\mapsto j}$, either $u \in S$ or $v \in S$, for $i \neq j \in \{1, 2\}$.

Initially, we have $V_1 \leftrightarrow V_2$. Our goal is to find a subset of vertices $S \subset V$ so that either $\widetilde{V}_1 \mapsto \widetilde{V}_2$ or $\widetilde{V}_2 \mapsto \widetilde{V}_1$, where $\widetilde{V}_1$ and $\widetilde{V}_2$ are the sets of remaining vertices in V_1 and V_2 after the removal of those vertices in S, that is, $\widetilde{V}_1 = V_1 - S$ and $\widetilde{V}_2 = V_2 - S$. The following theorem serves to the purpose of finding such a subset S.

Theorem 3: Let $\mathcal{G} = (V, E)$ be a digraph, $\Pi_{ES} = \{V_1, V_2\}$ be an edge separator of $\mathcal{G}$ and $S \subseteq V$. Now, $\widetilde{V}_i \mapsto \widetilde{V}_j$ if and only if S covers $E_{j\mapsto i}$, for $i \neq j \in \{1, 2\}$.

Proof: Take any $i \neq j \in \{1, 2\}$. (i) We assume that S covers $E_{j\mapsto i}$. Suppose there is an edge (u, v) such that $u \in \widetilde{V}_j \subseteq V_j$ and $v \in \widetilde{V}_i \subseteq V_i$. Then, $u \notin S$ and $v \notin S$, and S does not cover (u, v), which is a contradiction. (ii) We assume that $\widetilde{V}_i \mapsto \widetilde{V}_j$. Take any edge (u, v) such that $u \in V_j$, $v \in V_i$. Since $\widetilde{V}_i \mapsto \widetilde{V}_j$, either $u \notin \widetilde{V}_j$ or $v \notin \widetilde{V}_i$. Thus, $u \in S$ or $v \in S$. ■

The problem of finding a strong separator S can be encoded as finding a minimum vertex cover in the bipartite graph whose edge set is populated only by the edges of $E_{j\mapsto i}$. Algorithm 4 gives the pseudocode. As seen in the algorithm, we find two separators $S_{1\mapsto 2}$ and $S_{2\mapsto 1}$, which cover $E_{2\mapsto 1}$ and $E_{1\mapsto 2}$, and result in $\widetilde{V}_1 \mapsto \widetilde{V}_2$ and $\widetilde{V}_2 \mapsto \widetilde{V}_1$, respectively. At the end, we take the strong separator with the smaller cardinality.



(a) An edge separator $\Pi_{ES} = \{V_1, V_2\}$

(b) Covers for $E_{2\mapsto 1}$ (left) and $E_{1\mapsto 2}$ (right)

(c) Matrix view of Π_{ES}

Fig. 4: A sample 2-way partition $\Pi_{ES} = \{V_1, V_2\}$ by edge separator (a), bipartite graphs and covers for directed cuts $E_{2\mapsto 1}$ and $E_{1\mapsto 2}$ (b), and the matrix view of Π_{ES}.

In this method, how the edge separator is found matters. The objective for Π_{ES} is to minimize the cutsize $\Psi(\Pi_{ES})$ which is typically defined over the cut edges as follows:

$$\Psi_{es}(\Pi_{ES}) = |E_c| = |\{(u,v) \in E : \pi(u) \neq \pi(v)\}| \,, \qquad (2)$$

where $u \in V_{\pi(u)}$ and $v \in V_{\pi(v)}$, and E_c refers to set of cut edges. We note that the use of the above metric in Algorithm 4 is a generalization of a method used for symmetric matrices [30]. However, $\Psi_{es}(\Pi_{ES})$ is loosely related to the size of the cover found in order to build strong separator. A more suitable metric would be the number of vertices from which a cut edge emanates or the number of vertices to which a cut edge is directed. Such a cutsize metric can be formalized as follows:

$$\Psi_{cn}(\Pi_{ES}) = |V_c| = |\{v \in V : \exists (u,v) \in E_c\}| \,. \qquad (3)$$

The following theorem shows the close relation between $\Psi_{cn}(\Pi_{ES})$ and the size of the strong separator S.

Theorem 4: Let $\mathcal{G} = (V, E)$ be a digraph, $\Pi_{ES} = \{V_1, V_2\}$ be an edge separator of $\mathcal{G}$, and $\Pi^* = \{\widetilde{V}_1, \widetilde{V}_2, S\}$ be the strong separator built upon Π_{ES}. Then, $|S| \leq \Psi_{cn}(\Pi_{ES})/2$.

