RAIRO-Oper. Res. 55 (2021) 545–559
https://doi.org/10.1051/ro/2021009

RAIRO Operations Research
www.rairo-ro.org

DESIGN, ANALYSIS AND PERFORMANCE EVALUATION OF PARALLEL ALGORITHMS FOR SOLVING TRIANGULAR LINEAR SYSTEMS ON MULTICORE PLATFORMS*

Mounira Belmabrouk[1] and Mounir Marrakchi[2],**

Abstract. In this paper, we focus on the schedulings of 2-steps graph with constant task cost obtained when parallelizing algorithm solving a triangular linear system. We present three scheduling approaches having the same least theoretical execution time. The first is designed through solving a 0-1 integer problem by Mixed Integer Programming (MIP), the second is based on the Critical Path Algorithm (CPA) and the third is a particular Column-Oriented Scheduling (COS). The MIP approach experiments were carried out and confirmed that the makespan values of the MIP scheduling coincide with those of the corresponding lower bound already reached. Experimental results of the last two approaches detailing both makespans and efficiencies are presented and show that their practical performances differ though they are theoretically identical. We compare also these results to those of the appropriate procedure into so-called PLASMA library (Parallel Linear Algebra for Scalable Multi-core Architectures).

Mathematics Subject Classification. 90C10, 68M20, 65F05.

Received May 13, 2019. Accepted January 18, 2021.

1. Introduction

Solving a triangular linear system (STLS) $AX = B$ has been well studied theoretically and experimentally. Here, A is a non-singular $N \times N$ lower triangular matrix, X and B are two vectors of size N. In the literature, several parallel algorithms [6, 10, 12, 16, 18] have been designed to improve the performance of STLS which is a basic operation in numerical linear algebra [23]. These previous studies have not yet so far reached the optimum for two mains reasons: the processor activity has not been maximized yet and the communication costs still predominate the computing cost due to fine granularity, especially for point-wise methods. In this paper, our goal consists to study the behaviour of both point-wise and block-wise parallel algorithms for STLS assuming that the corresponding precedence graph is 2-steps structured and involve constant task cost. Our work deals with the efficient use of multicore systems in order to schedule such parallel algorithms based on the so-called task graph formalism approach [7]. The main contributions of the paper are the following. We propose a Mixed

Keywords. 0-1 integer problem, task scheduling, parallel algorithm, PLASMA library, triangular linear system.

* *This paper is an extension of a previous work presented at Codit'17 [4].*

[1] Faculty of Letters and Humanities, University of Sousse, Sousse, Tunisia.

[2] Department of Computer Sciences and Communication, Faculty of Sciences, University of Sfax, Sfax, Tunisia.

**Corresponding author: `marrakchimounir@yahoo.fr`, `mounir.marrakchi@fss.usf.tn`

Integer Programming (MIP) formulation for the problem consisting in scheduling, with minimum execution, the resolution of N-sized triangular system using p processors $2 \leq p \leq N$. We also present two other schedulings for solving the same triangular system which has theoretically the same optimal execution time, namely the Critical Path Algorithm (CPA) [4, 17] and Column Oriented Scheduling (COS) [3]. CPA is designed by sorting at each step the independent tasks, *i.e.*, whose predecessors have been executed, using the lengths of their critical paths. As to COS, the tasks of each column belonging to the 2-steps graph are both executed by only one processor. In order to validate our work, the practical results of MIP problem describing the scheduling of 2-steps graph have been analyzed and compared to those of previous works. CPA and COS have been implemented on a target parallel computer reserved in Grid'5000 and their performances are compared. We also compare our experimental results with those obtained by the corresponding procedure of the PLASMA library [1, 2, 8, 25]. It has to be noticed that the experimental CPU time of COS is minimum compared to others, especially when the size of the blocks is large enough. The reason is that the total processor idle time is minimum. Moreover, in the block-wise method, the need to go to global memory access is reduced since only one processor executes all the tasks of the same column of the precedence graph. Therefore the number of access to the global memory decreases when the block size increases thus reducing the total communication time [5, 15]. It should also be underlined that preemption is not allowed, *i.e.*, each task is executed by the same processor sequentially without interruption. Remark that henceforth we use in the same way the term processor or the term core to indicate one computing resource.

The remainder paper is organized as follows. We begin by presenting, in Section 2 a brief state-of-the-art on related works. In Section 3, the task decomposition and the precedence graph of the sequential algorithm are described. Section 4, is devoted to describing the three parallel schedulings, *i.e.*, the MIP-based, CPA and COS. In Section 5, before concluding we introduce the PLASMA (Parallel Linear Algebra for Scalable Multi-core Architectures) library and we present the programming environment used for our experimentations, expose some experimental results and compare them through makespans and efficiencies of CPA and COS schedulings with those of the corresponding routine named PLASMA-dtrsm() and included in PLASMA library.

2. Related works and motivation

STLS is the main step in numerous methods solving dense linear systems. Several authors have developed scalable algorithms adapted to current architectures such as distributed memory machine, shared memory machine, hybrid machine CPU/GPU, and many more. The work of Gonzalez-Domininiguez *et al.* [10] targeted multicore clusters using unified parallel C language of which they exploited its particularities to design efficient parallel implementations of dense triangular solvers, both in terms of execution time and required memory. Belmabrouk *et al.* [3] studied this problem on a distributed memory machine and introduced a parallel algorithm called Column Oriented Scheduling (COS) as well as compared their performances with that of the corresponding subroutine belonging to the PBLAS2 library. Wicky *et al.* [24] presented theoretically parallel algorithms to solve triangular systems with multiple right-hand sides. In this study, they have used a specific model to evaluate execution time and they have presented and analyzed algorithms to reduce communication overhead. The CPLEX tool [14] which solves some mathematical problems, provides a parallel subroutine for STLS on a distributed memory machine. This designed subroutine was offered in accordance with the proposed PBLAS library level 2. It should be noticed that the matrix is distributed in square block-cyclic fashion. This allows minimizing the overhead communication by exchanging large blocks of data. Other theoretical and experimental results have been presented on such problem, among these works, we mention those of Charara *et al.* [6] who proposed to adopt a recursive formulation implemented on GPUs that reduced the memory traffic in global memory. Mikkelsen *et al.* [16] presented parallel robust triangular solvers with floating-point arithmetic which use a task-based run-time system. In their parallel method, the matrix and the right-hand sides of the triangular system to be solved are split into blocks. In order to obtain a solution close to the exact one, they introduced an algorithm for updating the right-hand sides thus increasing the total execution time. As to our paper, we use Mixed Integer Programming that permits to design optimal scheduling for solving a triangular system.

Subsequently, we present two parallel algorithms COS [3] and CPA [17] to solve this problem having the same theoretical parallel execution times with that obtained by MIP for small values of matrix size. This is due to the significant complexity of MIP scheduling problem. Finally, we implemented in shared memory architecture CPA and COS and compare their experimental results with previous works like PLASMA. We show experimentally that COS is better than the others because its cost of communication is reduced [5,8,15].

