RAIRO-Oper. Res. 55 (2021) S905–S927
https://doi.org/10.1051/ro/2020013

RAIRO Operations Research
www.rairo-ro.org

PARALLEL GAUSSIAN ELIMINATION OF SYMMETRIC POSITIVE DEFINITE BAND MATRICES FOR SHARED-MEMORY MULTICORE ARCHITECTURES

Sirine Marrakchi* and Mohamed Jemni

Abstract. This study presents a new parallel Gaussian elimination approach for symmetric positive definite band systems. For each task, the appropriate start time and adequate processor are determined. Unnecessary dependencies between tasks are eliminated. Simultaneously, all processors perform their associated tasks with precedence constraints under consideration. Our main goal is to obtain a high degree of parallelism by balancing the load of processors and reducing the total idle and parallel execution times. The theoretical lower bounds for parallel execution time and number of processors required to execute the precedence graph at an optimal time are also computed. The validity of our investigation is confirmed by carrying out several experiments on a shared-memory multicore architecture using OpenMP. Practical results prove the efficiency of the proposed method.

Mathematics Subject Classification. 65F05, 65F50, 65Y05, 68W10, 90B35, 05C50.

Received June 11, 2019. Accepted February 10, 2020.

1. Introduction

Solving sparse triangular linear systems is a crucial building block in numerous numerical linear algebra methods [21, 22] which they can be classified as either direct or iterative. Direct methods generate a result in a finite number of steps, whereas iterative methods yield a set of consecutive approximations that converge to the precise result [24].

This paper focuses on a classical direct method named Gaussian elimination (GE). The core of the GE algorithm consists to produce an upper triangular system equivalent to the initial one after successive eliminations of unknowns [4]. For large matrices, the execution time when performing the GE algorithm sequentially is high. The concept of parallelism should be employed to achieve good efficiency.

In the literature, many researchers proposed various parallel approaches for GE using non-singular square dense matrices for various models of computations [2, 20, 23, 27, 29]. McGinn and Shaw [23] developed a parallel algorithm for GE without pivoting using OpenMP, as well as a distributed implementation using MPI. Bampis *et al.* [2] proposed a scheduling algorithm that consists to reduce the overhead by splitting the tasks of GE with a smaller granularity in order to keep longer a full efficiency. The obtained result is improved by Saad [29]. Parallel approaches of GE algorithm with partial pivoting have been discussed in [20, 27] on MIMD

Keywords. Gaussian elimination, symmetric positive definite band systems, parallel execution time, shared-memory multicore architecture, efficiency.

Research Laboratory of Technologies of Information and Communication & Electrical Engineering (LaTICE), ENSIT, University of Tunis, Tunis, Tunisia.
*Corresponding author: sirine.marrakchi@yahoo.com

computers. However, to the authors' best knowledge, few available publications have studied the parallelization of GE method for band matrices. Milovanović *et al.* [25] determined an optimal number of processors according to the number of diagonals in the matrix and they proposed a task scheduling algorithm for parallel GE with partial pivoting of band matrices. Chishti *et al.* [8] proposed a parallel GE algorithm for symmetric positive definite (SPD) band matrices. Processors execute sequences of tasks in parallel. After achieving the diagonal task, a processor should immediately broadcast the result to all other processors.

For general matrices, the pivoting strategy is required to ensure numerical stability [30]. It consists, on one hand, to switch the order of the equations for reducing round-off errors and on the other hand to prevent diagonal elements from becoming zero by making each diagonal element larger than any other entries belong on the same column of the given matrix. Unlike general matrices, GE applied to an SPD matrix is numerically stable, and pivoting is not needed [12]. In this paper, we are interested in band structure which is a special case of sparse matrices whose nonzero entries are all closely near the main diagonal. Calculations involving SPD band matrices commonly arise in the numerical treatment of many models encountered in various scientific and engineering applications [18, 35] such as the discretization of partial differential equations [14]. The symmetry property can advantageously be exploited for saving both the memory storage and the computing time requirements.

In this study, a new approach that improves the GE without pivoting using SPD band matrices is developed. For each task in the precedence graph, the suitable start time and an available processor are determined. Dependencies between tasks unused in the parallel processing stage are removed. All processors then perform their associated tasks simultaneously, considering the remaining dependencies. By doing so, our approach aims to attain remarkable performance advancement by balancing the load of processors, minimizing the total idle time and parallel runtime. Theoretically, the lower bounds for parallel execution time and the number of processors required to perform the precedence graph within the shortest possible parallel execution time are computed. The evaluation and the validation of our contribution were carried out by a series of experiments performed on a shared-memory multicore machine using the OpenMP interface. Thus, the efficiency of the proposed approach is verified by comparing the results with those obtained from the determined theoretical formulas and row block method.

The paper is organized as follows: Section 2 introduces the GE sequential algorithm and defines essential concepts. In Section 3, we determine the theoretical lower bounds for parallel execution time and the number of processors. Section 4 describes our practical investigation in details. Section 5 presents, analyses and discusses the experimental results. Section 6 offers concluding remarks and potential future extensions of this work.

2. Preliminaries

Let $M = (m_{i,j})$ where $1 \leq i, j \leq n$ be a real non-singular SPD band matrix if the non-negative integers r and s exist with the properties [1, 13]:

- $m_{i,j} = 0$ if $j - i > r$ and $i - j > s$ (band structure). The integers r and s represent the number of diagonals above and below the main diagonal, respectively. The bandwidth L of the matrix M is equal to $r + s + 1$ diagonals and the total number of non-zero elements for general band matrix $nz_{\text{General\_Band\_M}}$ can be computed by the following equation:

$$\begin{aligned} nz_{\text{General\_Band\_M}} &= \underbrace{\left(\sum_{i=1}^{n-r} r + \sum_{i=n-r+1}^{n-1} (n-i)\right)}_{\text{Strictly upper part of } M} + \underbrace{\left(\sum_{j=1}^{n-s} s + \sum_{j=n-s+1}^{n-1} (n-j)\right)}_{\text{Strictly lower part of } M} + \underbrace{n}_{\text{Main diagonal}} \\ &= \frac{2nr - r^2 - r + 2ns - s^2 - s + 2n}{2} \\ &= \frac{2n(r+s+1) - \Big((r+s+1)^2 - (r+s+1) - 2rs\Big)}{2} \\ &= \frac{L(2n - L + 1)}{2} + rs. \end{aligned} \tag{1}$$

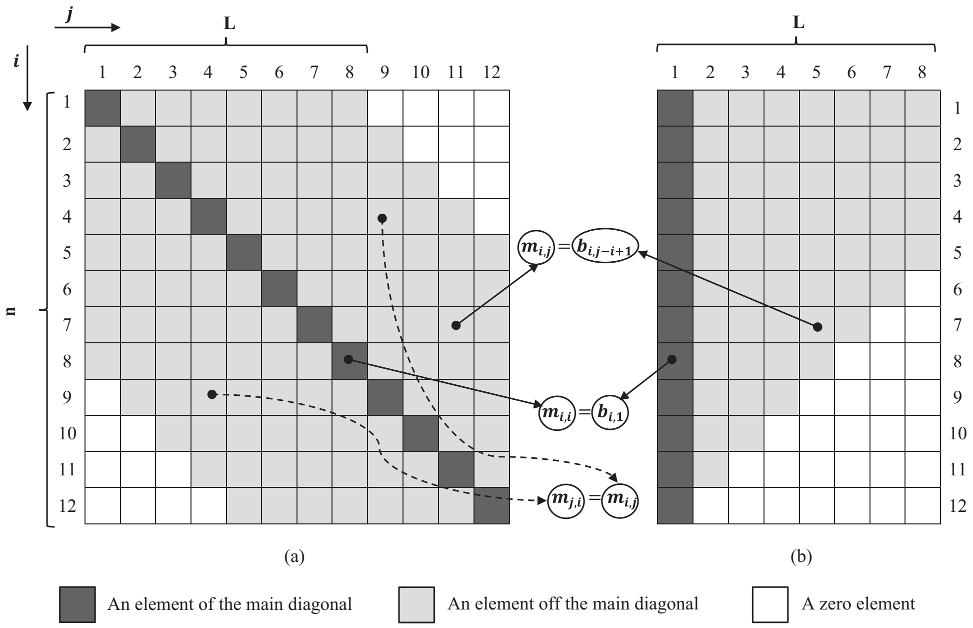

FIGURE 1. An illustrative example of symmetric band matrix stored in (a) 2-D array and (b) BND format.

- $m_{j,i} = m_{i,j}\ \forall\ i, j$, *i.e.*, M is equal to its transpose M^T (symmetry property). In this case, we have $r = s$ and $m_{i,j} = 0$ if $|i-j| > r$. Thus, the bandwidth L of the matrix M and the total number of non-zero elements for symmetric band matrix become equal to $2r + 1$ and $\frac{L(2n-L+1)}{2} + r^2$, respectively. In practice, the symmetry characteristic is beneficial in saving memory space and shortening computing time. For this matrix type, only the lower or the upper triangular part needs to be maintained. The bandwidth L is thereby reduced to $r + 1$. Equation (2) expresses the number of non-zero entries $nz_{\text{Symmetric\_Band\_M}}$ according to L.

$$nz_{\text{Symmetric\_Band\_M}} = \sum_{i=1}^{n-L+1} L + \sum_{i=n-L+2}^{n} (n-i+1) = \frac{L(2n-L+1)}{2}. \tag{2}$$

- The following statements are equivalent to ascertain that M is positive definite [9, 17, 32]:
  - $z^T M z > 0$ for any non-zero real vector z of order n where z^T denotes the transpose of z.
  - The determinants of the leading principal sub-matrices M_q are positive ($det(M_q) > 0$ where $1 \le q \le n$).
  - All eigenvalues of M are positive.
  - All pivots of M in the GE without pivoting are positive.

In this paper, the banded Linpack (BND) format is used to save only diagonals constituting the upper triangular part of the matrix M in a rectangular array B column by column. The bandwidth L represents the minimum number of columns needed in B [28]. Figure 1 illustrates an example of symmetric band matrix where $n = 12$ and $L = 8$ stored in (a) 2-D array and (b) BND format.