Proof: Let V_c denote the directed cut vertices, as defined in (3), induced by Π_{ES}. Then, $V_1 \cap V_c$ and $V_2 \cap V_c$ are valid covers for $E_{2\mapsto 1}$ for $E_{1\mapsto 2}$, respectively. Then, $|S| = \min\{|S_{1\mapsto 2}|, |S_{1\mapsto 1}|\} \leq \min\{|V_1 \cap V_c|, |V_2 \cap V_c|\}$. Hence, $|S| \leq |V_c|/2 = \Psi_{cn}(\Pi_{ES})/2$. ■

Apart from the theorem, we also note that optimizing (3) is likely to yield bipartite graphs which are easier to cover than those resulting from optimizing (2). This is because of the fact that all edges emanating from a single vertex or many different vertices are considered as the same with (2), whereas those emanating from a single vertex are preferred in (3).

We note that the cutsize metric as given in (3) can be modeled using *hypergraphs*, which is a generalization of graphs in which each edge (so called *hyperedge*) may connect more than two vertices. The hypergraph that models this cutsize metric has the same vertex set as the digraph $\mathcal{G} = (V, E)$, and a hyperedge for each vertex $v \in V$ that connects all $u \in V$ such that $(u, v) \in E$. For a given matrix, this hypergraph is called the column-net hypergraph model [4]. Any partitioning of the vertices of this hypergraph that minimizes the number of hyperedges in the cut induces a vertex partition on $\mathcal{G}$, with an edge separator that minimizes the cutsize according to the metric (3). A similar statement holds for another hypergraph constructed by reversing the edge directions (called the row-net model [4]).

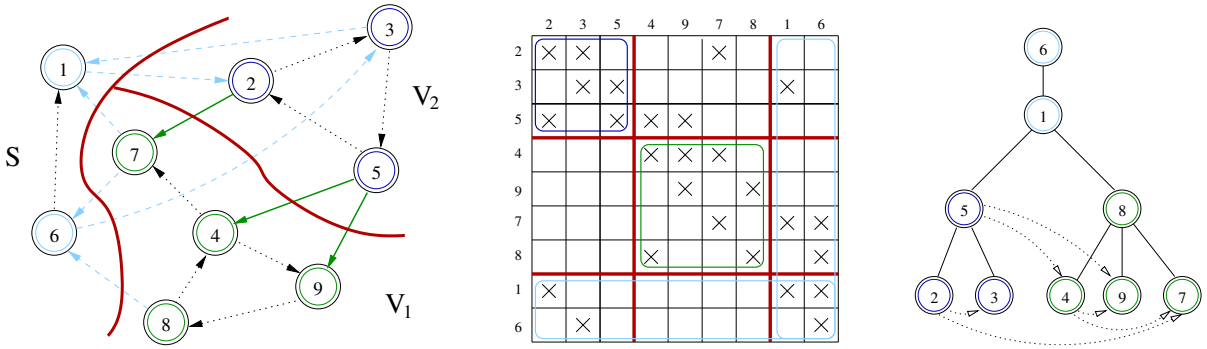
Figures 4 and 5 give an example for finding a strong separator based on an edge separator. In Figure 4a, the red curve implies the edge separator $\Pi_{ES} = \{V_1, V_2\}$ such that $V_1 = \{4, 6, 7, 8, 9\}$, the green vertices, and $V_2 = \{1, 2, 3, 5\}$, the blue vertices. The cut edges of $E_{2\mapsto 1} = \{(2,7), (5,4), (5,9)\}$ and $E_{1\mapsto 2} = \{(6,1), (6,3), (7,1)\}$ are given in color blue and green, respectively. We see two covers for the bipartite graphs corresponding to each of the directed cuts $E_{2\mapsto 1}$ and $E_{1\mapsto 2}$ in Figure 4b. As seen in Figure 4c, these directed cuts refer to nonzeros of the off-diagonal blocks in matrix view. Figure 5a gives the strong separator $\Pi_{2\to 1}$ built upon Π_{ES} by the cover $S_{2\to 1} = \{1, 6\}$ of the bipartite graph corresponding to $E_{1\to 2}$, which is given on the right of 4b. Figures 5b and 5c depict the matrix view of the strong separator $\Pi_{2\to 1}$, and the corresponding elimination tree, respectively. It is worth mentioning that in matrix view, V_2 proceeds V_1, since we use $\Pi_{2\to 1}$ instead of $\Pi_{1\to 2}$. As seen in 5c, since the permuted matrix (in Figure 5b) has an upper BBT postordering, all cross edges of the elimination tree are headed from left to right.

2) Vertex-separator-based method: A vertex separator is a strong separator restricted to that $V_1 \not\leftrightarrow V_2$. This method is based on refining the separator so that either $V_1 \mapsto V_2$ or $V_1 \mapsto V_2$. The vertex separator can be viewed as a 3-way vertex partition $\Pi_{VS} = \{V_1, V_2, \widehat{S}\}$. As in the previous method based on edge separators, we build two strong separators upon Π_{VS} and select the one having the smaller strong separator.