3. Precedence graph

Consider $AX = B$ a triangular linear system, where $A = (a_{i,m})$ is a non-singular $N \times N$ lower triangular matrix, $X = (x_m)$ and $B = (b_m)$ are two vectors of size N. The decomposition into tasks of STLS sequential algorithm is shown in Algorithm 1. It should be recalled that it is possible to have a point wise or block wise decomposition where the size of each block is $r \times r$. Consequently we assume that $N = n \times r$. As in [23], we split the matrix A (resp. X, B) into $r \times r$ (resp. $r \times 1$) sub-matrices $A_{k,j}$ (resp. subvectors X_j, B_j) where $1 \leq k \leq n$ and $1 \leq j \leq n$. We have $A_{k,j} = (a_{i,m})$ (resp. $X_j = (x_m)$, $B_j = (b_m)$) where $(j-1)r + 1 \leq i \leq jr$, $(k-1)r + 1 \leq m \leq kr$ and $A_{k,k}$ $(1 \leq k \leq n)$ is a lower triangular sub-matrice. Algorithm 1, where we assume that the elements of B are initially stored in X, describes the sequential block resolution of a lower triangular system segmented into tasks where the task $T^r_{k,k}$ $(1 \leq k \leq n)$ corresponds to the resolution of a triangular system of size r and the task $T^r_{k,j}$ $(k+1 \leq j \leq n)$ corresponds to the elimination of r (already determined), unknowns in the r associated equations. In fact, with task $T^r_{1,1}$, *i.e.*, $k = 1$, Algorithm 1 solves firstly the triangular system $A_{1,1}X_1 = B_1$ by forward substitution scheme and then with tasks $T^r_{1,j}$ $(2 \leq j \leq n)$, the determinated values of X_1 are removed. For $k = 2$, it solves triangular system with task $T^r_{2,2}$ to have values of X_2 which are subsequently eliminated with tasks $T^r_{2,j}$ $(3 \leq j \leq n)$ and so on until $k = n$. Here we have two precedence constraints:

$$T^r_{k,j} \ll T^r_{k+1,j} \quad \text{where} \quad 1 \leq k \leq n-1 \quad \text{and} \quad k+1 \leq j \leq n.$$
$$T^r_{k,k} \ll T^r_{k,j} \quad \text{where} \quad 1 \leq k \leq n-1 \quad \text{and} \quad k+1 \leq j \leq n.$$

$T \ll T'$ means that the execution of task T must be completed before T' begins. We assume in the remainder that a combined multiplication-substraction or a division costs one time unit. Hence, the execution time of each task $T^r_{k,j}, 1 \leq k < j \leq n$, (resp. $T^r_{k,k}$, $1 \leq k \leq n$) is constant and equal to r^2 (resp. $\frac{r^2+r}{2}$). Since the number of tasks $T^r_{k,j}$, $1 \leq k < j \leq n$, (resp. $T^r_{k,k}$, $1 \leq k \leq n$) is $\frac{n(n-1)}{2}$ (resp. n), we assume that the execution time of each task is constant and equal to r^2 time units in general and one time unit if $r = 1$. For simplicity and clearness, the tasks $T^r_{k,k}$ and $T^r_{k,j}$ are written in the following by T_{kk} and T_{kj} respectively. The precedence graph of size n called 2-steps graph [19] and illustrated for $n = 6$ in Figure 1, is constituted by $\frac{n(n+1)}{2}$ tasks T_{ij}, where $1 \leq i \leq j \leq n$. It involves n columns denoted $C_1, \ldots, C_n$, where each C_i $(1 \leq i \leq n)$ is constituted by tasks T_{1i}, $T_{2i}, \ldots, T_{ii}$. The critical path of a given task $T_{i,j}$ is is the longest path between $T_{i,j}$ and $T_{n,n}$. It is the path which begins by $T_{i,j}$ and ends by the terminal task $T_{n,n}$ belonging to the precedence graph. This is the path defined by $T_{i,j}; T_{i+1,j}; \ldots; T_{j,j}; T_{j+1,j}; \ldots; T_{n,n}$. Its length is equal to $cp(T_{i,j}) = 2n - i - j + 1$ which is the number of its tasks. The length of the longest path of the precedence graph, *i.e.*, $\{T_{i,i}; T_{i,i+1}, 1 \leq i \leq n-1\} \cup \{T_{n,n}\}$, is equal to $2n - 1$. Let us mention that the 2-steps graph is also the precedence graph of several other algorithms of some linear algebra algorithms, such as LU factorization. In the following, we assume that p processors, denoted $1, \ldots, p$, are available unless otherwise stated.

4. Parallel schedulings

In this section, we describe three schedulings to solve a triangular system in optimal time. Proposed approaches are based respectively on critical path heuristic, MIP tool and column-oriented method.

Algorithm 1. Decomposition into tasks.

```
for $k = 1$ to $n$ do
      for $i = (k-1)r + 1$ to $kr$ do                 ⎫
        for $m = (k-1)r + 1$ to $i - 1$ do            ⎪
          $x_i \leftarrow x_i - a_{i,m} * x_m$        ⎬ Task $T^r_{k,k}$
        end for                                       ⎪
        $x_i \leftarrow x_i / a_{i,i}$                ⎪
      end for                                         ⎭
  for $j = k + 1$ to $n$ do
        for $i = (j-1)r + 1$ to $jr$ do               ⎫
          for $m = (k-1)r + 1$ to $kr$ do             ⎪
            $x_i \leftarrow x_i - a_{i,m} * x_m$      ⎬ Task $T^r_{k,j}$
          end for                                     ⎪
        end for                                       ⎭
  end for
end for
```

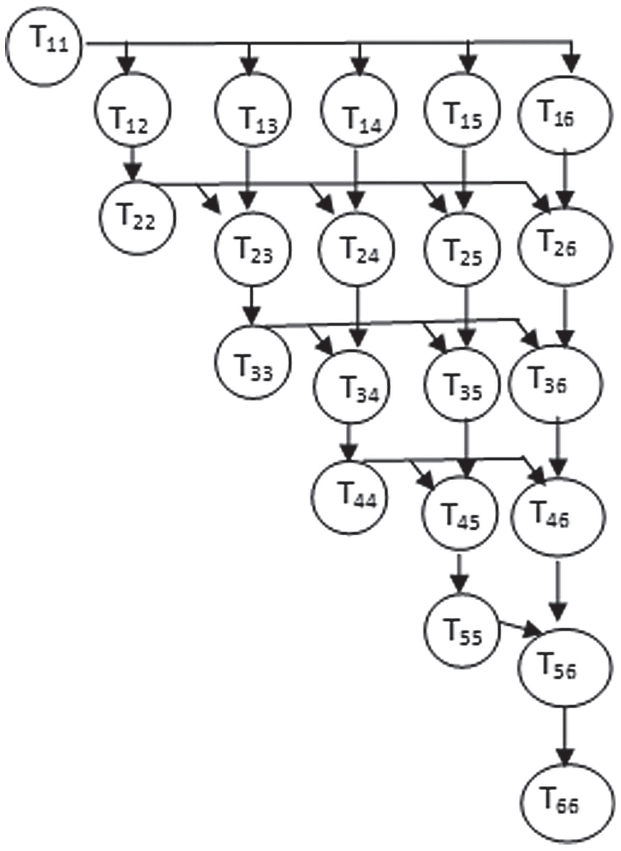

FIGURE 1. The 2-steps graph for $n = 6$.