The sequential algorithm of GE method without pivoting is presented in Figure 2 where the SPD band matrix is stored in ⓐ 2-D array and ⓑ BND format. The value of element $m_{i,j}$ after achieving the iteration k is

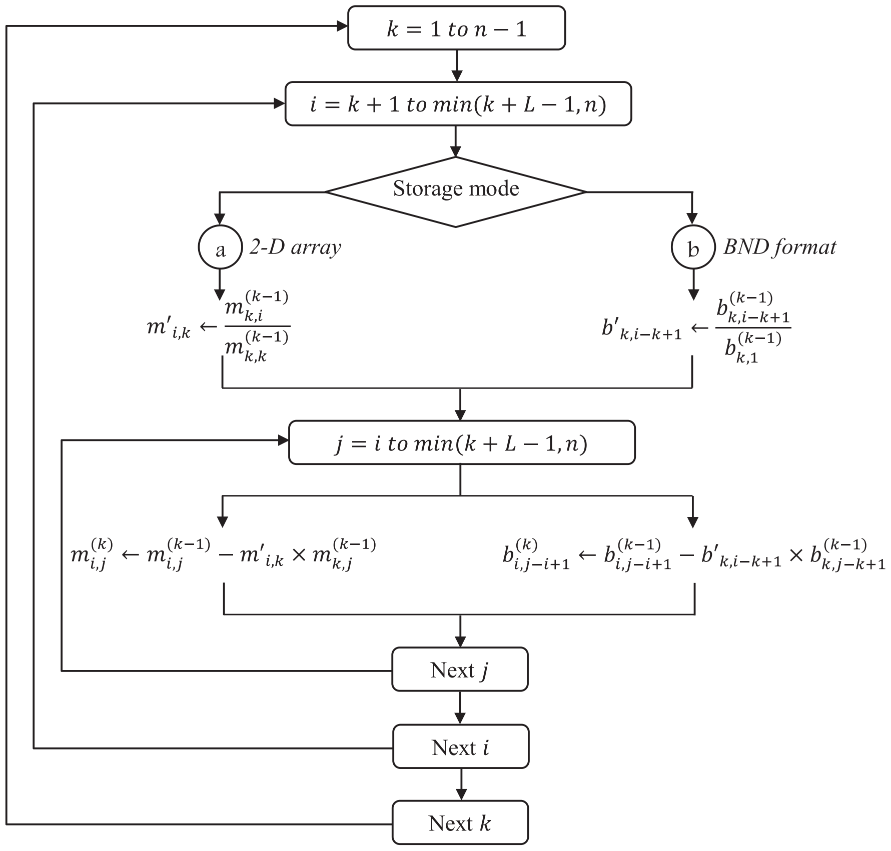
$k = 1$ to $n-1$
$i = k+1$ to $min(k+L-1, n)$
Storage mode
a 2-D array
b BND format
$m'_{i,k} \leftarrow \frac{m^{(k-1)}_{k,i}}{m^{(k-1)}_{k,k}}$
$b'_{k,i-k+1} \leftarrow \frac{b^{(k-1)}_{k,i-k+1}}{b^{(k-1)}_{k,1}}$
$j = i$ to $min(k+L-1, n)$
$m^{(k)}_{i,j} \leftarrow m^{(k-1)}_{i,j} - m'_{i,k} \times m^{(k-1)}_{k,j}$
$b^{(k)}_{i,j-i+1} \leftarrow b^{(k-1)}_{i,j-i+1} - b'_{k,i-k+1} \times b^{(k-1)}_{k,j-k+1}$
Next j
Next i
Next k

FIGURE 2. GE serial algorithm without pivoting for SPD band matrix stored in (a) 2-D array and (b) BND format.

TABLE 1. Task decomposition of the GE algorithm for SPD band matrix.

Role	Task $T_{k,k}$ is the update of matrix row k during iteration k	Task $T_{k,i}$ is the update of matrix row i during iteration k
(a)	$m'_{i,k} \leftarrow \frac{m^{(k-1)}_{k,i}}{m^{(k-1)}_{k,k}}$ $k+1 \leq i \leq ub$	$m^{(k)}_{i,j} \leftarrow m^{(k-1)}_{i,j} - m'_{i,k} \times m^{(k-1)}_{k,j}$ $i \leq j \leq ub$
(b)	$b'_{k,i-k+1} \leftarrow \frac{b^{(k-1)}_{k,i-k+1}}{b^{(k-1)}_{k,1}}$ $k+1 \leq i \leq ub$	$b^{(k)}_{i,j-i+1} \leftarrow b^{(k-1)}_{i,j-i+1} - b'_{k,i-k+1} \times b^{(k-1)}_{k,j-k+1}$ $i \leq j \leq ub$

designated by $m^{(k)}_{i,j}$ where $1 \leq k \leq n-1$ and $m^{(0)}_{i,j}$ refer to the initial value of $m_{i,j}$. The multipliers computed in (a) and (b) are saved in a lower triangular matrix M' and in a band matrix B', respectively [3,5].

By assuming $T_{u,v}$ represents a task, v is the matrix row which will be updated and u is the algorithm iteration during which the task will be executed where $u = k$ and $v = k$ or $v = i$. For simplicity, let define $ub = \min(k+L-1, n)$ where $L > 2$. The task decomposition of GE algorithm is detailed in Table 1.

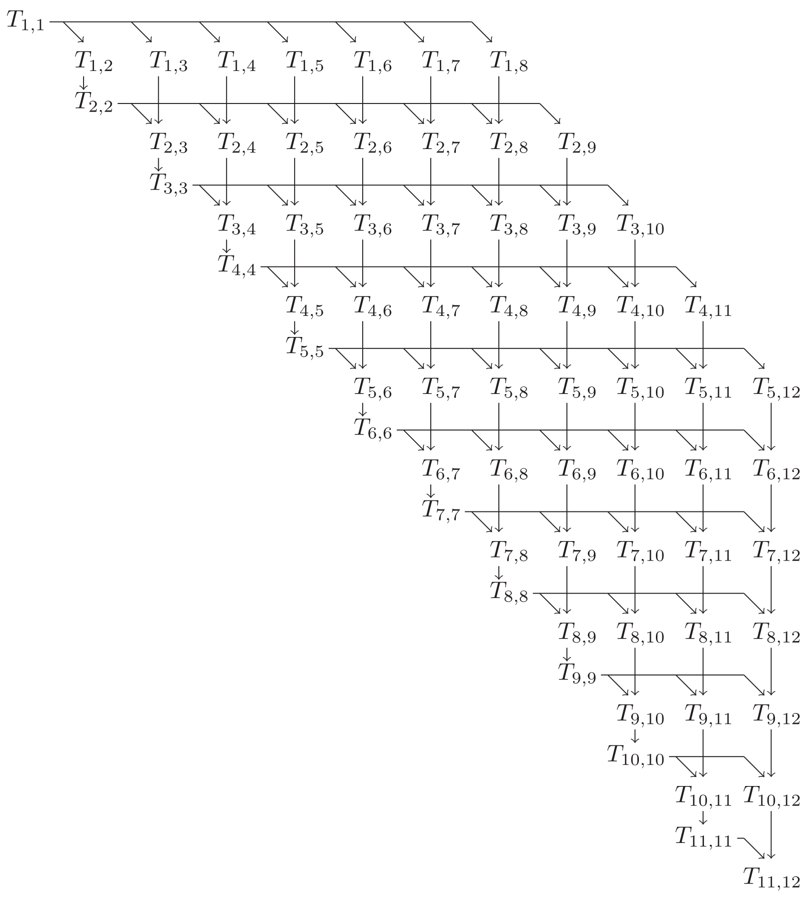

FIGURE 3. Precedence graph for $n = 12$ and $L = 8$.

Two types of constraints are distinguished: $T_{k,k} \ll T_{k,i}$ and $T_{k,i} \ll T_{k+1,i}$ where $1 \leq k \leq n-1$, $k+1 \leq i \leq ub$. $T_{u',v'} \ll T_{u,v}$ indicates that the task $T_{u',v'}$ is a predecessor of $T_{u,v}$, and $T_{u,v}$ must wait for $T_{u',v'}$ to finish its execution before starting its own. The precedence graph is directly built from these constraints and is composed of two sets named nodes and edges, representing the tasks and dependencies, respectively. It is referred to as directed acyclic graph [7, 11, 33]. Figure 3 outlines the precedence graph for $n = 12$ and $L = 8$.

Each task has a weight W indicating the number of time steps needed for its execution. One-time step consists of one multiplication and one subtraction or one division. The cost of $T_{k,k}$ and $T_{k,i}$ is calculated by equations (3) and (4), respectively.

$$W(T_{k,k}) = \begin{cases} L-1 & \text{if } 1 \leq k \leq n-L+1 \\ n-k & \text{if } n-L+2 \leq k \leq n-1 \end{cases} \tag{3}$$

$$W(T_{k,i}) = \begin{cases} L-i+k & \text{if } 1 \leq k \leq n-L+1,\ k+1 \leq i \leq k+L-1 \\ n-i+1 & \text{if } n-L+2 \leq k \leq n-1,\ k+1 \leq i \leq n. \end{cases} \tag{4}$$

The total number of tasks can be computed by equation (5).

$$N_{\text{tasks}} = \underbrace{(n-1)}_{\text{Number of diagonal tasks}} + \underbrace{\sum_{k=1}^{n-L+1}(L-1) + \sum_{k=n-L+2}^{n-1}(n-k)}_{\text{Number of non-diagonal tasks}}$$
$$= \frac{L(2n-L+1)}{2} - 1. \tag{5}$$

A diagonal task $T_{k,k}$ has $L-1$ successors if $1 \le k \le n-L+1$ and $n-k$ successors if $n-L+2 \le k \le n-1$. Each non-diagonal task $T_{k,i}$ has one successor where $1 \le k \le n-2$ and $k+1 \le i \le ub$. The final task $T_{n-1,n}$ has no successor. Thus, the total number of edges (*i.e.*, dependencies) can be determined by equation (6).

$$N_{\text{edges}} = \sum_{k=1}^{n-L+1}(L-1) + \sum_{k=n-L+2}^{n-1}(n-k) + \sum_{k=1}^{n-L+1}\left(\sum_{i=k+1}^{k+L-1} 1\right) + \sum_{k=n-L+2}^{n-2}\left(\sum_{i=k+1}^{n} 1\right)$$
$$= 2 \times \left(\sum_{k=1}^{n-L+1} L-1\right) + 2 \times \left(\sum_{k=n-L+2}^{n-1} n-k\right) - 1$$
$$= (L-1)(2n-L) - 1. \tag{6}$$

The sequential execution time t_1 given by equation (7) represents the total cost of tasks.

$$t_1 = \frac{L(L+1)(3n-2L+2)}{6} - n. \tag{7}$$

The critical path (CP) is the longest directed path from the entry task with no incoming edge to the final task with no outgoing edge [16, 34]. It is composed of the tasks:

$$T_{1,1}, T_{1,2}, \ldots, T_{k,k}, T_{k,k+1}, \ldots, T_{n-1,n-1}, T_{n-1,n}.$$

The length of CP can be computed by equation (8). It is equal to the lower bound for parallel execution time $t_{\text{opt}}(n, L)$.

$$t_{\text{opt}}(n,L) = \sum_{k=1}^{n-L+1}\left(\underbrace{L-1}_{W(T_{k,k})} + \underbrace{L-(k+1)+k}_{W(T_{k,k+1})}\right) + \sum_{k=n-L+2}^{n-1}\left(\underbrace{n-k}_{W(T_{k,k})} + \underbrace{n-(k+1)+1}_{W(T_{k,k+1})}\right)$$
$$= 2\sum_{k=1}^{n-L+1}(L-1) + 2\sum_{k=n-L+2}^{n-1}(n-k)$$
$$= (L-1)(2n-L). \tag{8}$$

3. Proposed theoretical formulas

In this section, we determine the theoretical lower bounds for parallel execution time $t_{\text{opt},p}(n, L)$ using p processors where $2 \le p < p_{\text{opt}}(n, L)$, and number of processors $p_{\text{opt}}(n, L)$ executing the precedence graph in optimal time $t_{\text{opt}}(n, L)$.

3.1. Determination of the lower bound for parallel execution time

The precedence graph is split into three parts as shown in Figure 4. In the next, we present the signification of each part.