The criterion used to find an initial vertex separator can be formalized as

$$\Psi_{vs}(\Pi_{VS}) = |\widehat{S}| \,. \qquad (4)$$

Algorithm 5 gives the overall process to obtain a strong separator. This algorithm follows Algorithm 4 closely. Here, the procedure that refines the initial vertex separator, namely



(a) Strong separator $\Pi_{2\mapsto 1}$ (b) $\mathbf{A}_{\mathrm{BBT}}$, matrix view of $\Pi_{2\mapsto 1}$ (c) Elimination tree $\mathrm{T}(\mathbf{A}_{\mathrm{BBT}})$

Fig. 5: $\Pi_{2\mapsto 1}$, the strong separator built upon $S_{2\mapsto 1}$, the cover of $E_{1\mapsto 2}$ given in 4b (a), the matrix $\mathbf{A}_{\mathrm{BBT}}$ obtained by $\Pi_{2\mapsto 1}$ (b) and the elimination tree $\mathrm{T}(\mathbf{A}_{\mathrm{BBT}})$.

Algorithm 5 FINDSTRONGVERTEXSEPARATORVS($\mathcal{G}$)

$\Pi_{VS} \leftarrow$ GRAPHBIPARTITEVS($\mathcal{G}$) ▶ $\Pi_{VS} = \{V_1, V_2, \widehat{S}\}$
$\Pi_{1\to 2} \leftarrow$ REFINESEPARATOR($\mathcal{G}, \Pi_{VS}, 1 \to 2$)
$\Pi_{2\to 1} \leftarrow$ REFINESEPARATOR($\mathcal{G}, \Pi_{VS}, 2 \to 1$)
if $|S_{1\to 2}| < |S_{2\to 1}|$ **then**
  return $\Pi_{1\to 2} = \{\widetilde{V}_1, \widetilde{V}_2, S_{1\to 2}\}$ ▶ $\widetilde{V}_1 \to \widetilde{V}_2$
else
  return $\Pi_{2\to 1} = \{\widetilde{V}_1, \widetilde{V}_2, S_{2\to 1}\}$ ▶ $\widetilde{V}_2 \to \widetilde{V}_1$

REFINESEPARATOR, works as follows. It visits the separator vertices once, and moves the vertex at hand, if possible, to V_1 or V_2 so that $V_i \mapsto V_j$ after the movement, where $i \mapsto j$ is specified as either $1 \mapsto 2$ or $2 \mapsto 1$.

D. BT decomposition: PERMUTEBT

The problem of permuting $\mathbf{A}$ into a BT form can be decoded as finding a *feedback vertex set*, which is defined as a subset of vertices whose removal makes a digraph acyclic. In matrix view, a feedback vertex set corresponds to border $\mathbf{A}_{\mathrm{B}*}$ when the matrix is permuted into a BBT form (1), where each A_{ii}, for $i = 1, \ldots, K$, is of unit size.

The problem of finding a feedback vertex set of size less than a given value is NP-complete [24], but fixed-parameter tractable [6]. In literature, there are greedy algorithms that perform well in practice [28], [34]. In this work, we adopt the algorithm proposed by Levy and Low [28].

For example, in Figure 5, we observe that the sets $\{5\}$ and $\{8\}$ are used as the feedback-vertex sets for $\widetilde{V}_1$ and $\widetilde{V}_2$, respectively, which is the reason those rows/columns are permuted last in their corresponding subblocks.

V. EXPERIMENTS

We investigate the performance of the proposed heuristic, here to be referred as `BBT`, in terms of the elimination tree height and ordering time. We have three variants of `BBT`: (i) `BBT-es`, based on minimizing the edge cut (2); (ii) `BBT-cn`, based on minimizing the hyperedge cut (3); (iii) `BBT-vs`, based on minimizing the vertex separator (4). As a baseline, we used `MeTiS` [25] on the symmetrized matrix $\mathbf{A} + \mathbf{A}^T$. `MeTiS` uses state of the art methods to order a symmetric matrix and leads to short elimination trees. Its use in the unsymmetric case on $\mathbf{A} + \mathbf{A}^T$ is a common practice. `BBT` is implemented in `C` and compiled with `mex` of MATLAB which uses `gcc` 4.4.5. We used MeTiS library for graph partitioning according to the cutsize metrics (2) and (4). For the edge cut metric (2), we input MeTiS the graph of $\mathbf{A} + \mathbf{A}^T$, where the weight of the edge $\{i, j\}$ is 2 (both $a_{ij}, a_{ji} \neq 0$) or 1 (either $a_{ij} \neq 0$ or $a_{ji} \neq 0$, but not both). For the vertex separator metric (4), we input the graph of $\mathbf{A} + \mathbf{A}^T$ (no weights) to the corresponding MeTiS procedure. We implemented the cutsize metric given in (3) using PaToH [5] (in the `default` setting) for hypergraph partitioning using the column-net model (we did not investigate the use of row-net model). We performed the experiments on a Quad-Core AMD Opteron Processor 8356 with 32 GB of RAM.

