A Makespan Lower Bound for the Tiled Cholesky Factorization Based on ALAP Schedule

Olivier Beaumont[1(✉)], Julien Langou[2], Willy Quach[3], and Alena Shilova[1]

[1] Inria Bordeaux – Sud-Ouest and Université de Bordeaux, Bordeaux, France
olivier.beaumont@inria.fr
[2] University of Colorado Denver, Denver, USA
[3] Northeastern University, Boston, USA

Abstract. Due to the advent of multicore architectures and massive parallelism, the tiled Cholesky factorization algorithm has recently received plenty of attention and is often referenced by practitioners as a case study. However, we note that a theoretical study of the parallelism of this algorithm is currently lacking. In this paper, we present new theoretical results about the tiled Cholesky factorization in the context of a parallel homogeneous model without communication costs. By a careful analysis on the number of tasks of each type that run simultaneously in the ALAP (As Late As Possible) schedule without resource limitation, we are able to determine precisely an upper bound on the number of busy processors at any time (as degree 2 polynomials). We then use this information to find a closed form formula for a lower bound on the minimum time to schedule a tiled Cholesky factorization of size n on P processors. We show that this lower bound outperforms (is larger than) classical lower bounds from the literature. We also demonstrate that ALAP(P), an ALAP-based schedule where the number of resources is limited to P, has a makespan extremely close to the lower bound, thus establishing both the effectiveness of ALAP(P) schedule and of our new lower bound on the makespan.

Keywords: Scheduling · Cholesky factorization · CPU · GPU · Lower bounds

1 Introduction

A large fraction of time-consuming tasks performed on supercomputers are linear algebra operations. With the advent of multicore architectures and massive parallelism, it is therefore of particular interest to optimize and understand their parallel behavior. In this paper, we consider the problem of the dense tiled Cholesky factorization. The algorithm first splits the initial matrix into square

J. Langou—Supported by NSF award #1645514.

sub-matrices, or *tiles* of the same size. The tile size is chosen so as to achieve a good efficiency on the target architecture.

The tiled Cholesky factorization algorithm has recently received plenty of attention, either as an algorithm in itself [16,19] or as a case study for task-based schedulers [1,2,5,12,20,23]. Examples of task-based schedulers which have considered the scheduling of tiled Cholesky factorization are DAGuE [9], StarPU [4,13], SMPSs [21], and SuperMatrix [22]. Let us also note that OpenMP since 3.1 supports task-based parallelism. The tiled Cholesky factorization algorithm is also used in practice and is implemented in Dense Linear Algebra state of the art libraries, for example DPLASMA [8], FLAME [15], and PLASMA [10]. Recently, the practical design of good static schedule for heterogeneous resources has been considered in [3] and extensions to incomplete factorization [18], sparse matrices [17] have also been proposed.

One of our main goals in this paper is to obtain a tight theoretical lower bound on the parallel time to achieve a Cholesky factorization, based on the individual costs of the different kernels on a homogeneous architecture without communication cost. Trivial lower bounds can be derived from general bounds of the literature on scheduling. Specifically, the time to process Cholesky factorization is trivially bounded both by the length of the critical path (the longest path in the task graph from the source node to the sink node) and by the overall work divided by P, the number of available resources. To our best knowledge, no theoretical study on the execution time of any schedule for the tiled Cholesky factorization have been determined beyond these trivial bounds. Therefore, in many situations, it is impossible to assess the efficiency of a given schedule or implementation, because of the low quality of available lower bounds. This motivates this paper.

In this paper, we assume homogeneous processing units. While the heterogeneous setting is more general, establishing theoretical bounds in the heterogeneous case is much more difficult (see [6] for a recent survey in the case of two types of resources). We also make the assumption that communication cost is zero. We justify this assumption (no communication cost) in two ways. First, if the tile size is large enough, it is possible to overlap communications and computations. Indeed, if the dimension of the tile is $s \times s$, the tile (memory) size is s^2 while all kernels involved in Cholesky factorization have a complexity s^3. It has been shown experimentally using task-based schedulers [1,2,5,12,20] that it is possible to almost completely overlap communications and computations. Secondly, we note that the lower bound on the execution time also holds true in the case when communication costs are taken into account, so that any practical implementations will execute slower than this model. The lower bounds that we exhibit are not trivial and are relevant for practical applications, as demonstrated in Sect. 5. Another technical assumption is that we are assuming that the time to perform the SYRK operation is not larger than the time to perform the GEMM operation. This is a mild assumption. It is very likely to be true. One reason being that, if not, one can replace the SYRK kernels by GEMM kernels.