4.1. Critical Path Algorithm (CPA)

The CPA is described as follows [17]. At each step, it executes among the independent tasks, *i.e.*, tasks whose predecessors have been executed thus tasks freed of precedence constraints, those having the longest critical path. Indeed, if at time t there is a free processor, CPA chooses, among the independent tasks, a task that has the longest critical path. If there are two or more independent tasks with the same critical path length, the choice becomes arbitrary between them. In other words, if T and T' are two independent tasks at time t: if $cp(T) > cp(T')$ then the execution of T begins no later than at the same time that T' and if $cp(T) = cp(T')$ then the order of execution is arbitrary. Algorithm 2 describes the assignment of processors to the tasks. The makespan of CPA is equal to

$$T_p = \begin{cases} \left\lceil \frac{(n-1)(n+2)}{2p} \right\rceil + p, & \text{if } \ 2 \le p < p_{\text{opt}} \\ 2n-1, & \text{if } p_{\text{opt}} \ \le p \le n-1 \end{cases}$$

where $p_{\text{opt}} = \left\lceil \frac{2n-1-\sqrt{2n^2-6n+5}}{2} \right\rceil$ is the minimum number of processors for which the execution of the 2-steps graph may be achieved in the optimal time, *i.e.*, equal to $2n-1$ length of the longest path of precedence graph of size n [17].

Algorithm 2. Critical Path Algorithm (CPA).

```
Assign one processor to execute $T_{1,1}$
/* $S$: set of independent tasks not yet executed and having longest critical path*/
$S \leftarrow \emptyset$
$L \leftarrow \{T_{1,i}, 2 \le i \le n\}$ /* Set of independent tasks*/
while ($L \ne \emptyset$) do
  $q \leftarrow 0$
  while ($L \ne \emptyset$) and ($q < p$) do
/* Selection, in current time, $q$ ($q \le p$) independent tasks to execute*/
    $T \leftarrow argmax\{cp(T'), T' \in L\}$
    $L \leftarrow L \backslash \{T\}$
    $S \leftarrow S \cup \{T\}$
    $q \leftarrow q + 1$
  end while
  Assign $q$ processors to execute $q$ tasks of $S$
  $S \leftarrow \emptyset$
  $L \leftarrow L \cup F$ /* $F$: set of tasks becoming independent in current time*/
end while
```

4.2. MIP scheduling

In the approach using MIP and like described in [9, 22], we formulate the scheduling as an optimization problem with binary variables under constraints defined by priority of the tasks and parallel environment used. The solution of the problem provides scheduling executing the 2-steps graph in minimum theoretical execution time. In the following, we detail the formulation based on $0-1$ variables $y_{i,j}^{k,q}$ and defined by:

$$y_{i,j\ge i}^{k,q} = \begin{cases} 1 & \text{if processor } q \text{ starts the execution of task } T_{i,j} \text{ at time } k-1 \\ 0 & \text{otherwise.} \end{cases}$$

Our goal is to find both the values of the binary variables and the minimum of the following objective function:

$$t_{n,p} = \sum_{k=1}^{K} \left(\max_{1 \le q \le p} \left(\sum_{i=1,j=i}^{n} y_{i,j}^{k,q} \right) \right)$$

where $t_{n,p}$ (resp. $K = \text{Max}(\lceil \frac{(n-1)(n+2)}{2p} \rceil + p, 2n-1)$) is the makespan of scheduling described by the binary variables $y^{k,q}_{i,j\geq i}$ (resp. of CPA) to execute 2-steps graph of size n with constant task cost equal to one unit time where p processors are available ($2 \leq p \leq n-1$). Let us precise that the execution of the precedence graph begins at time $t = 0$ by processing the task $T_{1,1}$. Then, the optimization problem, named MIP algorithm, may be formulated as follows:

$$\begin{aligned}
&\text{Minimize } (t_{n,p})\\
&\text{Subject to}\\
&\quad \text{(i)}\quad \sum_k \sum_q y^{k,q}_{i,j} = 1 && \forall i = 1,\dots,n,\ \forall j = i,\dots,n\\
&\quad \text{(ii)}\quad \sum_{i,j\geq i} y^{k,q}_{i,j} \leq 1 && \forall k = 1,\dots,K,\ \forall q = 1,\dots,p\\
&\quad \text{(iii)}\quad \sum_q \sum_{k=1}^{l-1} y^{k,q}_{i,j} \geq \sum_q y^{l,q}_{i+1,j} && \forall l = 2,\dots,K,\ \forall i = 1,\dots,n-1,\ \forall j = i+1,\dots,n\\
&\quad \text{(iv)}\quad \sum_q \sum_{k=1}^{l-1} y^{k,q}_{i,i} \geq \sum_q y^{l,q}_{i,j} && \forall l = 2,\dots,K,\ \forall i = 1,\dots,n-1,\ \forall j = i+1,\dots,n\\
&\quad \text{(v)}\quad y^{1,1}_{1,1} = 1, y^{1,q}_{i,j} = 0 && \forall i = 2,\dots,n,\ \forall j = i,\dots,n,\ \forall q = 1,\dots,p\\
&\quad \text{(vi)}\quad y^{1,q}_{1,j} = 0 && \forall j = 2,\dots,n,\ \forall q = 1,\dots,p\\
&\quad \text{(vii)}\quad y^{k,q}_{i,j} \in \{0,1\} && \forall i = 1,\dots,n,\ \forall j = i,\dots,n,\ \forall k = 1,\dots,K,\ \forall q = 1,\dots,p.
\end{aligned}$$