In the experiments, for each matrix we detected dense rows and columns, where a row/column is deemed to be dense if it has at least $10\sqrt{n}$ nonzeros (as in some other ordering problems [7]). We applied MeTiS and `BBT` on the submatrix of non-dense rows/columns, and produced the final ordering by appending the dense rows/columns after the permutations obtained for the submatrices. The performance results are computed as the median of eleven runs.

Recall from Section IV-B that whenever the size of the input matrix is smaller than the parameter τ, the recursion terminates in `BBT` variants and a feedback-vertex-set-based BT decomposition is applied. We first investigate the effect of τ in order to suggest a concrete approach. For this purpose, we build a dataset of matrices, called `UFL` below. The matrices in `UFL` are from the UFL Sparse Matrix Collection [8] chosen with the following properties: square, $5\mathrm{K} \leq n \leq 100\mathrm{K}$, $nnz \leq 1\mathrm{M}$, real, and not of type `Graph`. These criteria enable an automatic selection of matrices from different application domains without having to specify the matrices individually. Other criteria could be used; ours help to select a large set of matrices each of which is not too small and not too large to be handled easily within Matlab. We exclude matrices recorded as `Graph` in the UFL collection, because most of these matrices have nonzeros from a small set of integers (for example

	Tree Height ($h(\mathbf{A})$)			Ordering Time		
τ	-es	-cn	-vs	-es	-cn	-vs
3	0.81	0.69	0.68	1.68	3.53	2.58
50	0.84	0.71	0.72	1.26	2.71	1.91
250	0.93	0.82	0.86	1.12	2.12	1.43

TABLE I: Statistical indicators of the performance of BBT variants on the UFL dataset normalized to that of MeTiS with recursion termination parameter $\tau \in \{3, 50, 250\}$.

$\{-1, 1\}$) and are reducible. We further detect matrices with the same nonzero pattern in the UFL dataset and keep only one matrix per unique nonzero pattern. We then preprocess the matrices by applying MC64 [12] in order to permute the columns so that the permuted matrix has a maximum diagonal product. Then, we identify the irreducible blocks using Dulmage-Mendelsohn decomposition [13], permute the matrices to block diagonal form and delete all entries that lie in the off-diagonal blocks. This type of preprocessing is common in similar studies [16], and corresponds to preprocessing for numerics and reducibility. After this preprocessing, we discard the matrices whose total (remaining) number of nonzeros is less than twice its size, or whose pattern symmetry is greater than 90%, where the pattern symmetry score is defined as $(nnz(\mathbf{A}) - n)/(nnz(\mathbf{A} + \mathbf{A}^{\mathrm{T}}) - n) - 0.5$. We ended up with 99 matrices in the UFL dataset.

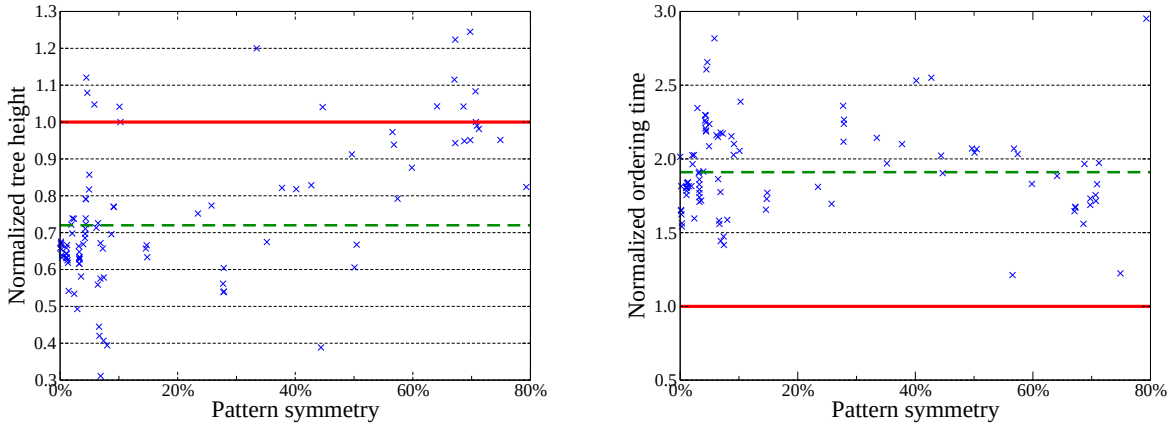
In Table I, we summarize the performance of the BBT variants as normalized to that of MeTiS on the UFL dataset for the recursion termination parameter $\tau \in \{3, 50, 250\}$. The table presents the geometric means of performance results over all matrices of the dataset. As seen in the table, for smaller values of τ, all BBT variants achieve shorter elimination trees at a cost of increased processing time to order the matrix. This is because of the fast but probably not of very high quality local ordering heuristic—as discussed before in Section IV-D, the heuristic here does not directly consider the height of the subtree corresponding to the submatrix as an optimization goal. Other τ values could also be tried but we find $\tau = 50$ as a suitable choice in general, as it performs close to $\tau = 3$ in quality and close to $\tau = 250$ in efficiency.