We can relate our work to the recent work of Agullo et al. [2] where the authors provide lower bound as well. The authors consider a more complicated model (heterogeneous) but rely on the linear programming formulation to find the schedule. We consider a simpler model (homogeneous) but we provide closed-form solutions and a tighter analysis. We are considering comparing both approaches as future work. We can also relate our work to the work of Cosnard, Marrakchi, Robert, and Trystram [14]. In this work, the authors study the scheduling of Gaussian elimination. A minor difference is that they work on LU while we work on Cholesky. The main difference is that they concentrate on an algorithm that works on the columns of the matrix, while our algorithm works on tiles.

In addition, it is of great interest to better understand how to efficiently schedule the parallel execution of the tiled Cholesky factorization algorithm. Indeed, even if a dynamic runtime scheduler is used, its behavior can be guided by priorities corresponding to a good static schedule in order to efficiently perform the parallel factorization, as shown in [3] in the context of StarPU. A contribution of our paper is to advocate the use of the ALAP (As Late As Possible) schedule where tasks are scheduled from the end as opposed from the start. We show that this simple heuristic turns out to provide results that are very close to the lower bound, therefore proving that it can be used in practice, for instance to fix priorities in a task based runtime scheduler.

The rest of the paper is organized as follows. In Sect. 2, tiled Cholesky factorization is presented. More specifically, we consider two different settings that correspond to different relative costs of the different kernels involved in tiled Cholesky factorization. We prove that these two cases are enough to cover all possible settings and typically correspond to the CPU and GPU settings and we provide the analysis of the critical path for each task. Then, in the case of the CPU case (Sect. 3.1) and to the GPU case (Sect. 3.2), we carefully analyze the number of tasks for every kernel at any instant of the factorization, when assuming an infinite number of processing resources. In turn, in Sect. 4, we prove that in the case of P processors, this analysis can be used to design a tight lower bound. In Sect. 5, we show using simulations that the makespan (the length) of the ALAP schedule with P processors is close to the theoretical bound, even for a small number of tiles, this demonstrates that the ALAP schedule is efficient and that the bound is tight. Concluding remarks and perspectives are finally proposed in Sect. 6.

2 Cholesky Factorization

2.1 Cholesky Algorithm

Given a Symmetric Positive Definite (SPD) matrix A, the Cholesky factorization computes a (lower) triangular matrix L such that $A = LL^T$. It is a core operation to solve linear systems in the case of SPD matrices as it allows to solve systems of the form $Ax = b$ by reducing it to computing solutions of $Ly = b$, and then $L^T x = y$. In order to compute the Cholesky factorization when using many processing units, the matrix A is split into $n \times n$ square tiles of size s, where s is chosen so as to perform kernels efficiently (as it improves data locality) and to allow to overlap communications and computations. Algorithm 1 depicts tiled Cholesky factorization.

Algorithm 1. Tiled Cholesky Factorization

```
for $k = 0$ to $n-1$ do
    $A_{k,k} \leftarrow POTRF(A_{k,k})$    $\{\text{POTRF}_k\}$
    for $i = k+1$ to $n-1$ do
        $A_{i,k} \leftarrow TRSM(A_{k,k}, A_{i,k})$    $\{\text{TRSM}_{i,k}\}$
    end for
    for $j = k+1$ to $n-1$ do
        $A_{j,j} \leftarrow SYRK(A_{j,k}, A_{j,j})$    $\{\text{SYRK}_{j,k}\}$
        for $i = j+1$ to $n-1$ do
            $Ai, j \leftarrow GEMM(A_{i,k}, A_{j,k})$    $\{\text{GEMM}_{i,j,k}\}$
        end for
    end for
end for
```

In Algorithm 1 and in the remainder of this paper, the tasks corresponding to POTRF kernels will be denoted as POTRF_i with $1 \leq i \leq n$ and correspond themselves to the Cholesky factorization of a real symmetric positive definite block of the matrix. The tasks corresponding to TRSM kernels will be denoted as $\text{TRSM}_{i,j}$ with $1 \leq j < i \leq n$ and correspond to the resolution of a triangular linear system of size s. The tasks corresponding to SYRK kernels will be denoted as $\text{SYRK}_{i,j}$ with $1 \leq j < i \leq n$ and correspond to a matrix multiplication with symmetric matrices, whereas the tasks corresponding to GEMM kernels, denoted as $\text{GEMM}_{i,j,k}$ with $1 \leq k < j < i \leq n$ correspond to general matrix product. Therefore, since we can always replace SYRK by GEMM, we will assume in the rest of the paper that the time to perform SYRK is at most the time to perform GEMM. The dependencies between the tasks are given by