Constraint (i) means that each task is executed by one processor within a time step. Constraint (ii) guarantees that each processor executes one task at most on each time step. Constraints (iii) and (iv) correspond to the inter-tasks precedence constraints. Constraints (v) and (vi) mean that initially, *i.e.*, at time $k = 1$, the only task processed is $T_{1,1}$. Constraint (vii) specifies that all variables are binary. We may see that the problem involves $Kp\frac{n(n+1)}{2}$ (resp. at least $\frac{n(n+1)}{2} + n(n-1)(K-1) + pK$) binary variables (resp. constraints). Both numbers grow on the order of $O(n^4)$. For the implementation and evaluation of the above binary integer program with different values of n and p, we used the Cplex Mixed Integer Programming (MIP) version 12.10 [13]. The experiments were carried out on an Intel Core 2 Quad Q8300, 4 cores, 2.50 GHz, 4 GB of RAM, setting a time limit of three hours. The obtained values are summarized in Table 1 which is split into four columns as follows: the first one indicates the instances defined by the size of 2-steps graph n and number of available processors p. In the second column gives MIP and CPA makespan's values are given, as well as the execution times of the MIP algorithm are mentioned in the third column. The last column contains p_{opt} [17] lower bound of the number of processors to execute 2-steps graph in minimum time. The experimental results of the MIP algorithm confirm those obtained by CPA. Note that the complexity of the problem is rather high for Cplex tool and that its running requires a large amount of time even for small values of n. Indeed, for some values of n as 18, 20, 22, 24 and $p = 2$, MIP algorithm lasts more than three hours without giving any results. This is justified by the large number of binary variables and constraints and also by the continuous activity of processors which slows down obtaining optimal scheduling. Furthermore, if the number of available processors is sufficient so that processors can be inactive for a certain time during execution, *i.e.*, the processors are not always busy during execution, then the assignment of processors to tasks becomes easy and MIP algorithm provides the MIP scheduling and exact theoretical makespan in a short time even if the number of binary variables and constraints is quite high. In this context, we cite the cases in Table 1 where $p \geq p_{\text{opt}}$ like $n = 22$ or 24 and $p = 8$.

Table 1. MIP and CPA results.

Instance		Makespan		CPU	p_{opt}
n	p	MIP	CPA [17]		
10	2	29	29	18.81	4
	4	19	19	1.60	
	6	19	19	1.38	
	8	19	19	1.79	
12	2	41	41	58.64	5
	4	24	24	10.95	
	6	23	23	2.34	
	8	23	23	2.80	
14	2	54	54	405.2	5
	4	30	30	47.11	
	6	27	27	4.22	
	8	27	27	5.19	
16	2	70	70	5080.56	6
	4	38	38	78.31	
	6	31	31	8.02	
	8	31	31	10.32	
18	2	–	87	10 800	6
	4	47	47	1088	
	6	35	35	13.5	
	8	35	35	22.10	
20	2	–	107	10 800	7
	4	57	57	5954	
	6	41	41	196.38	
	8	39	39	22.39	
22	2	–	128	10 800	7
	4	–	67	10 800	
	6	–	42	10 800	
	8	43	43	66.1	
24	2	–	152	10 800	8
	4	–	79	10 800	
	6	–	56	10 800	
	8	47	47	129.85	

4.3. Column Oriented Scheduling (COS)

In the following, we will present a Column Oriented Scheduling COS for the 2-steps graph when p processors are available [3]. Such algorithm assumes that the precedence graph has a particular size n equal to $2qp + 1$, where q is an integer greater or equal to 2. Such expression of n for the precedence graph size except its first task $T_{1,1}$ allows appropriate assignment of its columns to processors. We obtain q blocks denoted B_0, $B_1, \ldots, B_{q-1}$. Each block contains exactly $2q$ columns where $C_{2kp+j+1}$ and $C_{2(k+1)p-j+2}$ belonging to block B_k $(0 \leq k \leq q-2)$, are assigned to processor j $(1 \leq j \leq p)$. In the last block B_{q-1}, processor j executes tasks of $C_{2(q-1)p+j+1}$ and $C_{(2q-1)p+j+1}$. In the next, we define the order of execution for the tasks assigned to processor j. Note that after execution the first task T_{11} by one processor at time $t = 0$, each processor is active on two successively assigned columns of block $B_k(0 \leq k \leq q-1)$, *i.e.*, each processor j alternately executes a task from the current column and a task from the following assigned to j. Assume that C_u, C_v and C_w are three successive columns executed by the same processor j and belonging on two successive blocks B_k and B_{k+1} for any k $(0 \leq k \leq q-2)$. Processor j executes one task from C_u and one task from C_v until C_{u-1} will be finished whereas the remaining tasks of the column C_u are sequentially executed by processor j without interruption. Afterwards, the same work is achieved by processor j alternately between C_v and C_w as long as the execution C_{v-1} is not yet finished. Such process continues until processor 1 finishes the execution of the task T_{uu} where $u = 2(q-1)p+1$, *i.e.*, the last task of B_{q-2}. At that time, each processor continues the execution of the tasks not yet executed of its first column belonging to B_{q-1}. Then, it executes the other tasks not yet executed from its second column assigned in the same block. In fact, the processor does not, alternately, execute its tasks *i.e.*, the placement order of each task place in its column gives the order of its execution by its assigned processor. Algorithm 3 details the procedure (P_j) which describes the operation of any processor j $(1 \le j \le p)$ by executing the tasks assigned to it belonging to $B_k(0 \le k \le q-1)$ from the time 1 until the execution of T_{uu} $(u = 2(q-1)p+1)$ finishes. The example depicted in Table 2 provides the processor that executes each task as well as the time of its execution start, when $r = 1$, $p = 3$ and $q = 3$, hence $N = n = 19$. Here, each processor k denoted P_k, where $1 \le k \le 3$, executes tasks of six columns:

P_1 executes tasks of columns $C_2, C_7, C_8, C_{13}, C_{14}, C_{17}$.
P_2 executes tasks of columns $C_3, C_6, C_9, C_{12}, C_{15}, C_{18}$.
P_3 executes tasks of columns $C_4, C_5, C_{10}, C_{11}, C_{16}, C_{19}$.

In Table 2, i, j (resp. t) in column P_k (resp. Time) means processor k starts execution of $T_{i,j}$ at time t. To understand Algorithm 3, we give an overview of how procedure (P_2) works. Firstly, the variables k, u and v are respectively initialized to 0, 3 and 6. Then P_2 executes in order, as indicated in Table 2 and from $t = 1$ to $t = 4$, tasks $T_{1,3}$, $T_{1,6}$, $T_{2,3}$ and $T_{3,3}$. Since the execution of C_3 is achevied, the variables k, u and v are updated and become equal to 1, 6 and 9 respectively. P_2 treates then from $t = 5$, the tasks $T_{2,6}, T_{1,9}, T_{3,6}, T_{2,9}$ belonging to C_6 and C_9 as mentioned below. This lasts until time 9 where P_2 continues the execution of tasks belonging to C_6. At time 12, it finishes the execution of C_6 and u (resp. v) gets the value 9 (resp. 12). Thus, P_2 starts execution of C_{12} at time 13. This process continues until time 39 where P_1 finishes the execution of the tasks of column C_{13} belonging to B_1 (since $13 = 2(q-1)p+1$, where $q = p = 3$). At this time Procedure (P_2) is terminated. Algorithm 4 describes COS by specifying how the p processors operate in parallel, how each processor j $(1 \le j \le p)$ works to execute the procedure (P_j) and complete the execution of columns $C_{2(q-1)p+j+1}$ and $C_{(2q-1)p+j+1}$. In the continuation of the description of our example cited above for P_2, we