From Table I, we observe that BBT-vs method performs in the middle of the other two BBT variants, but close to BBT-cn, in terms of the tree height. All the three variants improve the height of the tree with respect to MeTiS. Specifically, at $\tau = 50$, BBT-es, BBT-cn, and BBT-vs obtain improvements of 16%, 29% and 28%, respectively. The graph based methods BBT-es and BBT-vs run slower than MeTiS, as they contain additional overheads of obtaining the covers and refining the separators of $\mathbf{A} + \mathbf{A}^T$ for $\mathbf{A}$. For example, at $\tau = 50$, the overheads result in 26% and 171% longer ordering time for BBT-es and BBT-vs, respectively. As expected, the hypergraph based method BBT-cn is much slower than others (for example, 112% longer ordering time with respect to MeTiS with $\tau = 50$). Since we identify BBT-vs as fast and of high quality (it is close to the best BBT variant in terms of the tree height and the fastest one in terms of the running time), we display its individual performance results in detail, with $\tau = 50$, in Figure 6. In Figures 6a and 6b, each individual point represents the ratio BBT-vs / MeTiS, in terms of tree height and ordering time, respectively. As seen in Figure 6a, BBT-vs reduces the elimination tree height on 85 out of 99 matrices in the UFL dataset, and achieves a reduction of 28%, in terms of the geometric mean, with respect to MeTiS (the green dashed line in the figure represents the geometric mean). We observe that the pattern symmetry and the reduction in the elimination tree height highly correlate after a certain pattern symmetry score. More precisely, the correlation coefficient between the pattern symmetry and the normalized tree height is 0.6187 for those matrices with pattern symmetry greater than 20%. This is of course in agreement with the fact that LU factorization methods exploiting the unsymmetric pattern have much to gain on matrices with lower pattern symmetry score and less to gain otherwise than the alternatives. On the other hand, as seen in Figure 6b, BBT-vs performs consistently slower than MeTiS, with an average of 91% slow down.

We now investigate the performance of BBT variants on the matrices used by Eisenstat and Liu [16] in their original work on the elimination tree. The set of these matrices is called EL below. We note that EL dataset contains matrices on which LU factorization methods that exploit the unsymmetric nonzero pattern make sense (if for example a matrix is symmetric, there is nothing to exploit; if the symmetry score is high, there is only little to gain). Therefore, the reductions in the elimination tree height achieved by the BBT variants on these matrices represent the potential gain in a selected set of practical problems. We applied the same preprocessing as before. A matrix, with the name Lucifora/cell2 is not included, since it has the same nonzero pattern with Lucifora/cell1 (we note that the all matrices in EL, except Lucifora/cell1, are contained in UFL).

Table II presents the performance results of BBT variants with $\tau = 50$ on the matrices from the EL dataset in terms of both the elimination tree height and ordering time (in seconds). The second and third columns respectively shows the number of rows/columns (n), and the number of nonzeros (nnz) of corresponding matrices. The fourth column of the table gives the pattern symmetry of the matrices computed after preprocessing. The first half of the remaining columns show the tree heights and the second half gives ordering time results for the methods MeTiS and BBT. The performance results for MeTiS are given as their exact values, whereas the performance of the three BBT variants is presented after normalizing with respect to those of MeTiS.

Table II reveals that the proposed BBT-es, BBT-cn, and BBT-vs methods obtain, respectively, 24%, 35%, and 31% reduction in the elimination tree height with respect to MeTiS, on average, on the EL dataset (the rankings are in accordance with the previous results). Upon a closer look at the data, we see that except two instances, BBT-vs performs better than BBT-es, whereas BBT-cn performs uniformly better than BBT-es. The reason that BBT-vs obtains better results than BBT-es is twofold. First, finding vertex separators with specialized heuristics on undirected graphs usually leads to



(a) BBT-vs / MeTiS Tree Height ($h(\mathbf{A})$)

(b) BBT-vs / MeTiS Ordering Time

Fig. 6: BBT-vs / MeTiS performance on the UFL dataset. Dashed, green lines mark the geometric mean of the ratios.