- $\text{POTRF}_j \rightarrow \text{TRSM}_{i,j},\ j < i \leq n$;
- $\text{TRSM}_{i,j} \rightarrow \text{SYRK}_{i,j},\ j < i \leq n$; $\text{TRSM}_{i,j} \rightarrow \text{GEMM}_{i,k,j},\ j < k < i \leq n$;
- $\text{TRSM}_{i,j} \rightarrow \text{GEMM}_{k,i,j},\ j < i < k \leq n$;
- $\text{SYRK}_{i,j} \rightarrow \text{SYRK}_{i,j+1},\ j+1 < i \leq n$; $\text{SYRK}_{i,i-1} \rightarrow \text{POTRF}_i,\ 1 < i \leq n$;
- $\text{GEMM}_{i,j,j-1} \rightarrow \text{TRSM}_{i,j},\ 1 < j < i \leq n$;
- $\text{GEMM}_{i,j,k} \rightarrow \text{GEMM}_{i,j,k+1},\ k+1 < j < i \leq n$.

Table 1. Number of tasks of each type

Type of task	POTRF	SYRK	TRSM	GEMM
Number of tasks	n	$\frac{n(n-1)}{2}$	$\frac{n(n-1)}{2}$	$\frac{n(n-1)(n-2)}{6}$

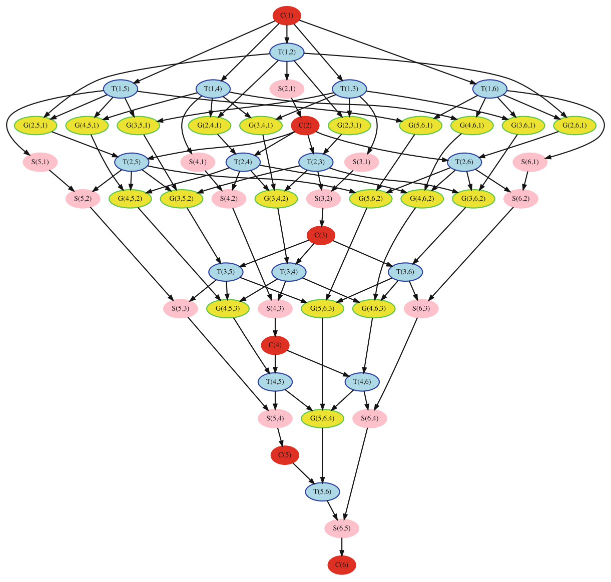

Fig. 1. DAG of a 6 × 6 Cholesky factorization

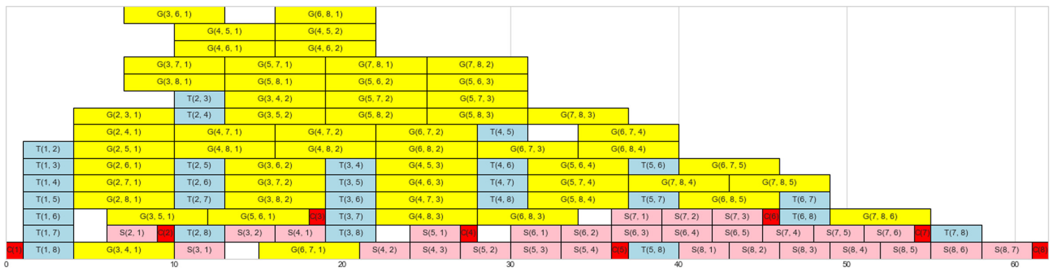

Fig. 2. ALAP schedule without resource limitation on 8 × 8 tiles with 1, 3, 3, 6 weights

Table 2. Kernel Performance (absolute and relative)

	POTRF	SYRK	TRSM	GEMM
GPU	11.55	1.277	3.420	1.733
CPU	11.27	47.76	44.02	87.60

time in ms

	POTRF	SYRK	TRSM	GEMM
GPU	1.00	0.11	0.30	0.15
CPU	1.00	4.24	3.91	7.77

ratio wrt POTRF

Figure 1 depicts the Directed Acyclic Graph (DAG) of the dependencies between the tasks of a 6 × 6 tiled Cholesky Factorization and the number of tasks for each kernel is given in the Table 1.

2.2 Kernel Performance

Table 2 (left part) describes the duration of individual tasks when $s = 960$ on an Intel Xeon E5-2680 (CPU) and an Nvidia GK110BGL GPU unit (GPU). All measurements were performed using Chameleon library [11], version 0.9.1.