Algorithm 3. Procedure (P_j).

```
$k \leftarrow 0$;
$u \leftarrow j+1$;
$v \leftarrow 2p-j+2$;
while $(u \le 2(q-1)p+1)$ do
    while (execution of $C_{u-1}$ not yet terminated) do
        $P_j$ executes one task of $C_u$ and then one task of $C_v$
    end while
    $P_j$ terminates the execution of $C_u$
    $u \leftarrow v$
    if $u = 2(k+1)p-j+2$ then
        $k \leftarrow k+1$;
        $v \leftarrow 2kp+j+1$
    else
        if $k \ne q-1$ then
            $v \leftarrow 2(k+1)p-j+2$
        else
            $v \leftarrow (2q-1)p+j+1$
        end if
    end if
end while
```

Algorithm 4. Column Oriented Scheduling (COS).

```
Execution of task $T_{11}$
for $j = 1$ to $p$ do                                    /* in parallel*/
  Procedure ($P_j$)
  Processor $P_j$ terminates execution of $C_{2(q-1)p+j+1}$ and after execution
  of $C_{(2q-1)p+j+1}$
end for
```

Table 2. COS for $r = 1$, $p = 3$ and $q = 3$.

Time	P_1	P_2	P_3	Time	P_1	P_2	P_3	Time	P_1	P_2	P_3
0	1, 1			22	3, 13	4, 12	7, 10	44	14, 14	12, 15	10, 16
1	1, 2	1, 3	1, 4	23	3, 14	1, 15	8, 10	45	1, 17	13, 15	11, 16
2	2, 2	1, 6	1, 5	24	4, 13	5, 12	9, 10	46	2, 17	14, 15	12, 16
3	1, 7	2, 3	2, 4	25	4, 14	2, 15	10, 10	47	3, 17	15, 15	13, 16
4	1, 8	3, 3	2, 5	26	5, 13	6, 12	7, 11	48	4, 17	3, 18	14, 16
5	2, 7	2, 6	3, 4	27	5, 14	3, 15	8, 11	49	5, 17	4, 18	15, 16
6	2, 8	1, 9	4, 4	28	6, 13	7, 12	9, 11	50	6, 17	5, 18	16, 16
7	3, 7	3, 6	3, 5	29	6, 14	4, 15	10, 11	51	7, 17	6, 18	5, 19
8	3, 8	2, 9	4, 5	30	7, 13	8, 12	11, 11	52	8, 17	7, 18	6, 19
9	4, 7	4, 6	5, 5	31	7, 14	9, 12	1, 16	53	9, 17	8, 18	7, 19
10	4, 8	5, 6	1, 10	32	8, 13	10, 12	1, 19	54	10, 17	9, 18	8, 19
11	5, 7	6, 6	1, 11	33	8, 14	11, 12	2, 16	55	11, 17	10, 18	9, 19
12	6, 7	3, 9	2, 10	34	9, 13	12, 12	2, 19	56	12, 17	11, 18	10, 19
13	7, 7	1, 12	2, 11	35	10, 13	5, 15	3, 16	57	13, 17	12, 18	11, 19
14	5, 8	4, 9	3, 10	36	11, 13	1, 18	3, 19	58	14, 17	13, 18	12, 19
15	6, 8	2, 12	3, 11	37	12, 13	6, 15	4, 16	59	15, 17	14, 18	13, 19
16	7, 8	5, 9	4, 10	38	13, 13	2, 18	4, 19	60	16, 17	15, 18	14, 19
17	8, 8	3, 12	4, 11	39	9, 14	7, 15	5, 16	61	17, 17	16, 18	15, 19
18	1, 13	6, 9	5, 10	40	10, 14	8, 15	6, 16	62		17, 18	16, 19
19	1, 14	7, 9	5, 11	41	11, 14	9, 15	7, 16	63		18, 18	17, 19
20	2, 13	8, 9	6, 10	42	12, 14	10, 15	8, 16	64			18, 19
21	2, 14	9, 9	6, 11	43	13, 14	11, 15	9, 16	65			19, 19

say that from time 39, it finishes the execution of the tasks not yet executed of C_{15} then of C_{18} sequentially. It should be noted that processors 1 and 3 work in a similar way as that of processor 2. In [3], we show that execution is done according to the precedence constraints and COS keeps all available processors active without interruption as long as possible. Each processor executes exactly $q(2qp + 3) + 2j - p - 1$ tasks in all from the 2-steps graph except its first task $T_{1,1}$. The latest task $T_{n,n}$ of precedence graph will be executed by processor p, thereafter, the makespan of COS is equal to $q(2qp + 3) + p - 1$. As a function of n, such makespan coincides with the optimal theoretical value $T_p = \lceil \frac{(n-1)(n+2)}{2p} \rceil + p$ for executing the 2-steps graph with the constant task cost where n is the size of the precedence graph and $2 \leq p \leq p_{\text{opt}}$ [17].

5. Experimental results

In this section, we give some details of the PLASMA library, describe parallel programming environment used and present some experimental results.

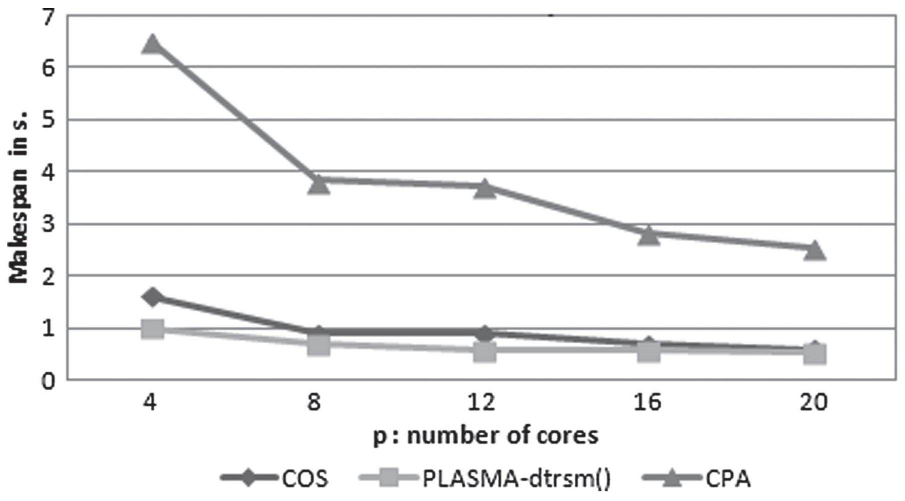

FIGURE 2. Experimental makespans for $N = 14\,401$ and $r = 1$.