	Properties			Tree Height ($h(\mathbf{A})$)				Ordering Time			
				MeTiS	BBT ($\tau = 50$)			MeTiS	BBT ($\tau = 50$)		
name	n	nnz	sym		-es	-cn	-vs	(sec)	-es	-cn	-vs
Bai/rw5151	5151	20199	0.0%	260	0.72	0.48	0.63	0.040	1.00	2.00	2.00
Hollinger/g7jac040	11790	107383	1.5%	991	0.78	0.57	0.63	0.200	1.30	2.55	1.80
Hohn/fd18	16428	63406	2.1%	389	0.81	0.75	0.74	0.160	1.50	5.12	1.87
Hohn/fd15	11532	44206	2.1%	329	0.78	0.73	0.71	0.100	1.60	5.20	1.90
Hohn/fd12	7500	28462	2.4%	273	0.74	0.71	0.69	0.060	1.67	5.33	1.83
Nasa/barth4	6019	23492	2.9%	154	0.66	0.37	0.45	0.040	1.25	3.00	2.00
Grund/bayer10	13436	71594	3.2%	204	0.74	0.58	0.59	0.180	1.11	2.83	1.83
Grund/bayer02	13935	63307	3.9%	170	0.76	0.59	0.66	0.140	1.36	3.36	2.00
Nasa/barth5	15606	61484	6.4%	157	0.71	0.49	0.66	0.120	1.25	3.17	2.17
Hamrle/Hamrle2	5952	22162	7.3%	103	0.68	0.58	0.62	0.040	1.50	4.25	2.00
Nasa/barth	6691	26439	8.7%	142	0.70	0.55	0.69	0.040	1.75	5.00	2.25
Hohn/sinc12	7500	283992	14.6%	1614	0.74	0.66	0.66	0.400	1.30	2.35	1.62
TOKAMAK/utm5940	5940	83842	37.7%	488	0.80	0.71	0.81	0.100	1.90	4.80	1.90
Goodwin/goodwin	7320	324772	40.2%	443	0.84	0.95	0.83	0.210	2.90	7.10	2.43
Graham/graham1	9035	335472	42.7%	544	0.88	0.93	0.80	0.200	3.00	7.05	2.75
Averous/epb1	14734	95053	57.4%	418	0.77	0.78	0.75	0.130	2.00	7.77	2.00
Lucifora/cell1	7055	30082	66.2%	194	0.65	0.58	0.88	0.040	1.50	5.25	2.00
Shen/e40r0100	17281	553562	79.3%	620	0.90	1.04	0.84	0.360	3.39	8.22	2.75
min					*0.65*	*0.37*	*0.45*		*1.00*	*2.00*	*1.62*
max					*0.90*	*1.04*	*0.88*		*3.39*	*8.22*	*2.75*
geomean				**317**	**0.76**	**0.65**	**0.69**	**0.109**	**1.64**	**4.31**	**2.04**

TABLE II: Performance results of EL dataset in terms of elimination tree height ($h(\mathbf{A})$) and ordering time for the methods MeTiS, BBT-es, BBT-cn and BBT-vs. The performance values of BBT variants are given as normalized to those of MeTiS. BBT variants use the recursion termination parameter $\tau = 50$. Matrices are in the nondecreasing order of pattern symmetry.

smaller separators than finding first an edge cut and then refining it with the bipartite cover algorithms (this is observed in MeTiS). On top of this, we refine the resulting separator using the heuristic discussed in Section IV-C2, potentially yielding much better separator size with BBT-vs than BBT-es. The reason that BBT-cn obtains better results than BBT-es is related to the difference in the objective functions (2) and (3). As said before, BBT-cn is likely to obtain bipartite graphs that have smaller covers than those that are obtained by BBT-es. The ratio of the size of the covers to the number of vertices and to the number of edges in the covered bipartite graphs were smaller in BBT-cn than BBT-es in the EL dataset.

We also examined the height of the elimination trees obtained by Eisenstat and Liu [16, Table 3] with a local ordering heuristic [2] on the same dataset. We have noted that the height of those trees are uniformly larger than those obtained by the three BBT variants (the geometric mean of the tree heights normalized with respect to MeTiS was 2.79 on the EL dataset). This is in accordance with the observations for the Cholesky factorization [20], where the graph partitioning based methods usually lead to shallower trees than the local ordering methods.