We can observe that, with respect to CPU, GPUs are typically very fast for GEMM (an improvement of 50 with respect to CPU), fast for SYRK and TRSM (a respective improvement of 37 and 13) but relatively slow for POTRF (a slight slowdown). In Table 2 (right part), we give the relative duration of POTRF, TRSM, SYRK and GEMM with respect to POTRF. Note that throughout the paper, the results are stated in general terms and expressed as C, T, S and G respectively and we stress that our theoretical analysis below is valid for any values of (C, S, T, G). Nevertheless, by analyzing the critical paths as in Sects. 2.3 and 2.4, we can observe (with the additional trivial assumption $S \leq G$ discussed above) that there are only two different situations, depending on the respective values of $S + C$ and G. These two cases will be analyzed separately in the following. For convenience, we will denote them as CPU case (when $S + C \leq G$) and GPU case (when $S + C > G$) because each of these cases is quite emblematic of what can be encountered on a core or on an accelerator. In the same way, for convenience, for numerical illustrations, we will use $(1, 3, 3, 6)$ for (C, T, S, G) as emblematic values for a CPU and $(12, 3, 1, 2)$ for a GPU. These values are close to our experimental values and have also been used in the literature. Typically, $(1, 3, 3, 6)$ corresponds exactly to the ratios of the number of floating point operations for the different cores.

2.3 Critical Paths in the CPU Case, $S + C \leq G$

Based on the above described dependencies, we can compute the critical path for each task involved in the Cholesky factorization, *i.e.* the longest path from this node (itself included) to the end of the last task of the graph, *i.e.* POTRF(n) if $n \times n$ is the size of the matrix (expressed in number of tiles). Let us assume that $S + C \leq G$, this is the CPU case. In this case, in particular $S + C + T \leq G + T$, so that the edges SYRK$(i + 1, i) \rightarrow$ POTRF$(i + 1)$ are not part of the critical paths (except those starting at SYRK$(i + 1, i)$ nodes). Due to lack of space, we refer the reader to the companion research report [7] for the proofs and only detail the case of POTRF tasks in the CPU case.

– Case of POTRF$(i), 1 \leq i \leq n$ node: the critical path from POTRF$(i), i < n$ is given by POTRF$(i) \rightarrow$ (TRSM$(i, n) \rightarrow$ GEMM$(i + 1, n, i)) \rightarrow \ldots \rightarrow$ (TRSM$(n - 2, n) \rightarrow$ GEMM$(n - 1, n, n - 2)) \rightarrow$ TRSM$(n - 1, n) \rightarrow$ SYRK$(n, n - 1) \rightarrow$ POTRF(n). Its length is given by $L(C, i) = C + (n - i - 1)(T + G) + T + S + C$. Therefore, the overall Critical Path CP is given by $\mathrm{CP} = 2C + T + S + (n - 2)(T + G)$ and $L(C, i) = \mathrm{CP} - (i - 1)(T + G)$.
– Case of TRSM$(i, j), 1 \leq i < j \leq n$: $L(T, i, j) = \mathrm{CP} - C - (i - 1)(T + G)$.
– Case of SYRK$(i, j), 1 \leq j < i < n$: $L(S, i, j) = \mathrm{CP} - (i - 1)(T + G) + (i - j)S$.
– Case of SYRK$(n, j), 1 \leq j < n$: $L(S, n, j) = (n - j)S + C$.
– Case of GEMM$(i, j, k), 1 \leq k < i < j \leq n$:
$L(G, i, j, k) = \mathrm{CP} - C + G + T - iT - kG$.

2.4 Critical Paths in the GPU Case, $S + C \geq G$

Let us now consider the case when $C + S \geq G$, which corresponds to GPU situation. In this case, in particular $S + C + T \geq G + T$, so that SYRK$(i+1, i) \rightarrow$ POTRF$(i+1)$ are now used in critical paths.

– Case of POTRF$(i), 1 \leq i < n$:
  $L(C, i) = C + (n - i)(T + S + C)$. In particular, CP $= C + (n - 1)(T + S + C)$.
– Case of TRSM$(i, j), 1 \leq i < j \leq n$:
  $L(T, i, j) = (j - i - 1)(T + G) + (n - j + 1)(T + S + C)$.
– Case of SYRK$(i, j), 1 \leq j < i \leq n$:
  $L(S, i, j) = (i - j)S + C + (n - i)(T + S + C)$.
– Case of GEMM$(i, j, k), 1 \leq k < i < j \leq n$:
  $L(G, i, j, k) = (i - k)G + (j - i - 1)(T + G) + (n - j + 1)(T + S + C)$.