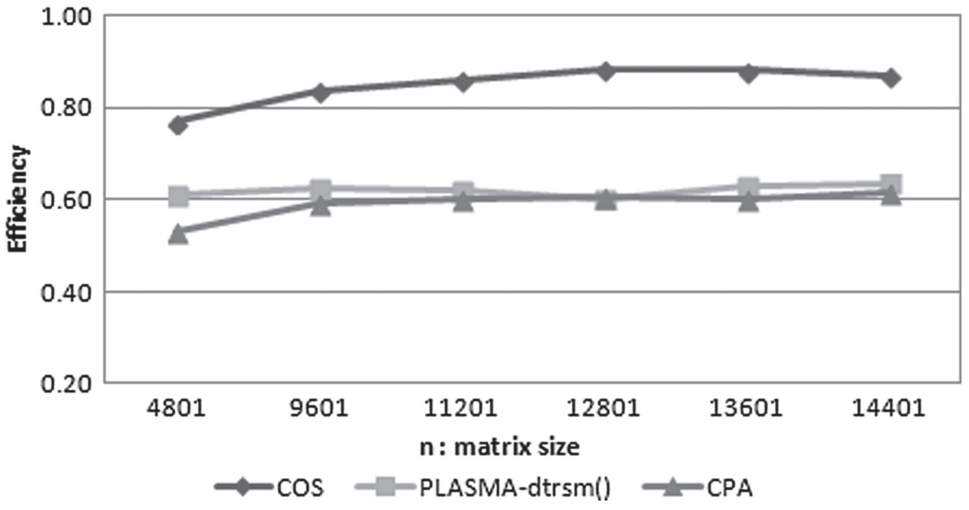

FIGURE 3. Experimental efficiencies for $p = 4$ and $r = 1$.

5.1. PLASMA library and programming environment

The Parallel Linear Algebra Software for Multicore Architectures (PLASMA) is a software library designed to be efficient on homogeneous multicore processors. PLASMA is a redesign of LAPACK and ScaLAPACK for shared memory architectures based on multi-core processor architectures [1, 5, 8]. This library is designed to deliver the goal of high performance that is reached by combining state of the art solutions in parallel algorithms, scheduling, and software engineering. In particular, PLASMA is built around the following three concepts. The first is the Tile Matrix Layout where the matrices are subdivided into square blocks, called tiles. A tile fits into the cache memory of one core. This method minimizes the number of cache misses and improves performance. Tile Algorithms is the second concept: PLASMA is in fact based on redesigned algorithms to work on tiles. This strategy maximizes data reuse and thus benefits from a better cache effect. The third concept is Dynamic Scheduling where the task assignment to the cores is done at run time, *i.e.*, the scheduling is based

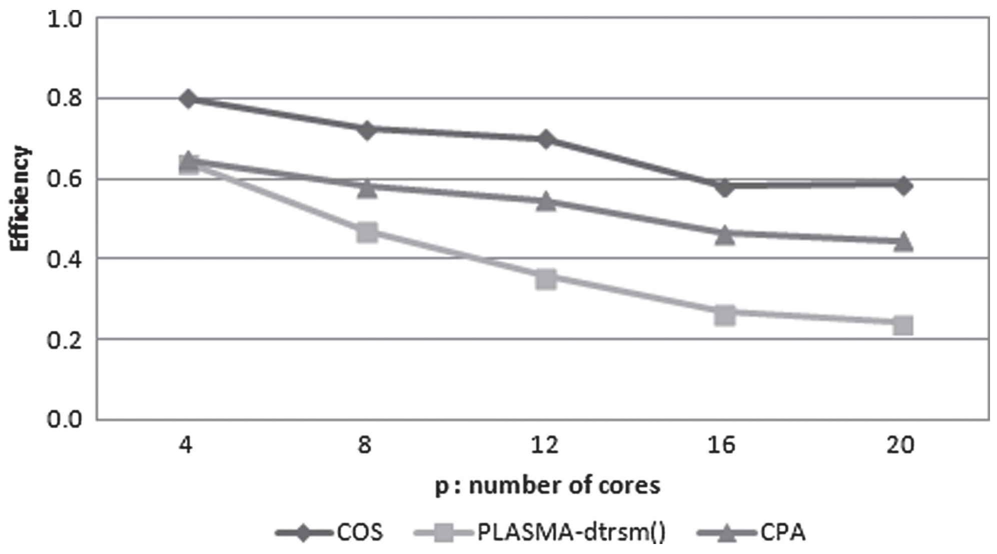

FIGURE 4. Experimental efficiencies for $N = 24\,004$ and $r = 4$.

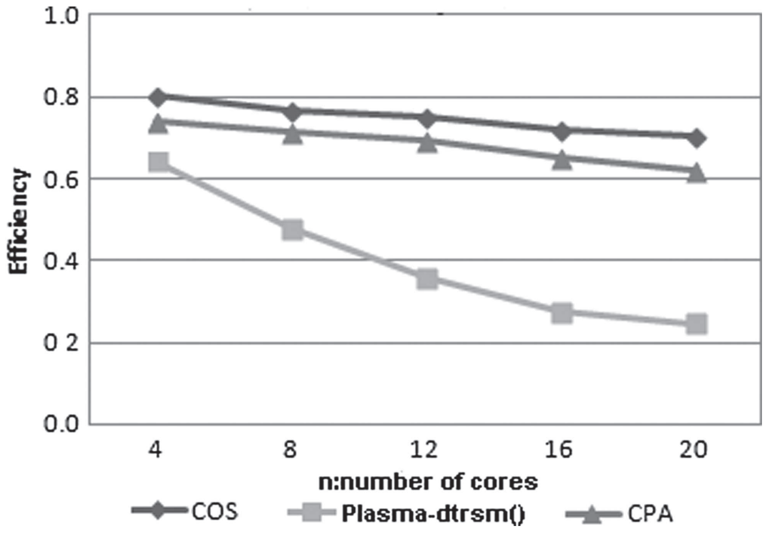

FIGURE 5. Experimental efficiencies for $N = 24\,020$ and $r = 20$.

on the idea of assigning work to cores based on the availability of data for processing at any given point in time. Thus is also sometimes called data-driven scheduling [8]. To validate the performances of the presented algorithms, we used the procedure named **PLASMA-dtrsm()** of PLASMA which solves a triangular system [21]. We prefer this choice firstly because PLASMA experimental results constantly surpass those of certain other libraries [1]. Secondly, because we are targeting to test the performance of such algorithms on multi-core platforms and the working environment that we use is the same as that of PLASMA: subdivision of the matrix into blocks and use the shared memory architecture. To compare experimental performances, the choice of CPA and COS schedulings is based on the fact that they have theoretically the same optimal makespan. Besides, the implementation of MIP scheduling is non-interesting because this approach deals with fairly small