The fill-in generally increases [16] when a matrix is reordered into a BBT form with respect to the original ordering using the elimination tree. We noticed that BBT-vs almost always increases the number of nonzeros in $\mathbf{L}$ (by 157% with respect to MeTiS in the EL dataset). However, the number of nonzeros in $\mathbf{U}$ almost always decreases (by 15% with respect to MeTiS in the EL dataset). This observation may be exploited during Gaussian elimination where $\mathbf{L}$ is not stored.

VI. Conclusion

We investigated the elimination tree model for unsymmetric matrices as a means of capturing dependencies among the tasks in row-LU and column-LU factorization schemes. Specifically, we focused on permuting a given unsymmetric matrix to obtain an elimination tree with reduced height in order to expose higher degree of parallelism by minimizing the critical path length in parallel LU factorization schemes. Based on the theoretical findings, we proposed a heuristic, which orders a given matrix to a bordered block diagonal form with a small border size and blocks of similar sizes, and then locally orders each block. We presented three variants of the proposed heuristic. These three variants achieved, on average, 24%, 31%, and 35% improvements with respect to a common method of using MeTiS (a state of the art tool used for the Cholesky factorization) on a small set of matrices. On a larger set of matrices, the performance improvements were 16%, 28%, and 29%. The best performing variant is about 2.71 times slower than MeTiS on the larger set, while others are slower by factors of 1.26 and 1.91.

As a future direction, we believe that a direct method to find strong separators would give better performance in terms of both quality and ordering runtime.

Acknowledgements

This work was supported by French National Research Agency (ANR) project SOLHAR (ANR-13-MONU-0007).

References

[1] E. Agullo, A. Buttari, A. Guermouche, and F. Lopez, "Multifrontal QR factorization for multicore architectures over runtime systems," in *Euro-Par 2013 Parallel Processing*, ser. LNCS. Springer Berlin Heidelberg, 2013, vol. 8097, pp. 521–532.

[2] P. Amestoy, X. Li, and E. Ng, "Diagonal Markowitz scheme with local symmetrization," *SIAM J. Matrix Anal. Appl.*, vol. 29, no. 1, pp. 228–244, 2007.

[3] P. R. Amestoy, I. S. Duff, and J.-Y. L'Excellent, "Multifrontal parallel distributed symmetric and unsymmetric solvers," *Comput. Method. Appl. M.*, vol. 184, no. 2–4, pp. 501–520, 2000.

[4] Ü. V. Çatalyürek and C. Aykanat, "Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication," *IEEE T. Parall. Distr.*, vol. 10, no. 7, pp. 673–693, 1999.

[5] ——, *PaToH: A Multilevel Hypergraph Partitioning Tool, Version 3.0*, Bilkent Univ., Dept. Computer Engineering, Ankara, 06533 Turkey. Available at http://bmi.osu.edu/~umit/software.htm, 1999.

[6] J. Chen, Y. Liu, S. Lu, B. O'Sullivan, and I. Razgon, "A fixed-parameter algorithm for the directed feedback vertex set problem," *J. ACM*, vol. 55, no. 5, pp. 21:1–21:19, 2008.

[7] T. A. Davis, J. R. Gilbert, S. I. Larimore, and E. G. Ng, "A column approximate minimum degree ordering algorithm," *ACM Trans. Math. Softw.*, vol. 30, no. 3, pp. 353–376, 2004.

[8] T. A. Davis and Y. Hu, "The University of Florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, pp. 1:1–1:25, 2011.

[9] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu, "A supernodal approach to sparse partial pivoting," *SIAM J. Matrix Anal. Appl.*, vol. 20, no. 3, pp. 720–755, 1999.

[10] J. W. Demmel, J. R. Gilbert, and X. S. Li, "An asynchronous parallel supernodal algorithm for sparse gaussian elimination," *SIAM J. Matrix Anal. Appl.*, vol. 20, no. 4, pp. 915–952, 1999.

[11] J. J. Dongarra, F. G. Gustavson, and A. Karp, "Implementing linear algebra algorithms for dense matrices on a vector pipeline machine," *SIAM Rev.*, vol. 26, no. 1, pp. pp. 91–112, 1984.

[12] I. S. Duff and J. Koster, "On algorithms for permuting large entries to the diagonal of a sparse matrix," *SIAM J. Matrix Anal. Appl.*, vol. 22, no. 4, pp. 973–996, 2000.

[13] A. L. Dulmage and N. S. Mendelsohn, "Coverings of bipartite graphs," *Can. J. Math.*, vol. 10, pp. 517–534, 1958.

[14] L. C. Eggan, "Transition graphs and the star-height of regular events." *Mich. Math. J.*, vol. 10, no. 4, pp. 385–397, 1963.

[15] S. C. Eisenstat and J. W. H. Liu, "The theory of elimination trees for sparse unsymmetric matrices," *SIAM J. Matrix Anal. Appl.*, vol. 26, no. 3, pp. 686–705, 2005.