2.5 ALAP Schedule

Let us now define the ALAP schedule for the $n \times n$ tiled Cholesky factorization without resource limitation (the case with resource limitation will be considered in Sect. 5). In the ALAP schedule without resource limitation, we consider the Cholesky graph from the end, *i.e.* we reverse the task graph depicted in Fig. 1 and we schedule tasks in this order as soon as they are available. Therefore, ALAP on the original graph is simply the inverse of the ASAP schedule on the reversed graph. A first observation that can be made is that using the ALAP schedule without resource limitation, then every task starts its execution at a instant that differs from the makespan by exactly its critical path (as defined in Sects. 2.3 and 2.4) to the end of the schedule. We will denote in what follows the difference between the starting time of a task and the makespan as the distance of this task. Therefore, the ALAP schedule is optimal with an infinite number of processing resources and more specifically as soon as the number of processors is larger than a given threshold. Indeed, without resource limitation, the distance of the initial task is by construction the critical path of the Cholesky graph. In Sects. 3.1 (CPU case) and 3.2 (GPU case), we precisely evaluate the number of tasks of each type running at any instant of the ALAP schedule without resource limitation, and then we use these bounds to compute a lower bound on the execution time of any schedule in Sect. 4. Figure 2 depicts the execution of an ALAP schedule (without resource limitation) on a 8×8 tiled Cholesky factorization, with the time on the x-axis.

3 ALAP Schedule Analysis Without Resource Limitation

3.1 Case $S + C \leq G$

In the ALAP Schedule without resource limitation, each task T starts at time CP $- t_T$, where CP denotes the Critical Path of Cholesky factorization and t_T denotes the critical path from task T. In what follows, given an instant CP $- d$, our goal is to determine an upper bound on the number of tasks of each type and an upper bound on the work performed by the tasks of each type and whose execution terminates after the instant $\mathrm{CP}-d$.

We will denote respectively by

- $\#\mathrm{GEMM}(d), \#\mathrm{TRSM}(d), \#\mathrm{SYRK}(d)$ and $\#\mathrm{POTRF}(d)$ as upper bounds on the number of tasks of each type that are being processed at this instant $\mathrm{CP}-d$ using the ALAP schedule
- $W_{\mathrm{GEMM}}(d), W_{\mathrm{TRSM}}(d), W_{\mathrm{SYRK}}(d)$ and $W_{\mathrm{POTRF}}(d)$ as upper bounds on the work performed by tasks of each type whose execution terminates after the instant $\mathrm{CP}-d$

Both the number of tasks and the overall work will be used later in Theorem 1 to prove a lower bound. Due to the length of derivations, we refer the interested reader to [7] for complete formulas (in terms of n, C, S, T, G) and proofs. In the present paper, we provide the detailed analysis for $\#\mathrm{GEMM}(d)$. For the other cases, whose proofs are based on the same techniques, we only provide the explicit the explicit formulas.

Case of GEMM Tasks. Let us now establish the result for GEMM tasks. $\mathrm{GEMM}(i,j,k)$ runs at all instants such that $\mathrm{CP}-C+T-iT-kG \le d \le \mathrm{CP}-C+T-iT-kG+G$, so that in particular $\frac{\mathrm{CP}-d-C+T-iT}{G} \le k \le \frac{\mathrm{CP}-d-C+T-iT}{G}+1$ so that at most one value of k is possible, for a fixed pair (i,d), where $k = \left\lceil \frac{\mathrm{CP}-d-C+T}{G} - \frac{iT}{G} \right\rceil$.

In order to determine how many triplets (i,j,k) correspond to a tasks $\mathrm{GEMM}(i,j,k)$ running at time $\mathrm{CP}-d$, we need to check to consider the constraints on (i,j,k) valid triplets, *i.e.* $1 \le k < i < j \le n$.

- The first constraint states that $k \ge 1$. Using the above defined value for k, we can rewrite the condition
$$k \ge 1 \Leftrightarrow \frac{\mathrm{CP}-d-C+T}{G} \ge \frac{iT}{G} \Leftrightarrow i \le \frac{\mathrm{CP}-d-C+T}{T}.$$
This constraint can be rewritten as $i \le n + \frac{nG+C+S-2G-d}{T}$. Note that in particular, when d is small enough, *i.e.* $d \le nG+C+S-2G$, then above constraint becomes trivial and can be replaced by $i \le n$. Otherwise, if $d \ge nG+C+S-2G$, then the constraint becomes $i \le n - \lceil d-(nG+C+S-2G)/T \rceil$.
- The second constraint states that
$$k < i \Leftrightarrow \frac{\mathrm{CP}-d-C+T}{G} - \frac{iT}{G} \le (i-1) \Leftrightarrow \mathrm{CP}-d-C+T+G \le i(G+T).$$
This constraint can be rewritten as
$$(n-i-2)(T+G) \le d-(C+G+S+2T) \Leftrightarrow i \ge n - \lceil d-(C+S+T)/T+G \rceil.$$

Due to these constraints, we will obtain different formulas for the number of GEMMs, depending on the value of d.