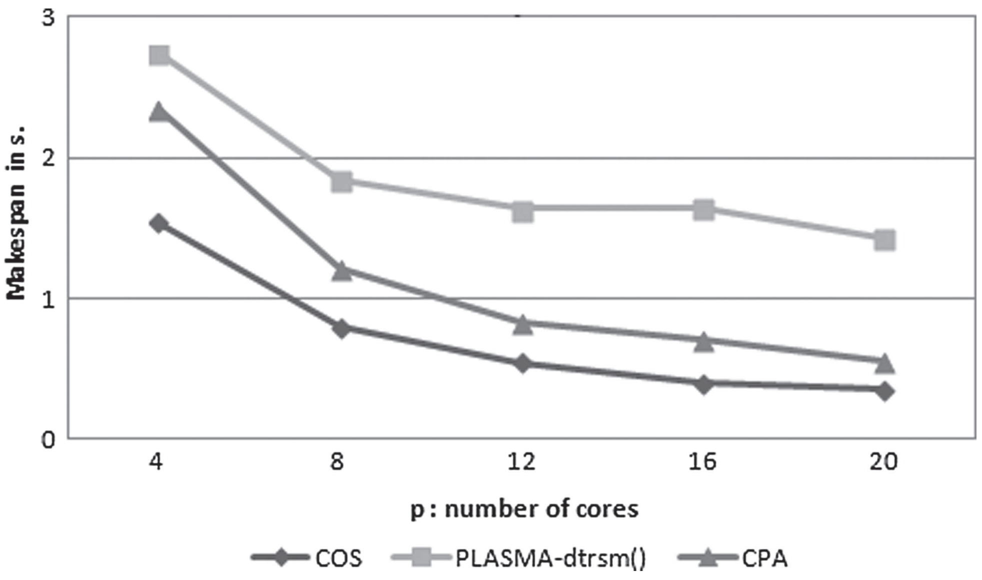

FIGURE 6. Experimental makespans for $N = 24\,020$ and $r = 20$.

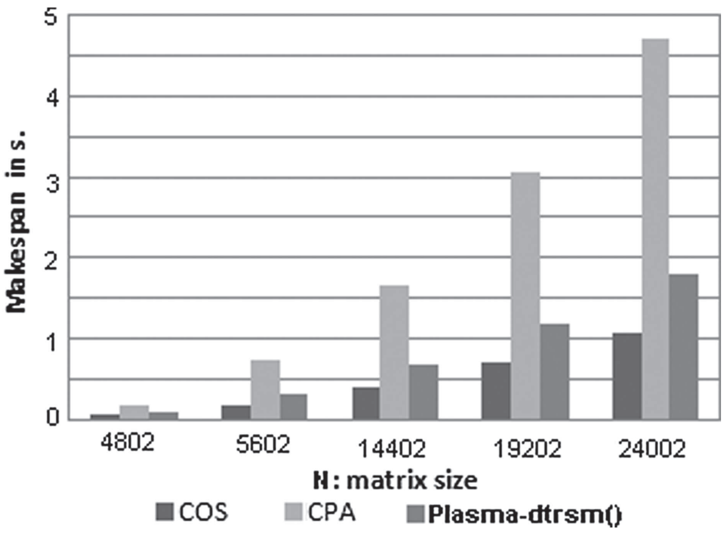

FIGURE 7. Experimental makespans for $r = 2$ and $p = 8$.

sizes due to the MIP algorithm complexity. We have to add that we implemented the COS and CPA parallel algorithms by using an application programming interface (API) that supports the programming of shared memory multiprocessing platforms. Thus, we chose OpenMP (Open Multi-Processing) [20]. In C/C++, OpenMP uses **#pragma** to specify the parallel sections. The communication inter-core and/or processors in the shared memory are implicit. The architecture of system we targeted is a node having two AMD Opteron 6164 HE CPUs, each has 12 cores, 1.7 GHz, 12 MB (RAM), 85 W (GRID'5000) [11]. The processed triangular matrices chosen randomly. Experimentally, the makespan is measured in seconds (s). It is the mean of many values (one hundred) of the execution time of the same parallel algorithm with the same parameters.

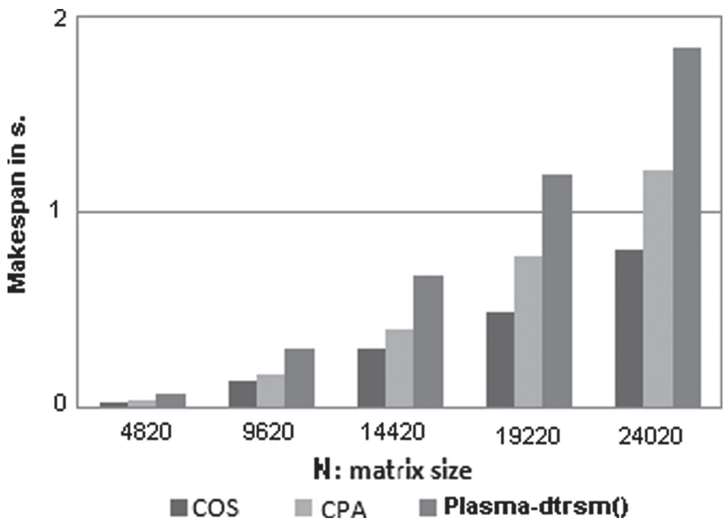

FIGURE 8. Experimental makespans for $r = 20$ and $p = 8$.

5.2. Test results and discussion

In this subsection, we compare the experimental results of the COS, CPA and the subroutine of PLASMA library. Figure 2 illustrates the variation of the makespan in seconds according to the number of cores, that ranges from 4 to 20. We can remark that for the different number of cores, the execution time of the PLASMA subroutine is the best when the block size r is equal to 1. If the number of cores increases, the execution time of COS becomes near to that of the PLASMA subroutine. Also, the experimental execution time of the CPA is larger than that of the COS. This is due to PLASMA subroutine which ignores the point-wise method and always works with the block-wise method. On the contrary in Figure 3, where the x-axis represents the matrix size n varying between 4801 and 14 401 and the y-axis represents the efficiency from 0.2 to 1, it is clear in the point-wise version and for a constant number of cores equal to 4, efficiencies, *i.e.*, core activity rate in COS exceed 0.8 and are better than those of the others. Efficiencies of CPA and PLASMA subroutine are pretty small. Efficiencies increase with the matrix size because processors idle times are independent of the problem size and remain constant when the number of the core is constant. Given p the number of cores belonging to $\{4, 8, 12, 16, 20\}$, Figures 4 (resp. 5) illustrates the efficiency values for $N = 24\,004$ and $r = 4$ (resp. $N = 24\,020$ and $r = 20$). The efficiencies of COS (resp. CPA) is better than that of CPA (resp. PLASMA subroutine). This shows that the core activity rate in COS is better than that of the others. It is obvious from Figure 6 depicting the makespan (in s.) in terms of the number of cores in the interval [4, 20], that the COS scheduling is better than that of the CPA (resp. PLASMA subroutine) for a matrix size $N = 24\,020$ and a block size $r = 20$ although theoretically CPA and COS have the same execution time without communication cost. This may be explained by the fact that in COS, the execution of each column of the 2-steps graph is assigned to only one core, which minimizes communication cost. The same results described above are illustrated in Figures 7 and 8 where the block size is equal respectively to 2 and 20. These figures present the variations of the makespan according to the size of matrix when the number of cores is constant equal to 8. In fact, in COS scheduling each core reuses the information, which has just been calculated. The latter is still present in its cache and does not need to re-access the global memory. Furthermore, COS and CPA are faster than PLASMA subroutine, because cores in COS and CPA are active as long as possible, *i.e.*, the idle time of cores is minimum and the block method allows reduce memory traffic in a global memory [15].