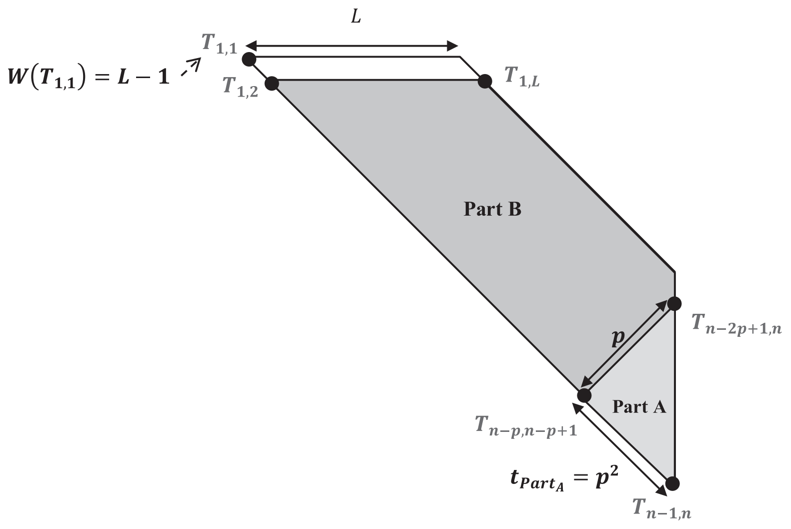

FIGURE 4. Overview of precedence graph decomposition.

- $T_{1,1}$ is an entry task.
- $Part_A$ contains the following tasks:

$$\begin{cases} T_{k,n-p+j}, & n-p-j+1 \leq k \leq n-p+j-1 \text{ and } 1 \leq j \leq p \\ T_{n-p+j,n-p+j}, & 1 \leq j \leq p-1. \end{cases}$$

- $Part_B$ includes all tasks except those from $Part_A$ and task $T_{1,1}$.

The cost time of $T_{1,1}$, $Part_A$ and $Part_B$ are given respectively by equations (9), (10) and (11).

$$W(T_{1,1}) = L - 1 \tag{9}$$

$$\begin{aligned} W(Part_A) &= \sum_{j=1}^{p} \left(\sum_{k=n-p-j+1}^{n-p+j-1} W(T_{k,n-p+j}) \right) + \sum_{j=1}^{p-1} W(T_{n-p+j,n-p+j}) \\ &= \sum_{j=1}^{p} \left(\sum_{k=n-p-j+1}^{n-p+j-1} \Big(n - (n-p+j) + 1\Big) \right) + \sum_{j=1}^{p-1} \Big(n - (n-p+j)\Big) \\ &= \sum_{j=1}^{p} (2j-1)(p-j+1) + \sum_{j=1}^{p-1} (p-j) \\ &= \frac{p^3 + 3p^2 - p}{3} \end{aligned} \tag{10}$$

$$W(Part_B) = t_1 - W(T_{1,1}) - W(Part_A). \tag{11}$$

The minimum time required to execute $Part_A$ using p processors is equal to

$$\begin{aligned} t_{Part_A} &= \sum_{k=n-p}^{n-1} W(T_{k,k+1}) + \sum_{k=n-p+1}^{n-1} W(T_{k,k}) \\ &= \sum_{k=n-p}^{n-1} (n - (k+1) + 1) + \sum_{k=n-p+1}^{n-1} (n-k) \end{aligned}$$

$$\begin{aligned} &= \sum_{k=n-p}^{n-1} (n-k) + \sum_{k=n-p+1}^{n-1} (n-k) \\ &= p + 2 \sum_{k=n-p+1}^{n-1} (n-k) \\ &= p^2. \end{aligned} \tag{12}$$

By assuming that all processors are active after executing $T_{1,1}$ until the moment $t_{n-p,n-p+1}$ (*i.e.*, the time when execution of the task $T_{n-p,n-p+1}$ begins). Thus, the following equation is obtained:

$$\sum_{j=1}^{p} \Big(t_{n-p-j+1,n-p+j} - W(T_{1,1}) \Big) = W(Part_B). \tag{13}$$

Equation (14) can be written as follows:

$$t_{n-p,n-p+1} + \sum_{j=1}^{p-1} \Big(t_{n-p-j,n-p+j+1} \Big) = W(Part_B) + p \times W(T_{1,1}). \tag{14}$$

The cost difference between the two tasks $T_{k,n-p+j}$ and $T_{k-1,n-p+j+1}$ is equal to one where $1 \le j \le p-1$ and $n-p-j+1 \le k \le n-p+j-1$. Thus, the time difference between $t_{n-p-j,n-p+j+1}$ and $t_{n-p-j+1,n-p+j}$ can be computed by equation (15).

$$\begin{aligned} t_{n-p-j,n-p+j+1} - t_{n-p-j+1,n-p+j} &= \sum_{k=n-p-j+1}^{n-p+j-1} \big(W(T_{k,n-p+j}) - W(T_{k-1,n-p+j+1}) \big) \\ &= 2j - 1. \end{aligned} \tag{15}$$

Thereby, the time difference between $t_{n-p-j,n-p+j+1}$ where $1 \le j \le p-1$ and $t_{n-p,n-p+1}$ is equal to

$$\begin{aligned} t_{n-p-j,n-p+j+1} &= t_{n-p,n-p+1} + \sum_{\beta=1}^{j} (2\beta - 1) \\ &= t_{n-p,n-p+1} + j^2. \end{aligned} \tag{16}$$

Replacing the obtained expression in equation (14), we have:

$$p \times t_{n-p,n-p+1} + \sum_{j=1}^{p-1} j^2 = W(Part_B) + p \times W(T_{1,1}). \tag{17}$$

The equation that determines the time $t_{n-p,n-p+1}$ is defined as:

$$\begin{aligned} t_{n-p,n-p+1} &= \left\lceil \frac{t_1 + (p-1) \times W(T_{1,1}) - W(Part_A) - \sum_{j=1}^{p-1} j^2}{p} \right\rceil \\ &= \left\lceil \frac{3nL^2 + 3nL - 6n - 2L^3 - 4L + 6 - 4p^3 - 3p^2 - 5p}{6p} \right\rceil + L. \end{aligned} \tag{18}$$

The lower bound for parallel execution time $t_{\text{opt},p}(n, L)$ is equal to the sum of $t_{n-p,n-p+1}$ and t_{Part_A}.

$$t_{\text{opt},p}(n, L) = t_{n-p,n-p+1} + t_{Part_A}. \tag{19}$$

Therefore, the final equation of $t_{\text{opt},p}(n, L)$ is obtained.

$$t_{\text{opt},p}(n, L) = \left\lceil \frac{3nL^2 + 3nL - 6n - 2L^3 - 4L + 6 - 4p^3 - 3p^2 - 5p}{6p} \right\rceil + L + p^2. \tag{20}$$

Theorem 3.1. *With p processors $(2 \le p \le L-1)$, the lower bound for parallel execution time is expressed as:*

$$t_{\text{opt},p}(n, L) = \begin{cases} \left\lceil \frac{2p^3-3p^2+p(6L-5)+3n(L^2+L-2)-2L^3-4L+6}{6p} \right\rceil, & 2 \le p < p_{\text{opt}} \\ t_{\text{opt}}(n, L) = (L-1)(2n-L), & p_{\text{opt}} \le p \le L-1. \end{cases} \tag{21}$$

3.2. Determination of the lower bound for number of processors

Considering the function $F_{n,L}(x)$ which correspond to equation (20).

$$F_{n,L}(x) = \frac{2x^3 - 3x^2 + x(6L-5) + 3nL^2 + 3nL - 6n - 2L^3 - 4L + 6}{6x}. \tag{22}$$

The theoretical lower bound for number of processors $p_{\text{opt}}(n, L)$ required to perform the precedence graph in time $t_{\text{opt}}(n, L) = (L-1)(2n-L)$ represents the smallest positive integer verifying $F_{n,L}(x) - t_{\text{opt}}(n, L) \le 0$. Let firstly determine the boundaries of $p_{\text{opt}}(n, L)$. The minimum bound $b_{\min}$ is equal to the upper integer part of $\frac{t_1}{t_{\text{opt}}(n,L)}$. It is expressed as:

$$b_{\min} = \left\lceil \frac{L(L+1)(3n-2L+2) - 6n}{6(L-1)(2n-L)} \right\rceil. \tag{23}$$

The maximum bound $b_{\max}$ is equal to

$$b_{\max} = \max_{2 \le k \le n-1} \left(\left\lceil \frac{\sum_{i=k+1}^{\min(k+L-2,n)} W(T_{k-1,i})}{W(T_{k-1,k}) + W(T_{k,k})} \right\rceil \right) + 1. \tag{24}$$

Three cases are distinguished and defined in equation (25).

$$\begin{cases} \left\lceil \frac{(L-2)(L-1)}{4L-4} \right\rceil + 1, & 2 \le k \le n-L+1 \\ \left\lceil \frac{(L-2)(L-1)}{4L-6} \right\rceil + 1, & k = n-L+2 \\ \left\lceil \frac{(n-k)(n-k+1)}{2(2n-2k+1)} \right\rceil + 1, & n-L+3 \le k \le n-1. \end{cases} \tag{25}$$

To determine the maximum bound $b_{\max}$, the three cases are compared.

- Having $4L-4 > 4L-6$ thereby $\frac{1}{4L-4} < \frac{1}{4L-6}$ and finally we obtain $\left\lceil \frac{(L-2)(L-1)}{4L-4} \right\rceil + 1 \ge \left\lceil \frac{(L-2)(L-1)}{4L-6} \right\rceil + 1$.
- If $k = n-L+3$, so the maximum number of active processors is equal to $\left\lceil \frac{(L-2)(L-1)}{4L-10} \right\rceil + 1$. Having $4L-4 > 4L-10$ thereby $\frac{1}{4L-4} < \frac{1}{4L-10}$ and finally we obtain $\left\lceil \frac{(L-2)(L-1)}{4L-4} \right\rceil + 1 \ge \left\lceil \frac{(L-2)(L-1)}{4L-10} \right\rceil + 1$.

- If $k = n-1$, so the maximum number of active processors is equal to 2. We have $\left\lceil \frac{(L-2)(L-1)}{4L-4} \right\rceil + 1 = \left\lceil \frac{L-2}{4} \right\rceil + 1$ and $L > 2$ so $\left\lceil \frac{(L-2)(L-1)}{4L-4} \right\rceil + 1 \geq 2$.

As is clear from the comparisons that the maximum bound $b_{\max}$ is equal to

$$b_{\max} = \left\lceil \frac{L-2}{4} \right\rceil + 1. \tag{26}$$

Therefore, $p_{\text{opt}}(n,L) = \lceil x \rceil$, where x represents the solution of equation (27) verifying that $b_{\min} \leq \lceil x \rceil \leq b_{\max}$.

$$F_{n,L}(x) - t_{\text{opt}}(n,L) = 0. \tag{27}$$

Replacing each term by the corresponding expression, we obtain:

$$\frac{2x^3 - 3x^2 + x(12n + 6L^2 - 12nL - 5) + 3nL^2 + 3nL - 6n - 2L^3 - 4L + 6}{6x} = 0. \tag{28}$$

To compute the solution of equation (28), the trigonometric method of the cubic equation presented in [10] is used.