– If $d \le (n-2)G + C + S + T = d_G$, then the only constraints are $i \ge n - \lceil \frac{d-(C+S+T)}{T+G} \rceil$ and $i < j \le n$ so that

$$\#\text{GEMM}(d) = \sum_{l=1}^{\lceil \frac{d-(C+S+T)}{T+G} \rceil} l = (\lceil \frac{d-(C+S+T)}{T+G} \rceil)(\lceil \frac{d-(C+S+T)}{T+G} \rceil + 1)/2,$$

$$\#\text{GEMM}(d) \le B_1^{\text{GEMM}} d^2 + C_1^{\text{GEMM}} d + D_1^{\text{GEMM}},$$

where $B_1^{\text{GEMM}} = \frac{1}{2(G+T)^2}$, $C_1^{\text{GEMM}} = \frac{(3G+T-2C-2S)}{2(G+T)^2}$ and $D_1^{\text{GEMM}} = \frac{(G-C-S)(2G+T-C-S)}{2(G+T)^2}$.

In order to estimate $W_{\text{GEMM}}(d)$, we rely on the integral of $\#\text{GEMM}(t)$ between 0 and d so that $W_{\text{GEMM}}(d) \le A_1^{\text{GEMM},W} d^3 + B_1^{\text{GEMM},W} d^2 + C_1^{\text{GEMM},W} d$, where $A_1^{\text{GEMM},W} = \frac{B_1^{\text{GEMM}}}{3}$, $B_1^{\text{GEMM},W} = \frac{C_1^{\text{GEMM}}}{2}$ and $C_1^{\text{GEMM},W} = D_1^{\text{GEMM}}$.

– If $d \ge \text{CP} - C - T$, then there is no GEMM task to perform (only TRSMs and one POTRF remain) and in this case, $\#\text{GEMM}(d) = 0$.

– If $d_G = (n-2)G + C + S + T \le d \le \text{CP} - C - T$, then the constraints are $n - \lceil \frac{d-(C+S+T)}{T+G} \rceil \le i \le n - \lceil \frac{d-(nG+C+S-2G)}{T} \rceil$ and $i < j \le n$, so that

$$\#\text{GEMM}(d) \le (B_2^{\text{GEMM}} d^2 + C_2^{\text{GEMM}} d + D_2^{\text{GEMM}}),$$

where $B_2^{\text{GEMM}} = \frac{1}{2(G+T)^2} - \frac{1}{2T^2}$, $C_2^{\text{GEMM}} = \frac{1}{2(T+G)} + \frac{1}{2T} - \frac{C+S-G}{(T+G)^2} + \frac{(n-2)G+C+S}{T^2}$ and $D_2^{\text{GEMM}} = 1 - \frac{C+S+T}{2(T+G)} - \frac{(n-2)G+C+S}{2T} + \frac{(C+S-G)^2}{2(T+G)^2} - \frac{((n-2)G+C+S)^2}{2T^2}$

In order to estimate $W_{\text{GEMM}}(d)$, we rely on the integral of $\#\text{GEMM}(t)$ between d_G and d plus $W_{\text{GEMM}}(d_G)$ so that

$$W_{\text{GEMM}}(d) \le A_2^{\text{GEMM},W} d^3 + B_2^{\text{GEMM},W} d^2 + C_2^{\text{GEMM},W} d + D_2^{\text{GEMM},W},$$

where $A_2^{\text{GEMM},W} = \frac{B_2^{\text{GEMM}}}{3}$, $B_2^{\text{GEMM},W} = \frac{C_2^{\text{GEMM}}}{2}$, $C_2^{\text{GEMM},W} = D_2^{\text{GEMM}}$ and $D_2^{\text{GEMM},W} = (A_1^{\text{GEMM},W} - A_2^{\text{GEMM},W}) d_G^3 + (B_1^{\text{GEMM},W} - B_2^{\text{GEMM},W}) d_G^2 + (C_1^{\text{GEMM},W} - C_2^{\text{GEMM},W}) d_G$.