[16] ——, "Algorithmic aspects of elimination trees for sparse unsymmetric matrices," *SIAM J. Matrix Anal. Appl.*, vol. 29, no. 4, pp. 1363–1381, 2008.

[17] S. C. Eisenstat and J. W. Liu, "A tree-based dataflow model for the unsymmetric multifrontal method," *Electronic Transactions on Numerical Analysis*, vol. 21, pp. 1–19, 2005.

[18] A. George, "Nested dissection of a regular finite element mesh," *SIAM J. Numer. Anal.*, vol. 10, no. 2, pp. 345–363, 1973.

[19] H. Gruber, "Digraph complexity measures and applications in formal language theory," *Discrete Mathematics & Theoretical Computer Science*, vol. 14, no. 2, pp. 189–204, 2012.

[20] A. Guermouche, J.-Y. L'Excellent, and G. Utard, "Impact of reordering on the memory of a multifrontal solver," *Parallel Comput.*, vol. 29, no. 9, pp. 1191–1218, 2003.

[21] A. Gupta, "Improved symbolic and numerical factorization algorithms for unsymmetric sparse matrices," *SIAM J. Matrix Anal. Appl.*, vol. 24, no. 2, pp. 529–552, 2002.

[22] J. Hogg, J. Reid, and J. Scott, "Design of a multicore sparse Cholesky factorization using DAGs," *SIAM J. Sci. Comp.*, vol. 32, no. 6, pp. 3627–3649, 2010.

[23] J. A. G. Jess and H. G. M. Kees, "A data structure for parallel L/U decomposition," *IEEE Trans. Comput.*, vol. 31, no. 3, pp. 231–239, 1982.

[24] R. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, ser. The IBM Research Symposiax, R. Miller, J. Thatcher, and J. Bohlinger, Eds. Springer US, 1972, pp. 85–103.

[25] G. Karypis and V. Kumar, "A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices," *Univ. Minnesota, Dept. Computer Science and Engineering, Army HPC Research Center, Minneapolis, MN*, 1998.

[26] K. Kaya and B. Uçar, "Constructing elimination trees for sparse unsymmetric matrices," *SIAM J. Matrix Anal. Appl.*, vol. 34, no. 2, pp. 345–354, 2013.

[27] S. Kintali, N. Kothari, and A. Kumar, "Approximation algorithms for directed width parameters," *CoRR*, vol. abs/1107.4824, 2011.

[28] H. Levy and D. W. Low, "A contraction algorithm for finding small cycle cutsets," *J. Algorithms*, vol. 9, no. 4, pp. 470–493, 1988.

[29] X. S. Li and J. W. Demmel, "Making sparse gaussian elimination scalable by static pivoting," in *Proc. of the 1998 ACM/IEEE Conference on Supercomputing*, ser. SC '98, Washington, DC, USA, 1998, pp. 1–17.

[30] J. W. H. Liu, "A graph partitioning algorithm by node separators," *ACM Trans. Math. Softw.*, vol. 15, no. 3, pp. 198–219, 1989.

[31] J. W. Liu, "Computational models and task scheduling for parallel sparse Cholesky factorization," *Parallel Comput.*, vol. 3, no. 4, pp. 327–342, 1986.

[32] ——, "Reordering sparse matrices for parallel elimination," *Parallel Comput.*, vol. 11, no. 1, pp. 73 – 91, 1989.

[33] J. Nešetřil and P. Ossona de Mendez, "Tree-depth, subgraph coloring and homomorphism bounds," *Eur. J. Combin.*, vol. 27, no. 6, pp. 1022–1041, 2006.

[34] P. Pardalos, T. Qian, and M. Resende, "A greedy randomized adaptive search procedure for the feedback vertex set problem," *J. Comb. Optim.*, vol. 2, no. 4, pp. 399–412, 1998.

[35] A. Pothen, "The complexity of optimal elimination trees," Pennsylvania State Univ., Tech. Rep. CS-88-13, 1988.

[36] D. J. Rose and R. E. Tarjan, "Algorithmic aspects of vertex elimination on directed graphs," *SIAM J. Appl. Math.*, vol. 34, no. 1, pp. 176–197, 1978.

[37] R. Schreiber, "A new implementation of sparse Gaussian elimination," *ACM Trans. Math. Softw.*, vol. 8, no. 3, pp. 256–276, 1982.

[38] O. Wing and J. Huang, "A computation model of parallel solution of linear equations," *IEEE Trans. Comput.*, vol. C-29, no. 7, pp. 632–638, 1980.

[39] T. Yang and A. Gerasoulis, "DSC: scheduling parallel tasks on an unbounded number of processors," *IEEE T. Parall. Distr.*, vol. 5, no. 9, pp. 951–967, 1994.