Theorem 3.2. *The theoretical lower bound for the number of processors $p_{\text{opt}}(n,L)$ required to execute the task graph in minimum time $t_{\text{opt}}(n,L) = (L-1)(2n-L)$ is expressed as:*

$$p_{\text{opt}}(n,L) = \left\lceil 2\sqrt{2nL - L^2 - 2n + \frac{13}{12}} \cos\left(\theta + \frac{4\pi}{3}\right) + \frac{1}{2} \right\rceil \tag{29}$$

where

$$\theta = \frac{1}{3} \arccos(\alpha) \tag{30}$$

and

$$\alpha = \frac{-3nL^2 - 3L^2 + 2L^3 + 3nL + 4L - 3}{4\left(2nL - L^2 - 2n + \frac{13}{12}\right)^{\frac{3}{2}}}. \tag{31}$$

Proof. Multiplying equation (28) by $6x$, we obtain:

$$2x^3 - 3x^2 + x(12n + 6L^2 - 12nL - 5) + 3nL^2 + 3nL - 6n - 2L^3 - 4L + 6 = 0.$$

By dividing the previous equation with 2 and performing the variable change $x = y + \frac{1}{2}$ in order to obtain a cubic equation that has no term in y^2.

$$y^3 - \left(6nL - 3L^2 - 6n + \frac{13}{4}\right) y + \frac{3nL^2 + 3L^2 - 2L^3 - 3nL - 4L + 3}{2} = 0.$$

Consider the substitution $y = 2\sqrt{2nL - L^2 - 2n + \frac{13}{12}} \cos\theta$ and the expression $4\cos^3\theta - 3\cos\theta = \cos(3\theta)$, the equation become:

$$2\left(2nL - L^2 - 2n + \frac{13}{12}\right)^{\frac{3}{2}} \cos(3\theta) + \frac{3nL^2 + 3L^2 - 2L^3 - 3nL - 4L + 3}{2} = 0.$$

Thus, $\cos(3\theta) = \alpha$ where α is defined in equation (31). The solution x of equation (28) is equal to

$$x = 2\sqrt{2nL - L^2 - 2n + \frac{13}{12}} \cos\left(\theta + \frac{4\pi}{3}\right) + \frac{1}{2}$$

where θ is given in equation (30).

The lower bound for the number of processors $p_{\text{opt}}(n,L) = \lceil x \rceil$. □

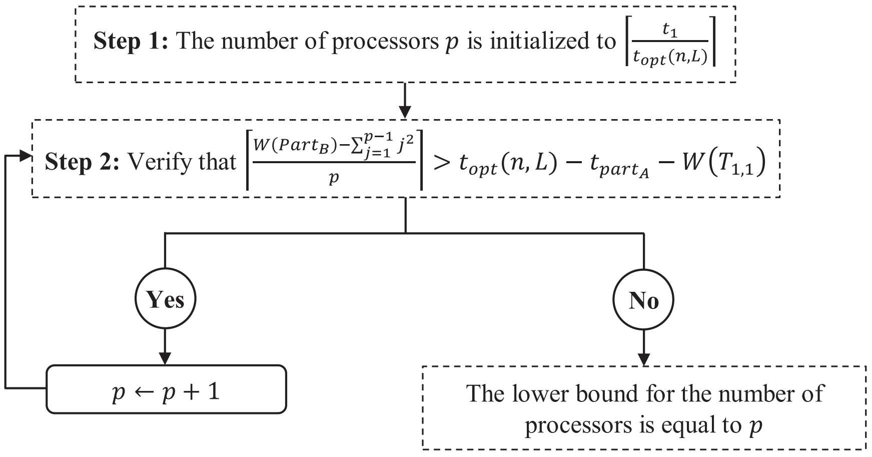
Step 1: The number of processors p is initialized to $\left\lceil \frac{t_1}{t_{opt}(n,L)} \right\rceil$
Step 2: Verify that $\left\lceil \frac{W(Part_B)-\sum_{j=1}^{p-1} j^2}{p} \right\rceil > t_{opt}(n,L) - t_{part_A} - W(T_{1,1})$
Yes
No
$p \leftarrow p+1$
The lower bound for the number of processors is equal to p

FIGURE 5. Determination of the lower bound for number of processors.

TABLE 2. Values of $P_{\text{opt}}(n, L)$ for some examples of SPD band matrices.

n	L	$p_{\text{opt}}(n, L)$	n	L	$p_{\text{opt}}(n, L)$
1000	200	49	1024	64	17
2000		50		128	32
3000		50		256	62
4000		51		384	90
5000		51		512	116
6000		51		640	138

To verify the correctness of the Theorem 3.2, we develop an algorithm that consists to determine the lower bound for the number of processors (see Fig. 5). At the beginning, p is initialized to $b_{\min} = \left\lceil \frac{t_1}{t_{\text{opt}}(n,L)} \right\rceil$. Subsequently, the two values indicated in equations (32) and (33) are compared.

$$\left\lceil \frac{W(part_B) - \sum_{j=1}^{p-1} j^2}{p} \right\rceil = \left\lceil \frac{3nL^2 - 2L^3 + 3nL - 4L - 6n + 6 - 4p^3 - 3p^2 + p}{6p} \right\rceil \tag{32}$$

$$t_{\text{opt}}(n, L) - t_{Part_A} - W(T_{1,1}) = (L-1)(2n - L - 1) - p^2. \tag{33}$$

The theoretical value of $p_{\text{opt}}(n, L)$ is validated experimentally by the algorithm shown in Figure 5. Table 2 gives some examples.

4. Practical proposed approach

This section describes our approach called parallel Gaussian elimination for symmetric positive definite band systems (PGE-SPDB). Firstly for each task, the earliest start time (EST)[1] and the latest start time (LST)[2]

[1]The EST of a task represents the earliest time at which it can begin if all its predecessors start as early as possible and are finished within their estimated times [31].

[2]The LST of a task represents the latest time at which it can begin without delaying the completion time for executing the precedence graph [31].

TABLE 3. Determination of the earliest start time (EST) for tasks.

Task type		Condition	Earliest start time of the task
Diagonal task		Entry task	$\text{EST}(T_{1,1}) = 0$
		$2 \leq k \leq n-1$	$\text{EST}(T_{k,k}) = \text{EST}(T_{k-1,k}) + W(T_{k-1,k})$
Non-diagonal task	With one predecessor	$k = 1$ and $2 \leq i \leq L$	$\text{EST}(T_{k,i}) = \text{EST}(T_{k,k}) + W(T_{k,k})$
		$2 \leq k \leq n-L+1$ and $i = k+L-1$	
	With two predecessors	$2 \leq k \leq n-1$ and $k+1 \leq i \leq \min(k+L-2, n)$	$\text{EST}(T_{k,i}) = \max(\mathit{value}\ 1, \mathit{value}\ 2)$ $\mathit{value}\ 1 = \text{EST}(T_{k,k}) + W(T_{k,k})$ $\mathit{value}\ 2 = \text{EST}(T_{k-1,i}) + W(T_{k-1,i})$

TABLE 4. Determination of the latest start time (LST) for tasks.

Task type	Condition	Latest start time of the task
Diagonal task	$1 \leq k \leq n-1$	$\text{LST}(T_{k,k}) = \text{LST}(T_{k,k+1}) - W(T_{k,k})$
Non-diagonal task	Final task	$\text{LST}(T_{n-1,n}) = t_{\text{opt}}(n, L) - W(T_{n-1,n})$
	$1 \leq k \leq n-2$ and $k+1 \leq i \leq ub$	$\text{LST}(T_{k,i}) = \text{LST}(T_{k+1,i}) - W(T_{k,i})$

obtained respectively from all its immediate predecessors and successors are computed. Formulas that consist to determine the EST and the LST for band structure are defined in Tables 3 and 4, respectively.

Secondly, tasks are arranged in ascending order of their LST. A variable *begin* is associated for each task $T_{u,v}$ and initialized from their predecessors $pred(s)$. Let $t_s(T_{u,v})$ denote an array that contains the start times of tasks. Four cases are distinguished and expressed by equation (34).

$$\mathit{begin} = \begin{cases} 0 & \text{for } T_{1,1} \\ t_s(T_{k-1,k}) + W(T_{k-1,k}) & \text{for } T_{k,k},\ 2 \leq k \leq n-1 \\ t_s(T_{k,k}) + W(T_{k,k}) & \text{for } T_{k,i} \text{ has 1 pred} \\ \max\{t_s(T_{k,k}) + W(T_{k,k}), t_s(T_{k-1,i}) + W(T_{k-1,i})\} & \text{for } T_{k,i} \text{ has 2 preds} \end{cases}. \tag{34}$$

After calculating the variable *begin* for the current task by applying the corresponding case in equation (34), a verifying step is needed to check the existence or the unavailability of a free slot time from *begin* to the instant $\mathit{begin} + W(T_{u,v}) - 1$. If the slot time is unsaturated then the task $T_{u,v}$ can start its execution at the instant *begin*. Otherwise, the variable *begin* is incremented by one. Step 2 is repeated until a free slot time is found for the current task. The algorithm that determines the adequate start time is depicted in Figure 6.

Then, the task list is sorted in ascending order of start time values. An available processor is assigned for each task using the algorithm shown in Figure 7 and saved in the array $Pr(T_{u,v})$. This phase consists to verify if an available processor exists from $t_s(T_{u,v})$ to $t_s(T_{u,v}) + W(T_{u,v}) - 1$. Firstly, the processor index j is initialized to one. If the current processor is not occupied thereby the task is assigned to it. Otherwise, the variable j is incremented by one. Step 1 is repeated until an available processor is found for the current task.

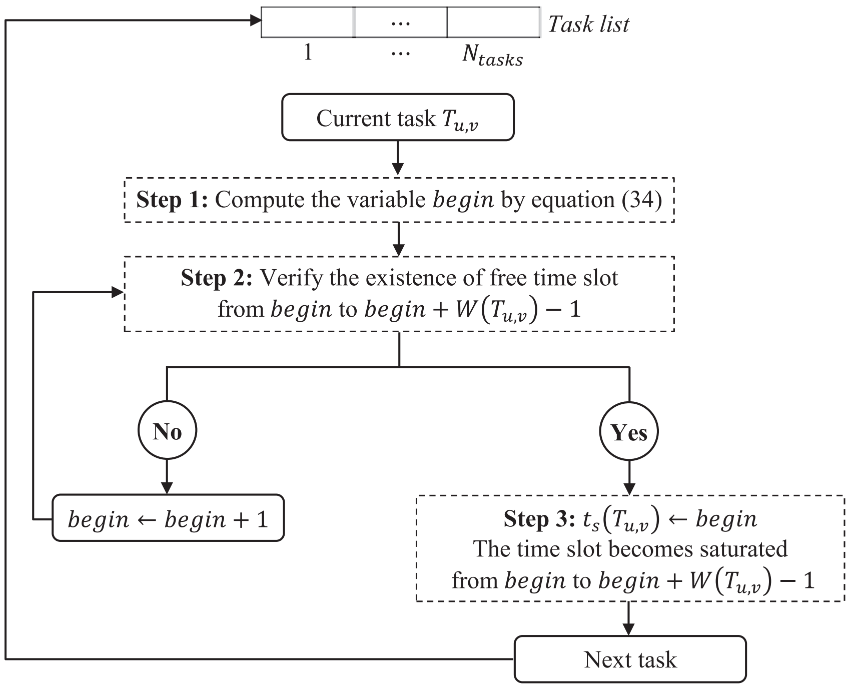
Task list
1
⋯
N_{tasks}
Current task $T_{u,v}$
Step 1: Compute the variable $begin$ by equation (34)
Step 2: Verify the existence of free time slot from $begin$ to $begin + W(T_{u,v}) - 1$
No
Yes
$begin \leftarrow begin + 1$
Step 3: $t_s(T_{u,v}) \leftarrow begin$ The time slot becomes saturated from $begin$ to $begin + W(T_{u,v}) - 1$
Next task

FIGURE 6. Determination phase of start times for all tasks.