Case of POTRF Tasks. Clearly, at any instant, at most one POTRF task can be running since there is a dependency path $\text{POTRF}(i) \longrightarrow \text{TRSM}(i, i+1) \longrightarrow \text{SYRK}(i+1, i) \longrightarrow \text{POTRF}(i+1)$, therefore $\forall d \ge 0$, $\#\text{POTRF}(d) \le 1$ and the total amount of work done after $\text{CP} - d$ is defined by $\forall d \ge 0$, $W_{\text{POTRF}}(d) \le C^{\text{POTRF},W} d + D^{\text{POTRF},W}$, where $C^{\text{POTRF},W} = \frac{C}{T+G}$ and $D^{\text{POTRF},W} = \frac{C(2G+T-S-C)}{T+G}$.

Case of TRSM Tasks. $\text{TRSM}(i,j)$ runs at all instants such that $\text{CP} - C - (i-1)(T+G) - T \le d \le \text{CP} - C - (i-1)(T+G)$. From above inequalities, we can prove [7] that the amount of the TRSM tasks running at the time $\text{CP} - d$ is either 0 or $\#\text{TRSM}(d) = \left\lceil \frac{d-S-C+G}{T+G} \right\rceil \leq C^{\text{TRSM}} d + D^{\text{TRSM}}$, where $C^{\text{TRSM}} = \frac{1}{T+G}$ and $D^{\text{TRSM}} = \frac{2G-S-C+T}{T+G}$. and $W_{\text{TRSM}}(d) \leq B^{\text{TRSM},W} d^2 + C^{\text{TRSM},W} d + D^{\text{TRSM},W}$, where $B^{\text{TRSM},W} = \frac{T}{2(T+G)^2}$, $C^{\text{TRSM},W} = \frac{T(3T+3G-2S-2C)}{2(T+G)^2}$ and $D^{\text{TRSM},W} = \frac{T(T+G-S-C)(2T+2G-S-C)}{2(T+G)^2}$.

Case of SYRK Tasks. Clearly, at any instant, at most one $\text{SYRK}(n, j)$ task can be running since there is a dependency path $\text{SYRK}(n, j) \longrightarrow \text{SYRK}(n, j+1)$, so that $\forall d \geq 0,\ \#\text{SYRK}(n, j, d) \leq 1$.

Let us now consider the case of tasks $\text{SYRK}(i, j)$ for $1 \leq j < i < n$. $\text{SYRK}(i, j)$ runs at all instants such that $\text{CP} + (T+G) - i(T+G-S) - jS - S \leq d \leq \text{CP} + (T+G) - i(T+G-S) - jS$ From above inequalities, we can prove [7] that

- If $d \leq (n-1)S + 2C + T = d_S$, then $\#\text{SYRK}(d) \leq C_1^{\text{SYRK}} d + D_1^{\text{SYRK}}$, where $C_1^{\text{SYRK}} = \frac{1}{T+G}$ and $D_1^{\text{SYRK}} = \frac{G-2C-S}{T+G}$. Similarly, we obtain that $W_{\text{SYRK}}(d) \leq B_1^{\text{SYRK},W} d^2 + C_1^{\text{SYRK},W} d + D_1^{\text{SYRK},W}$, where $B_1^{\text{SYRK},W} = \frac{C_1^{\text{SYRK}}}{2}$, $C_1^{\text{SYRK},W} = D_1^{\text{SYRK}} + 1$ and $D_1^{\text{SYRK},W} = -C$.
- If $d > (n-1)S + 2C + T = d_S$, then $\forall d > nS + 2C - G, \#\text{SYRK}(d) \leq C_2^{\text{SYRK}} d + D_2^{\text{SYRK}}$, where $C_2^{\text{SYRK}} = \frac{-S}{(T+G)(T+G-S)}$ and $D_2^{\text{SYRK}} = 1 + \frac{S((n-1)(T+G)-G+2C+S)}{(T+G)(T+G-S)}$. Similarly, $W_{\text{SYRK}}(d) \leq B_2^{\text{SYRK},W} d^2 + C_2^{\text{SYRK},W} d + D_2^{\text{SYRK},W}$, where $B_2^{\text{SYRK},W} = \frac{C_2^{\text{SYRK}}}{2}$, $C_2^{\text{SYRK},W} = D_2^{\text{SYRK}}$ and $D_2^{\text{SYRK},W} = (B_1^{\text{SYRK},W} - B_2^{\text{SYRK},W}) d_S^2 + (C_1^{\text{SYRK},W} - C_2^{\text{SYRK},W}) d_S + (n-1)S$.