6. Conclusion

In this paper, we have tried to select the most efficient parallel algorithm to solve a triangular linear system on a multicore shared-memory machine. For this, we have presented three scheduling approaches: MIP, COS, CPA. MIP scheduling is designed by the Cplex tool which formulates the task assignment to processors as an optimization problem with constraints. The solutions, found by MIP, of this problem, confirmed the existing results. COS, CPA are implemented and compared. They theoretically have the same makespan, but their experimental makespans differ, although a shared memory is used. This is due to communication overhead. Also, the practical complexities of COS and CPA are compared to those of PLASMA subroutine. COS has the least experimental execution time and highest efficiency because processor idle time is minimum and the tasks of each column of the 2-steps graph are executed by one processor. This allows the minimization of the communication overhead. The block-wise method reduces the number of accesses to global memory. It is possible to extend this experimental study to other parallel algorithms either on a shared or distributed memory system. In particular, we can implement other parallel algorithms on a distributed memory machine and compare their performances with the corresponding parallel BLAS subroutines.

Acknowledgements. We thank prof. Nahid EMAD for her reception in PRISM laboratory, University of Versailles and provision an account of GRID'5000 to make experimentations.

References

[1] A. Abdelfattah, H. Anzt, J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, S. Tomov, I. Yamazaki and A. YarKhan, Linear algebra software for large-scale accelerated multicore computing. *Acta Numer.* **25** (2016) 1–160.

[2] E. Agullo, J. Demmel, J. Dongarra, B. Hadri, J. Kurzak, J. Langou, H. Ltaief, P. Luszczek and S. Tomov, Numerical linear algebra on emerging architectures: the plasma and magma projects. In: Vol. 180 of *Journal of Physics: Conference Series*. IOP Publishing, Bristol, UK (2009) 012037.

[3] M. Belmabrouk and M. Marrakchi, Optimal parallel scheduling for resolution a triangular system with availability constraints. In: *2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA)*. IEEE, Piscataway, NJ, USA (2015) 1–7.

[4] M. Belmabrouk and M. Marrakchi, Comparison of parallel scheduling for triangular system resolution on multi-core processors. In: *2017 4th International Conference on Control, Decision and Information Technologies (CoDIT)*. IEEE, Piscataway, NJ, USA (2017) 0651–0656.

[5] A. Buttari, J. Langou, J. Kurzak and J. Dongarra, A class of parallel tiled linear algebra algorithms for multicore architectures. *Parallel Comput.* **35** (2009) 38–53.

[6] A. Charara, D. Keyes and H. Ltaief, A framework for dense triangular matrix kernels on various manycore architectures. *Concurrency Comput. Pract. Experience* **29** (2017) e4187.

[7] E.G. Coffman and P.J. Denning, Operating Systems Theory. Prentice-Hall Englewood Cliffs, NJ, USA (1973).

[8] J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, P. Wu, I. Yamazaki, A. YarKhan, M. Abalenkovs, N. Bagherpour and S. Hammarling, Plasma: Parallel linear algebra software for multicore using openmp. *ACM Trans. Math. Softw. (TOMS)* **45** (2019) 1–35.

[9] C.A. Floudas and X. Lin, Mixed integer linear programming in process scheduling: modeling, algorithms, and applications. *Ann. Oper. Res.* **139** (2005) 131–162.

[10] J. González-Domínguez, M.J. Martín, G.L. Taboada and J. Tourino, Dense triangular solvers on multicore clusters using upc. *Proc. Comput. Sci.* **4** (2011) 231–240.

[11] Grid'5000, [online] https://www.grid5000.fr/mediawiki/index.php/Grid5000:Home (2007).

[12] R. Iakymchuk, D. Defour, S. Collange and S. Graillat, Reproducible triangular solvers for high-performance computing. In: *2015 12th International Conference on Information Technology-New Generations*. IEEE, Piscataway, NJ, USA (2015) 353–358.

[13] IBM ILOG CPLEX Optimization Studio CPLEX Users Manual (1999).

[14] IBM Knowlege Center, Solution of triangular system of equations with a single right-hand side. [online] https://www.ibm.com/support/knowledgecenter/.

[15] X. Jin, T. Yang and X. Tang, A comparison of cache blocking methods for fast execution of ensemble-based score computation. In: *Proceedings of the 39th International ACM SIGIR Conference On Research and Development in Information Retrieval* (2016) 629–638.

[16] C.C. Kjelgaard Mikkelsen, A.B. Schwarz and L. Karlsson, Parallel robust solution of triangular linear systems. *Concurrency Comput. Pract. Experience* **31** (2019) e5064.

[17] M. Marrakchi, Optimal parallel scheduling for the 2-steps graph with constant task cost. *Parallel Comput.* **18** (1992) 169–176.

[18] P.D. Michailidis and K.G. Margaritis, Parallel direct methods for solving the system of linear equations with pipelining on a multicore using openmp. *J. Comput. Appl. Math.* **236** (2011) 326–341.

[19] N.M. Missirlis and F. Tjaferis, Parallel matrix factorizations on a shared memory mimd computer. In: International Conference on Supercomputing. Vol. 297 of : *Lecture Notes in Computer Science*. Springer, Berlin-Heidelberg (1987) 926–938.

[20] OpenMP, The OpenMP API specification for parallel programming. [online] http://openmp.org (1997).

[21] PLASMA, [online] http://icl.cs.utk.edu/projectsfiles/plasma/html/htmlbrowsing/dtrsm.c.html (2009).

[22] H. Shioda, K. Konishi and S. Shin, Optimal task scheduling algorithm for parallel processing. In: *Proceedings of the 2011 2nd International Congress on Computer Applications and Computational Science*. Vol. 145 of: *Advances in Intelligent and Soft Computing*. Springer, Berlin-Heidelberg (2012) 79–87.

[23] C.F. Van Loan and G.H. Golub, Matrix Computations. Johns Hopkins University Press, Baltimore, MD, USA (1983).

[24] T. Wicky, E. Solomonik and T. Hoefler, Communication-avoiding parallel algorithms for solving triangular systems of linear equations. In: *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, Piscataway, NJ, USA (2017) 678–687.

[25] A. YarKhan, J. Kurzak, P. Luszczek and J. Dongarra, Porting the plasma numerical library to the openmp standard. *Int. J. Parallel Program.* **45** (2017) 612–633.