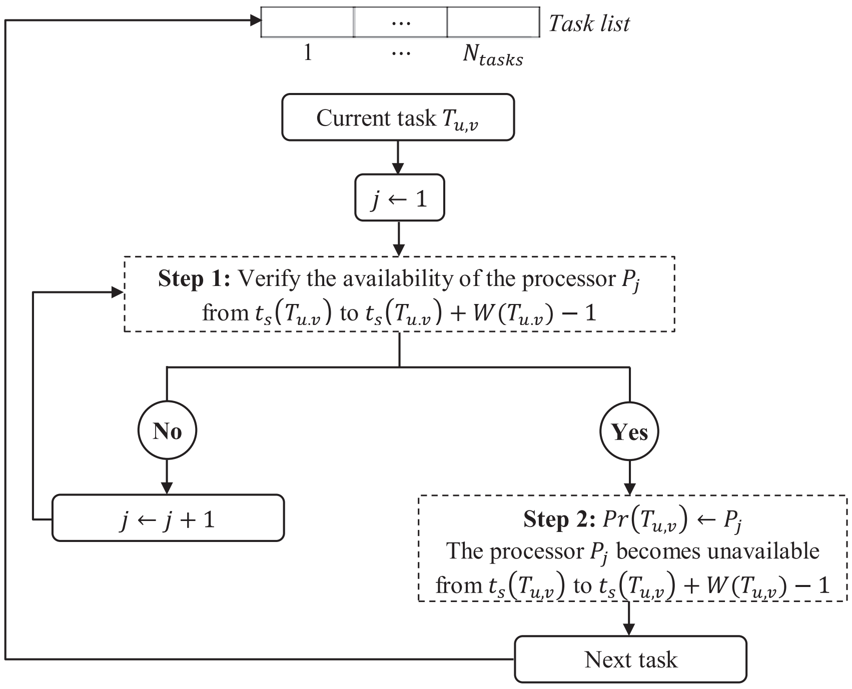
Task list
1
⋯
N_{tasks}
Current task $T_{u,v}$
$j \leftarrow 1$
Step 1: Verify the availability of the processor P_j from $t_s(T_{u,v})$ to $t_s(T_{u,v}) + W(T_{u,v}) - 1$
No
Yes
$j \leftarrow j + 1$
Step 2: $Pr(T_{u,v}) \leftarrow P_j$ The processor P_j becomes unavailable from $t_s(T_{u,v})$ to $t_s(T_{u,v}) + W(T_{u,v}) - 1$
Next task

FIGURE 7. Determination phase of processors for all tasks.

TABLE 5. List of start times and processors numbers for tasks where $n = 12$, $L = 8$ and $p = 3$.

$T_{u,v}$	$\{t_s, Pr\}$	$T_{u,v}$	$\{t_s, Pr\}$	$T_{u,v}$	$\{t_s, Pr\}$
$T_{1,1}$	$\{0, P_1\}$	$T_{3,10}$	$\{44, P_3\}$	$T_{6,12}$	$\{83, P_2\}$
$T_{1,2}$	$\{7, P_1\}$	$T_{4,4}$	$\{42, P_1\}$	$T_{7,7}$	$\{82, P_1\}$
$T_{1,3}$	$\{7, P_2\}$	$T_{4,5}$	$\{49, P_1\}$	$T_{7,8}$	$\{87, P_1\}$
$T_{1,4}$	$\{7, P_3\}$	$T_{4,6}$	$\{49, P_2\}$	$T_{7,9}$	$\{87, P_2\}$
$T_{1,5}$	$\{12, P_3\}$	$T_{4,7}$	$\{49, P_3\}$	$T_{7,10}$	$\{87, P_3\}$
$T_{1,6}$	$\{13, P_2\}$	$T_{4,8}$	$\{54, P_3\}$	$T_{7,11}$	$\{90, P_3\}$
$T_{1,7}$	$\{16, P_2\}$	$T_{4,9}$	$\{55, P_2\}$	$T_{7,12}$	$\{91, P_2\}$
$T_{1,8}$	$\{16, P_3\}$	$T_{4,10}$	$\{58, P_2\}$	$T_{8,8}$	$\{92, P_1\}$
$T_{2,2}$	$\{14, P_1\}$	$T_{4,11}$	$\{58, P_3\}$	$T_{8,9}$	$\{96, P_1\}$
$T_{2,3}$	$\{21, P_1\}$	$T_{5,5}$	$\{56, P_1\}$	$T_{8,10}$	$\{96, P_2\}$
$T_{2,4}$	$\{21, P_2\}$	$T_{5,6}$	$\{63, P_1\}$	$T_{8,11}$	$\{96, P_3\}$
$T_{2,5}$	$\{21, P_3\}$	$T_{5,7}$	$\{63, P_2\}$	$T_{8,12}$	$\{98, P_3\}$
$T_{2,6}$	$\{26, P_3\}$	$T_{5,8}$	$\{63, P_3\}$	$T_{9,9}$	$\{100, P_1\}$
$T_{2,7}$	$\{27, P_2\}$	$T_{5,9}$	$\{68, P_3\}$	$T_{9,10}$	$\{103, P_1\}$
$T_{2,8}$	$\{30, P_2\}$	$T_{5,10}$	$\{69, P_2\}$	$T_{9,11}$	$\{103, P_2\}$
$T_{2,9}$	$\{30, P_3\}$	$T_{5,11}$	$\{72, P_2\}$	$T_{9,12}$	$\{103, P_3\}$
$T_{3,3}$	$\{28, P_1\}$	$T_{5,12}$	$\{72, P_3\}$	$T_{10,10}$	$\{106, P_1\}$
$T_{3,4}$	$\{35, P_1\}$	$T_{6,6}$	$\{70, P_1\}$	$T_{10,11}$	$\{108, P_1\}$
$T_{3,5}$	$\{35, P_2\}$	$T_{6,7}$	$\{76, P_1\}$	$T_{10,12}$	$\{108, P_2\}$
$T_{3,6}$	$\{35, P_3\}$	$T_{6,8}$	$\{76, P_2\}$	$T_{11,11}$	$\{110, P_1\}$
$T_{3,7}$	$\{40, P_3\}$	$T_{6,9}$	$\{76, P_3\}$	$T_{11,12}$	$\{111, P_1\}$
$T_{3,8}$	$\{41, P_2\}$	$T_{6,10}$	$\{80, P_3\}$		
$T_{3,9}$	$\{44, P_2\}$	$T_{6,11}$	$\{81, P_2\}$		

The schedule length is obtained from the finish time of the last task in the precedence graph.

$$t_p = t_s(T_{n-1,n}) + W(T_{n-1,n}) \tag{35}$$

To better understand these algorithms, Table 5 contains the start times $t_s(T_{u,v})$ and the processors numbers $Pr(T_{u,v})$ corresponding to all tasks $T_{u,v}$ of the precedence graph illustrated in Figure 3 using three processors where $1 \le u \le n-1$ and $u \le v \le \min(k+L-1, n)$. For $n = 12$, $L = 8$ and $p = 3$, the schedule length is equal to $t_3 = 112$.

To profit in execution time using p processors and obtain a high degree of parallelism, eliminating the useless dependencies is required. A dependency constraint is unnecessary for tasks allocated to the same processor because they will naturally perform in schedule order. In addition, tasks depend on the same diagonal task and assigned to the identical processor do not need a precedence constraint only for the first task. For instance, $T_{2,2} \ll T_{2,5}$ means that the task $T_{2,5}$ can start its execution only if $T_{2,2}$ is realized. $T_{2,2} \ll T_{2,6}$ and $T_{2,2} \ll T_{2,9}$ are redundant because $T_{2,6}$ and $T_{2,9}$ are allocated to P_3 (the same processor of $T_{2,5}$).

Figures 8 and 9 present the algorithms for eliminating the dependencies $T_{k,k} \ll T_{k,i}$ and $T_{k,i} \ll T_{k+1,i}$, respectively. The dependency $T_{k,k} \ll T_{k,i}$ is removed in two cases.

- If the tasks $T_{k,k}$ and $T_{k,i}$ are attributed to the same processor.
- If the task $T_{k,i}$ is not the first task with row index k assigned to a processor.

If the tasks $T_{k,i}$ and $T_{k+1,i}$ are attributed to the same processor, then the dependency $T_{k,i} \ll T_{k+1,i}$ is removed.

Table 6 outlines all the dependencies corresponding to the precedence graph illustrated in Figure 3. For $n = 12$, $L = 8$ and $p = 3$, each eliminated dependency is mentioned in the table by the letter E.

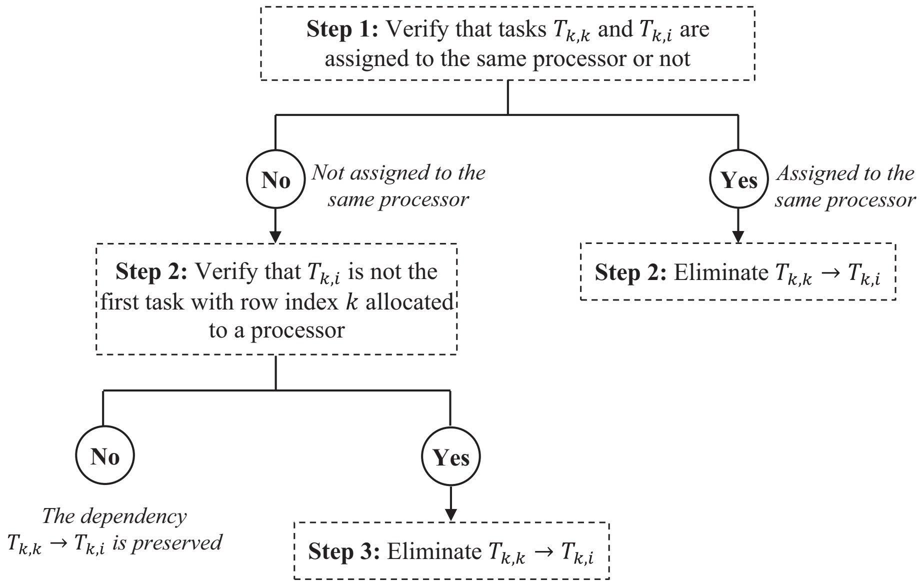
Step 1: Verify that tasks $T_{k,k}$ and $T_{k,i}$ are assigned to the same processor or not
No
Not assigned to the same processor
Yes
Assigned to the same processor
Step 2: Verify that $T_{k,i}$ is not the first task with row index k allocated to a processor
Step 2: Eliminate $T_{k,k} \rightarrow T_{k,i}$
No
Yes
The dependency $T_{k,k} \rightarrow T_{k,i}$ is preserved
Step 3: Eliminate $T_{k,k} \rightarrow T_{k,i}$

FIGURE 8. Elimination of the unnecessary dependencies in the form $T_{k,k} \ll T_{k,i}$.

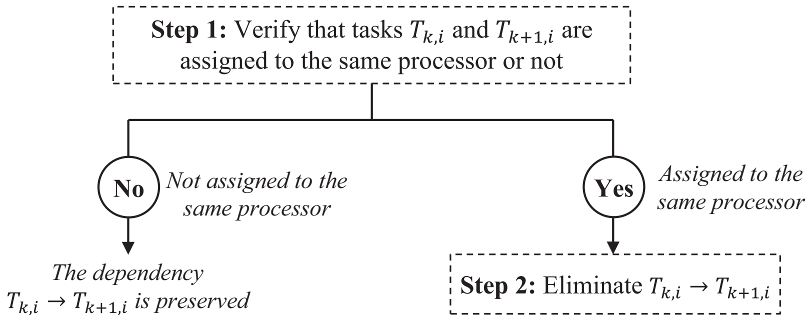
Step 1: Verify that tasks $T_{k,i}$ and $T_{k+1,i}$ are assigned to the same processor or not
No
Not assigned to the same processor
Yes
Assigned to the same processor
The dependency $T_{k,i} \rightarrow T_{k+1,i}$ is preserved
Step 2: Eliminate $T_{k,i} \rightarrow T_{k+1,i}$

FIGURE 9. Elimination of the unnecessary dependencies in the form $T_{k,i} \ll T_{k+1,i}$.

In this example, there are 111 dependencies inter-tasks. By applying algorithms illustrated in Figures 8 and 9 and using three processors, precisely 61 dependencies will be eliminated (*i.e.*, approximately 54.95% of the total number of edges). The corresponding Gantt chart is shown in Figure 10.

Simultaneously, all processors perform their associated tasks respecting the necessary dependencies. Let $Finish(T_{u,v})$ represents the array that contains the tasks' status (1 or 0 means respectively that $T_{u,v}$ is carried out or not yet executed). In addition, a task can start if their predecessors' status is set to one. The parallel processing phase is illustrated in Figure 11.

For the example illustrated in Figure 10, we have the following scenario: the processor P_1 executes the task $T_{1,1}$ while P_2 and P_3 wait for the required data because tasks $T_{1,3}$ and $T_{1,4}$ depend to $T_{1,1}$ (see Tab. 6). Once the execution of task $T_{1,1}$ is achieved which is indicated by $Finish(T_{1,1}) = 1$, the processors P_1, P_2 and P_3 perform respectively the tasks $T_{1,2}$, $T_{1,3}$ and $T_{1,4}$. The processors continue to carry out the tasks assigned to them taking into account the dependencies.

TABLE 6. List of dependencies and their status.

Dependencies	status	Dependencies	status	Dependencies	status
$T_{1,1} \rightarrow T_{1,2}$	E	$T_{3,6} \rightarrow T_{4,6}$		$T_{6,6} \rightarrow T_{6,11}$	E
$T_{1,1} \rightarrow T_{1,3}$		$T_{3,7} \rightarrow T_{4,7}$	E	$T_{6,6} \rightarrow T_{6,12}$	E
$T_{1,1} \rightarrow T_{1,4}$		$T_{3,8} \rightarrow T_{4,8}$		$T_{6,7} \rightarrow T_{7,7}$	E
$T_{1,1} \rightarrow T_{1,5}$	E	$T_{3,9} \rightarrow T_{4,9}$	E	$T_{6,8} \rightarrow T_{7,8}$	
$T_{1,1} \rightarrow T_{1,6}$	E	$T_{3,10} \rightarrow T_{4,10}$		$T_{6,9} \rightarrow T_{7,9}$	
$T_{1,1} \rightarrow T_{1,7}$	E	$T_{4,4} \rightarrow T_{4,5}$	E	$T_{6,10} \rightarrow T_{7,10}$	E
$T_{1,1} \rightarrow T_{1,8}$	E	$T_{4,4} \rightarrow T_{4,6}$		$T_{6,11} \rightarrow T_{7,11}$	
$T_{1,2} \rightarrow T_{2,2}$	E	$T_{4,4} \rightarrow T_{4,7}$		$T_{6,12} \rightarrow T_{7,12}$	E
$T_{1,3} \rightarrow T_{2,3}$		$T_{4,4} \rightarrow T_{4,8}$	E	$T_{7,7} \rightarrow T_{7,8}$	E
$T_{1,4} \rightarrow T_{2,4}$		$T_{4,4} \rightarrow T_{4,9}$	E	$T_{7,7} \rightarrow T_{7,9}$	
$T_{1,5} \rightarrow T_{2,5}$	E	$T_{4,4} \rightarrow T_{4,10}$	E	$T_{7,7} \rightarrow T_{7,10}$	
$T_{1,6} \rightarrow T_{2,6}$		$T_{4,4} \rightarrow T_{4,11}$	E	$T_{7,7} \rightarrow T_{7,11}$	E
$T_{1,7} \rightarrow T_{2,7}$	E	$T_{4,5} \rightarrow T_{5,5}$	E	$T_{7,7} \rightarrow T_{7,12}$	E
$T_{1,8} \rightarrow T_{2,8}$		$T_{4,6} \rightarrow T_{5,6}$		$T_{7,8} \rightarrow T_{8,8}$	E
$T_{2,2} \rightarrow T_{2,3}$	E	$T_{4,7} \rightarrow T_{5,7}$		$T_{7,9} \rightarrow T_{8,9}$	
$T_{2,2} \rightarrow T_{2,4}$		$T_{4,8} \rightarrow T_{5,8}$	E	$T_{7,10} \rightarrow T_{8,10}$	
$T_{2,2} \rightarrow T_{2,5}$		$T_{4,9} \rightarrow T_{5,9}$		$T_{7,11} \rightarrow T_{8,11}$	E
$T_{2,2} \rightarrow T_{2,6}$	E	$T_{4,10} \rightarrow T_{5,10}$	E	$T_{7,12} \rightarrow T_{8,12}$	
$T_{2,2} \rightarrow T_{2,7}$	E	$T_{4,11} \rightarrow T_{5,11}$		$T_{8,8} \rightarrow T_{8,9}$	E
$T_{2,2} \rightarrow T_{2,8}$	E	$T_{5,5} \rightarrow T_{5,6}$	E	$T_{8,8} \rightarrow T_{8,10}$	
$T_{2,2} \rightarrow T_{2,9}$	E	$T_{5,5} \rightarrow T_{5,7}$		$T_{8,8} \rightarrow T_{8,11}$	
$T_{2,3} \rightarrow T_{3,3}$	E	$T_{5,5} \rightarrow T_{5,8}$		$T_{8,8} \rightarrow T_{8,12}$	E
$T_{2,4} \rightarrow T_{3,4}$		$T_{5,5} \rightarrow T_{5,9}$	E	$T_{8,9} \rightarrow T_{9,9}$	E
$T_{2,5} \rightarrow T_{3,5}$		$T_{5,5} \rightarrow T_{5,10}$	E	$T_{8,10} \rightarrow T_{9,10}$	
$T_{2,6} \rightarrow T_{3,6}$	E	$T_{5,5} \rightarrow T_{5,11}$	E	$T_{8,11} \rightarrow T_{9,11}$	
$T_{2,7} \rightarrow T_{3,7}$		$T_{5,5} \rightarrow T_{5,12}$	E	$T_{8,12} \rightarrow T_{9,12}$	E
$T_{2,8} \rightarrow T_{3,8}$	E	$T_{5,6} \rightarrow T_{6,6}$	E	$T_{9,9} \rightarrow T_{9,10}$	E
$T_{2,9} \rightarrow T_{3,9}$		$T_{5,7} \rightarrow T_{6,7}$		$T_{9,9} \rightarrow T_{9,11}$	
$T_{3,3} \rightarrow T_{3,4}$	E	$T_{5,8} \rightarrow T_{6,8}$		$T_{9,9} \rightarrow T_{9,12}$	
$T_{3,3} \rightarrow T_{3,5}$		$T_{5,9} \rightarrow T_{6,9}$	E	$T_{9,10} \rightarrow T_{10,10}$	E
$T_{3,3} \rightarrow T_{3,6}$		$T_{5,10} \rightarrow T_{6,10}$		$T_{9,11} \rightarrow T_{10,11}$	
$T_{3,3} \rightarrow T_{3,7}$	E	$T_{5,11} \rightarrow T_{6,11}$	E	$T_{9,12} \rightarrow T_{10,12}$	
$T_{3,3} \rightarrow T_{3,8}$	E	$T_{5,12} \rightarrow T_{6,12}$		$T_{10,10} \rightarrow T_{10,11}$	E
$T_{3,3} \rightarrow T_{3,9}$	E	$T_{6,6} \rightarrow T_{6,7}$	E	$T_{10,10} \rightarrow T_{10,12}$	
$T_{3,3} \rightarrow T_{3,10}$	E	$T_{6,6} \rightarrow T_{6,8}$		$T_{10,11} \rightarrow T_{11,11}$	E
$T_{3,4} \rightarrow T_{4,4}$	E	$T_{6,6} \rightarrow T_{6,9}$		$T_{10,12} \rightarrow T_{11,12}$	
$T_{3,5} \rightarrow T_{4,5}$		$T_{6,6} \rightarrow T_{6,10}$	E	$T_{11,11} \rightarrow T_{11,12}$	E

5. Experimental results and discussion

In order to evaluate and validate the effectiveness of the proposed approach, the performances of PGE-SPDB and those of determined theoretical formulas and row block method are compared.

5.1. Setup

The experiments are performed on a multicore processor composed of 8 cores Intel Xeon E5-2660 running at 2.2 GHz which is reserved in the platform Grid'5000[3] [15] and belongs to the cluster Econome in the Nantes site.

[3]Grid'5000 is a large-scale testbed for experimentations with a focus on high-performance parallel computing (https://www.grid5000.fr).

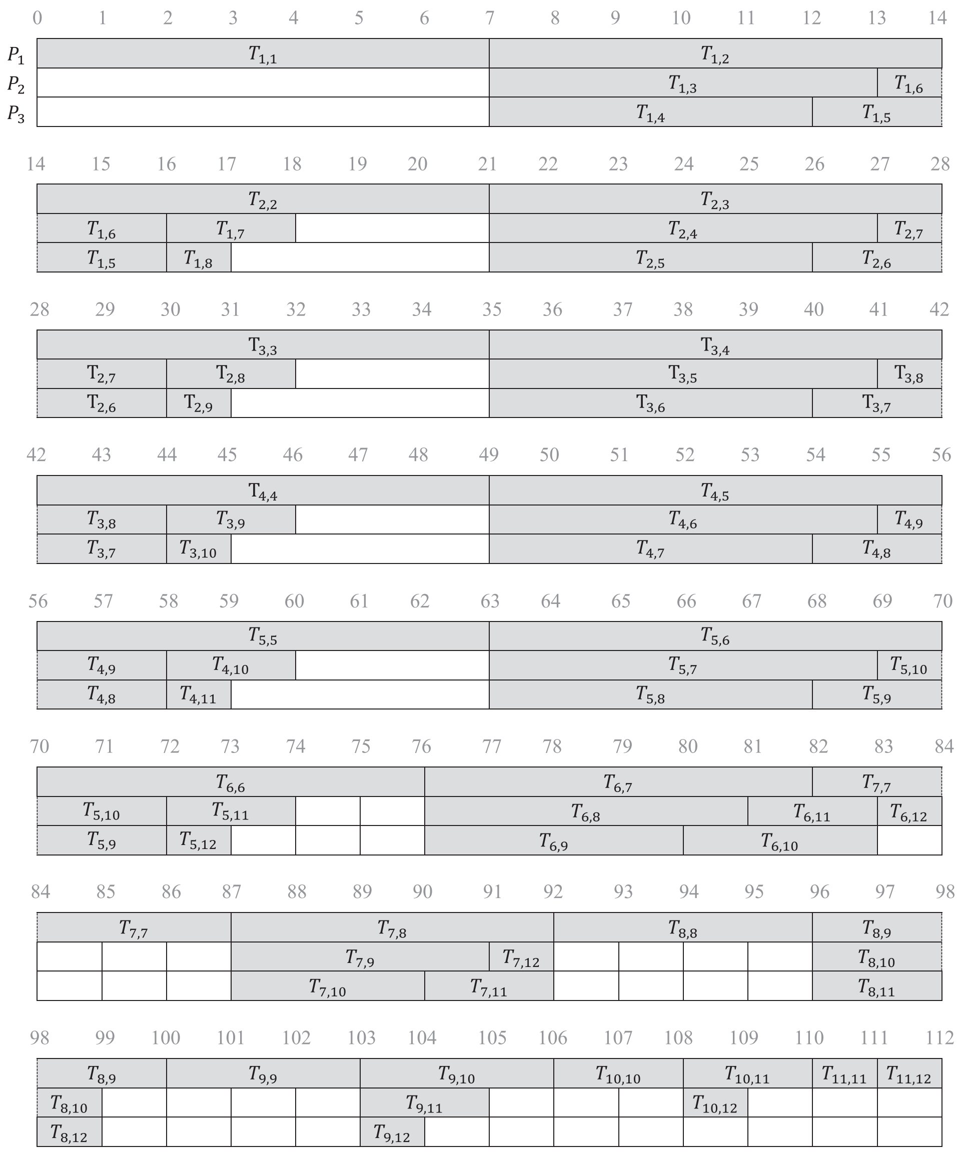

FIGURE 10. Gantt chart of the precedence graph illustrated in Figure 3 using three processors.

Actually, clusters of multicore nodes based on different architectures have become the most used option for new High-Performance Computing systems due to their performance/cost ratio and scalability. All programs were implemented in C programming language using OpenMP[4] interface for the parallel regions. A set of randomly generated SPD band matrices listed in Table 7 are used by varying the bandwidth L and the matrix size n. For all matrices, number of tasks N_{tasks}, number of dependencies N_{edges}, sequential runtime t_1 and optimal time $t_{\text{opt}}(n, L)$ are calculated using the equations (5), (6), (7) and (8), respectively.

[4]The Open Multi-Processing (OpenMP) represents an Application Programming Interface (API) which aims to enable portable shared memory parallel programming [6].

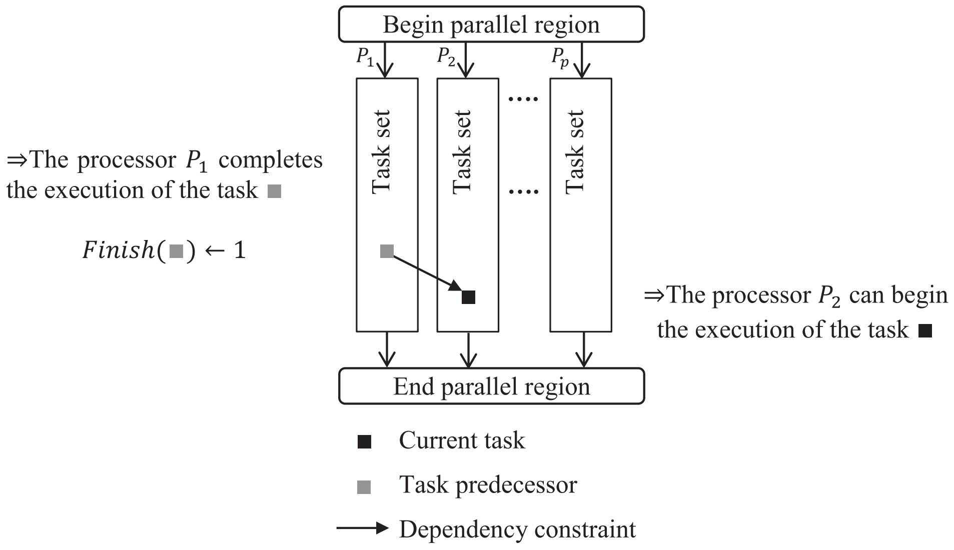

FIGURE 11. Parallel processing phase.

TABLE 7. Properties of test matrices.

n	L	N_{tasks}	N_{edges}	t_1	$t_{\text{opt}}(n, L)$
	200	180 099	358 199	17 432 400	358 200
1000	400	320 199	638 399	58 865 800	638 400
	600	420 299	838 599	108 299 200	838 600
1500		343 874	684 749	41 852 750	684 750
2500	250	593 874	1 182 749	73 226 750	1 182 750
3500		843 874	1 680 749	104 600 750	1 680 750

TABLE 8. Results of tested matrices using the theoretical formulas.

n	L	$t_{\text{opt},2}$	$t_{\text{opt},4}$	$t_{\text{opt},8}$	p_{opt}
	200	8 716 300	4 358 253	2 179 242	49
1000	400	29 433 100	14 716 753	7 358 592	93
	600	54 149 900	27 075 253	13 537 942	131
1500		20 926 500	10 463 378	5 231 830	62
2500	250	36 613 500	18 306 878	9 153 580	62
3500		52 300 500	26 150 378	13 075 330	63

5.2. Comparison with theoretical formulas

Table 8 gives the theoretical lower bound for parallel execution time $t_{\text{opt},p}(n, L)$ where $p = \{2, 4, 8\}$ and number of processors $p_{\text{opt}}(n, L)$ which are calculated using the equations (20) and (29), respectively. The PGE-SPDB schedule length t_p where $p = \{2, 4, 8, p_{\text{opt}}\}$ is computed by the algorithm shown in Figure 6 and equation (35). Given $p_{\text{PGE-SPDB}}$ processors, the schedule length t_p attains the optimal time $t_{\text{opt}}(n, L)$. These data are presented in Table 9.

TABLE 9. Results of tested matrices using the PGE-SPDB approach.

n	L	t_2	t_4	t_8	$t_{p_{\text{opt}}}$	$p_{\text{PGE-SPDB}}$
	200	8 716 300	4 358 253	2 179 242	358 231	50
1000	400	29 433 100	14 716 753	7 358 592	639 389	94
	600	54 149 900	27 075 253	13 537 942	840 814	132
1500		20 926 500	10 463 378	5 231 830	684 750	62
2500	250	36 613 500	18 306 878	9 153 580	1 186 361	63
3500		52 300 500	26 150 378	13 075 330	1 680 750	63

TABLE 10. Comparison between experimental and theoretical results.

n	L	$\delta_t(t_{p_{\text{opt}}}, t_{\text{opt}})$	np	n	L	$\delta_t(t_{p_{\text{opt}}}, t_{\text{opt}})$	np
	200	31	1	1500		0	0
1000	400	989	1	2500	250	3611	1
	600	2214	1	2500		0	0

Tables 7–9 indicate that t_1, $t_{\text{opt},p}(n, L)$, $t_{\text{opt}}(n, L)$ and t_p increase as the bandwidth or the matrix size increases. This result can be justified by the rise in the number of tasks and dependencies. Let $\delta_t(t_a, t_b)$ symbolize the time difference between the experimental and theoretical execution times noted respectively t_a and t_b where $t_a \geq t_b$. It is expressed by

$$\delta_t(t_a, t_b) = \begin{cases} \delta_t\big(t_p(n, L), t_{\text{opt},p}(n, L)\big) \geq 0 & \text{if } p < p_{\text{opt}}(n, L) \\ \delta_t\big(t_p(n, L), t_{\text{opt}}(n, L)\big) \geq 0 & \text{if } p = p_{\text{opt}}(n, L) \end{cases} \tag{36}$$

$\delta_t(t_a, t_b)$ varies according to the matrix size n, the bandwidth L and the number of processors p. Two cases are distinguished:

- $\delta_t(t_a, t_b) = 0$ imply that the experimental and theoretical execution times are equal.
- $\delta_t(t_a, t_b) > 0$ signify that it exists a small difference between the values of t_a and t_b. This phase shift occurs because the tasks do not have the same cost.

Let the variable np denote the difference between the two numbers $p_{\text{PGE-SPDB}}(n, L)$ and $p_{\text{opt}}(n, L)$ where $p_{\text{PGE-SPDB}}(n, L) \geq p_{\text{opt}}(n, L)$.

- If $np = 0$, then p_{opt} processors are sufficient to schedule the PGE-SPDB approach in optimal time $t_{\text{opt}}(n, L)$.
- Otherwise, more processors' number is needed and equal to np.

The difference between times $\delta_t(t_2, t_{\text{opt},2})$, $\delta_t(t_4, t_{\text{opt},4})$, and $\delta_t(t_8, t_{\text{opt},8})$ for the given matrices are equal to zero. The values of $\delta_t(t_{p_{\text{opt}}}, t_{\text{opt}})$ and np are summarized in Table 10.

The numerical results show that the PGE-SPDB method is effective because the experimental values are close to the theoretical ones.

5.3. Comparison with row block method

This section presents and discusses the experimental results obtained by the methods PGE-SPDB and row block.

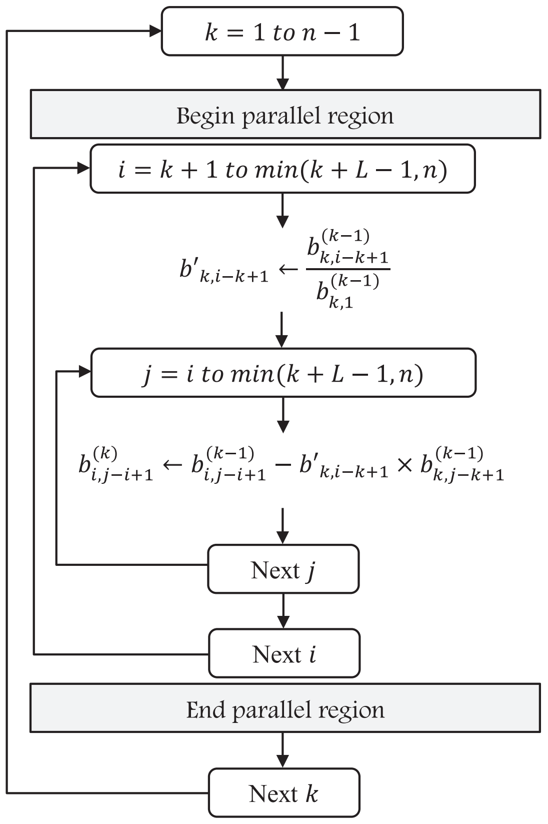
$k = 1 \; to \; n-1$
Begin parallel region
$i = k+1 \; to \; min(k+L-1, n)$
$b'_{k,i-k+1} \leftarrow \frac{b^{(k-1)}_{k,i-k+1}}{b^{(k-1)}_{k,1}}$
$j = i \; to \; min(k+L-1, n)$
$b^{(k)}_{i,j-i+1} \leftarrow b^{(k-1)}_{i,j-i+1} - b'_{k,i-k+1} \times b^{(k-1)}_{k,j-k+1}$
Next j
Next i
End parallel region
Next k

FIGURE 12. Row block method for SPD band matrices.

5.3.1. *Row block method*

McGinn and Shaw [23] developed for dense matrices a parallel algorithm for GE in a shared memory environment using OpenMP with different scheduling strategies specified by the schedule clause. Using the static scheme without a specified chunk size implies that OpenMP splits iterations into p blocks and statically assigned to cores in a block-wise distribution. Refer to [23], Figure 12 outlines the parallel algorithm applied for SPD band matrices using BND format. This algorithm uses the row block data distribution.

5.3.2. *Results and analysis*

To compare the performance of the proposed approach (PGE-SPDB) with the row block method, the efficiency metric E_p is calculated. It is defined as:

$$E_p = \frac{t_{\text{seq}}}{t_{\text{par}} \times p} \tag{37}$$

where t_{par} represents the parallel runtime running on p cores and t_{seq} the sequential execution time measured by a single core [19, 26]. The efficiencies measurements are summarized in Figures 13 and 14.

From the resulting histograms, it can be seen that:

(i) The developed approach (PGE-SPDB) is a significant improvement of parallel GE compared to the row block method as it can effectively reduce the parallel execution time for all matrices.
(ii) The PGE-SPDB efficiency increases slightly as the bandwidth increases for fixed matrix size n and a number of cores p (see Fig. 13).

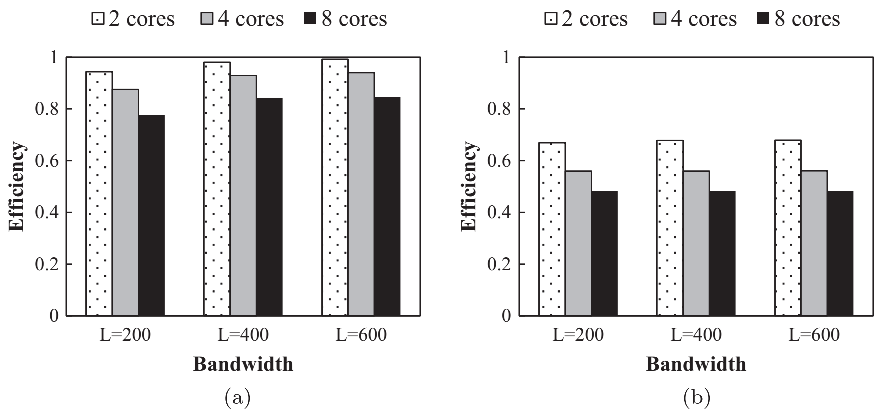

FIGURE 13. Efficiencies measurements for fixed matrix size $n = 1000$. (a) PGE-SPDB and (b) Row block.

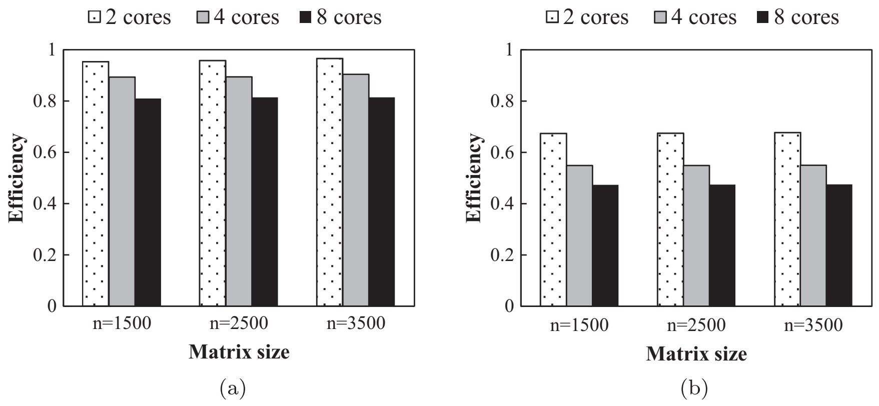

FIGURE 14. Efficiencies measurements for fixed bandwidth $L = 250$. (a) PGE-SPDB and (b) Row block.

(iii) A slight variation is noted in PGE-SPDB efficiency by varying the matrix size for fixed bandwidth L and a number of cores p (see Fig. 14).
(iv) The efficiency decreases with each additional core. This is related to the growth of the total idle time of cores.
(v) In PGE-SPDB method, all cores treated simultaneously their associated tasks. A task starts execution if its predecessors have been performed. However, in row block method, all cores must synchronize after each iteration k where $1 \le k \le n-1$.
(vi) In PGE-SPDB approach, there is a load balancing between cores. Whereas in row block method, the partition of the tasks according to their weight among cores is not equal.
(vii) The overhead of synchronization and the no balancing into weight of tasks between cores lead to an increase of total idle time. Therefore the row block method's efficiency becomes relatively low.

The numerical experiments performed on a multicore processor proved the efficiency of our investigation. The high parallelism degree and the shortest schedule length were obtained by PGE-SPDB are due to:

- Balancing the load among cores.

- Assigning for each task the adequate start time and processor yielding to minimize parallel execution time and increase the activity time of cores thereby reduce their idle time.
- Eliminating the useless dependencies leads to a decrease in the waiting time and speed up the execution of the precedence graph.

6. Conclusion

In this study, a new approach of parallel GE for SPD band matrices is developed. Summing up the results, it can be concluded that the PGE-SPDB approach can achieve a higher degree of parallelism by reducing the overall idle and parallel execution times and balancing the load between processors. This can be done by appropriately determining the start time and available processor to each task and eliminating the unnecessary dependencies inter-tasks. In addition, we have calculated the lower bound of parallel execution time and number of processors. The experimental results acquired from the PGE-SPDB approach are fitted to theoretical values. It is obvious that several other questions remain to be addressed. In our future research, we intend to implement the proposed method using different matrix structures and other parallel architectures such as Graphics Processing Unit (GPU) accelerated multicore systems.

References

[1] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe and H. van der Vorst, Editors, Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide. Society for Industrial and Applied Mathematics, Philadelphia, PA (2000).

[2] E. Bampis, J.C. Konig and D. Trystram, Impact of communications on the complexity of the parallel Gaussian elimination. *Parallel Comput.* **17** (1991) 55–61.

[3] Å. Björck, Numerical Methods in Matrix Computations. In: Vol. 59 of *Texts in Applied Mathematics*, edited by: J. Bell, R. Kohn, P. Newton, C. Peskin, R. Pego, L. Ryzhik, A. Singer, A. Stevens, A. Stuart, T. Witelski, S. Wright. Springer, New York, NY (2015).

[4] R. Butt, Introduction to Numerical Analysis Using MATLAB. Jones and Bartlett Publishers, Burlington, MA (2010).

[5] T.R. Chandrupatla and A.D. Belegundu, Introduction to Finite Elements in Engineering, 4th edition. Pearson Education, London (2012).

[6] B. Chapman, G. Jost and R. Van Der Pas, Using OpenMP: Portable Shared Memory Parallel Programming. MIT Press, Cambridge, MA (2008).

[7] Q. Chen and M. Guo, Task Scheduling for Multi-core and Parallel Architectures: Challenges, Solutions and Perspectives. Springer Singapore, Singapore (2017).

[8] F.H. Chishti, A.R. Clare and M. Razaz, Gaussian elimination of symmetric, positive definite, banded systems on transputer networks. In: Transputer/Occam Japan 4. Proceedings of the 4th Transputer/Occam International Conference. IOS Press, Amsterdam (1992) 73–84.

[9] R.M. Corless and N. Fillion, A Graduate Introduction to Numerical Methods: From the Viewpoint of Backward Error Analysis. Springer, New York, NY (2013).

[10] D.A. Cox, Galois Theory. John Wiley & Sons, Hoboken, NJ (2004).

[11] M. Drozdowski, *Scheduling for Parallel Processing.* Springer, London (2009).

[12] W. Gander, M.J. Gander and F. Kwok, Scientific Computing – An Introduction Using Maple and MATLAB. In: Vol. 11 of *Texts in Computational Science and Engineering*, edited by: T.J. Barth, M. Griebel, D.E. Keyes, R.M. Nieminen, D. Roose, T. Schlick. Springer, New York, NY (2014).

[13] G.H. Golub and C.F. Van Loan, Matrix Computations, 4th edition. Johns Hopkins University Press, Baltimore, MA (2013).

[14] A. Greenbaum and T.P. Chartier, Numerical Methods: Design, Analysis, and Computer Implementation of Algorithms. Princeton University Press, Princeton, NJ (2012).

[15] Grid5000, https://www.grid5000.fr.

[16] M. Hakem and F. Butelle, Critical path scheduling parallel programs on an unbounded number of processors. *Int. J. Found. Comput. Sci.* **17** (2006) 287–301.

[17] D.A. Harville, Matrix Algebra from a Statistician's Perspective. Springer, New York, NY (1997).

[18] A. Kavcic and J.M.F. Moura, Matrices with banded inverses: inversion algorithms and factorization of Gauss–Markov processes. *IEEE Trans. Inf. Theory* **46** (2000) 1495–1509.

[19] J. Kwiatkowski, Parallel applications performance evaluation using the concept of granularity. In: Vol. 8385 of *Lecture Notes in Computer Science. Parallel Processing and Applied Mathematics. Proceedings of the International Conference PPAM 2013*, edited by R. Wyrzykowski, J. Dongarra, K. Karczewski, J. Waśniewski. Springer, New York, NY (2014) 215–224.

[20] M. Marrakchi and Y. Robert, Optimal algorithms for Gaussian elimination on an MIMD computer. *Parallel Comput.* **12** (1989) 183–194.

[21] S. Marrakchi and M. Jemni, Fine-grained parallel solution for solving sparse triangular systems on multicore platform using OpenMP interface. In: 2017 International Conference on High Performance Computing Simulation (HPCS). IEEE, New York, NY (2017) 659–666.

[22] S. Marrakchi and M. Jemni, A parallel scheduling algorithm to solve triangular band systems on multicore machine. In: Vol. 32 of *Advances in Parallel Computing. Parallel Computing is Everywhere. Proceedings of the International Conference on Parallel Computing. ParCo 2017*, edited by: S. Bassini, M. Danelutto, P. Dazzi, G.R. Joubert, F. Peters. IOS Press, Amsterdam (2018) 127–136.

[23] S.F. McGinn and R.E. Shaw, Parallel Gaussian elimination using OpenMP and MPI. In: Proceedings 16th Annual International Symposium on High Performance Computing Systems and Applications. IEEE, New York, NY (2002) 169–173.

[24] G. Meurant, Gaussian elimination for the solution of linear systems of equations. In: Vol. 7 of Handbook of Numerical Analysis. Elsevier, New York, NY (2000) 3–170.

[25] I.Ž. Milovanović, E.I. Milavanović and M.K. Stojčev, An optimal algorithm for Gaussian elimination of band matrices on an MIMD computer. *Parallel Comput.* **15** (1990) 133–145.

[26] A. Munir, A. Gordon-Ross and S. Ranka, Modeling and Optimization of Parallel and Distributed Embedded Systems. Wiley-IEEE Press, New York, NY (2016).

[27] Y. Robert and D. Trystram, Optimal scheduling algorithms for parallel Gaussian elimination. *Theor. Comput. Sci.* **64** (1989) 159–173.

[28] Y. Saad, SPARSKIT: A Basic Tool Kit for Sparse Matrix Computations – Version 2. Technical report (1994).

[29] R. Saad, An optimal schedule for Gaussian elimination on an MIMD architecture. *J. Comput. Appl. Math.* **185** (2006) 91–106.

[30] W.H.A. Schilders and E.J.W. Ter Maten, Numerical Methods in Electromagnetics. In: Vol. 13 of *Handbook of Numerical Analysis*, edited by P. Ciarlet. Elsevier, New York, NY (2005).

[31] R. Sivarethinamohan, Operations Research. Tata McGraw-Hill, New York, NY (2008).

[32] R.S. Tsay, Analysis of Financial Time Series, 3rd edition. John Wiley & Sons, New York, NY (2010).

[33] S. Yu, K. Li and Y. Xu, A DAG task scheduling scheme on heterogeneous cluster systems using discrete IWO algorithm. *J. Comput. Sci.* **26** (2018) 307–317.

[34] N. Zhou, D. Qi, X. Wang, Z. Zheng and W. Lin, A list scheduling algorithm for heterogeneous systems based on a critical node cost table and pessimistic cost table. *Concurr. Comput. Pract. E.* **29** (2017) e3944.

[35] Z. Zlatev, P. Vu, J. Wasniewski and K. Schaumburg, Computations with symmetric, positive definite and band matrices on a parallel vector processor. *Parallel Comput.* **8** (1988) 301–312.