3.2 Case $S + C \geq G$

We can establish the same results in the GPU case, using the same type of proof techniques than in the case of GEMM tasks when $S + C \leq G$. We refer the interested reader to [7], where all detailed proofs are presented, and we just summarize results below.

Case of POTRF Tasks. $\forall d \geq 0$, $\#\text{POTRF}(d) \leq 1$. and $\forall d \geq 0$, $W_{\text{POTRF}}(d) \leq C^{\text{POTRF},W} d + D^{\text{POTRF},W}$, where $C^{\text{POTRF},W} = \frac{C}{T+S+C}$ and $D^{\text{POTRF},W} = C$.

Case of TRSM Tasks. Let $d_T \leq (n-1)(T+G) + C + S - G$. Then

- When $d \leq d_T$, then $\#\text{TRSM}(d) = \left\lfloor \frac{d+G-C-S}{C+S+T} \right\rfloor + 1 \leq \frac{d+G+T}{C+S+T} = C_1^{\text{TRSM}} d + D_1^{\text{TRSM}}$ where $C_1^{\text{TRSM}} = \frac{1}{C+S+T}$ and $D_1^{\text{TRSM}} = \frac{G+T}{C+S+T}$ and $W_{\text{TRSM}}(d) \leq \frac{C_1^{\text{TRSM}}}{2} d^2 + D_1^{\text{TRSM}} d$.

– When $d \geq d_T$, then $\#\text{TRSM}(d) =\leq C_2^{\text{TRSM}} d + D_2^{\text{TRSM}}$ where $C_2^{\text{TRSM}} = -\frac{T+G}{(C+S-G)(C+S+T)}$ and $D_2^{\text{TRSM}} = 2 + \frac{(n-1)(T+G)}{C+S-G} - \frac{C+S-G}{C+S+T}$ and $W_{\text{TRSM}}(d) \leq B^{\text{TRSM},W} d^2 + C^{\text{TRSM},W} d + D^{\text{TRSM},W}$ where $B^{\text{TRSM},W} = \frac{C_2^{\text{TRSM}}}{2}$, $C^{\text{TRSM},W} = D_2^{\text{TRSM}}$ and $D^{\text{TRSM},W} = \frac{C_1^{\text{TRSM}} - C_2^{\text{TRSM}}}{2} d_T^2 + (D_1^{\text{TRSM}} - D_2^{\text{TRSM}}) d_T$.

Case of SYRK Tasks. $\forall d \geq 0$, $\#\text{SYRK}(d) \leq C^{\text{SYRK}} d + D^{\text{SYRK}}$, where $C^{\text{SYRK}} = \frac{-S}{(C+S+T)(C+T)}$ and $D^{\text{SYRK}} = 1 + \frac{(n-1)S}{(C+T)} + \frac{CS}{(C+S+T)(C+T)}$ and $W_{\text{SYRK}} \leq \frac{C^{\text{SYRK}}}{2} d^2 + C^{\text{SYRK}} d$.

Case of GEMM Tasks. Let $d_G \leq nG + S + C - 2G$. Then

– When $d \geq d_G$, then $\#\text{GEMM}(d) \leq B_1^{\text{GEMM}} d^2 + C_1^{\text{GEMM}} + D_1^{\text{GEMM}}$, and $W_{\text{GEMM}} \leq \frac{B_1^{\text{GEMM}}}{3} d^3 + \frac{C_1^{\text{GEMM}}}{2} d^2 + D_1^{\text{GEMM}} d$, where $B_1^{\text{GEMM}} = \frac{1}{2(G+S+T)(T+G)}$, $C_1^{\text{GEMM}} = \frac{3T+4G+S}{2(G+S+T)(T+G)}$ and $D_1^{\text{GEMM}} = 1$
– When $d_G \leq d \leq n(G+T) + S + C - 2G$, then $\#\text{GEMM}(d) \leq B_2^{\text{GEMM}} d^2 + C_2^{\text{GEMM}} + D_2^{\text{GEMM}}$, where $B_2^{\text{GEMM}} = \frac{1}{2(G+S+T)(T+G)} - \frac{1}{2(T+S+C-G)(T)}$, $C_2^{\text{GEMM}} = \frac{3T+4G+S}{2(G+S+T)(T+G)} - \frac{nG}{2(T+S+C-G)(T)}$ and $D_2^{\text{GEMM}} = 1 - \frac{n^2G^2}{2(T+S+C-G)(T)}$ and $W_{\text{GEMM}} \leq \frac{B_2^{\text{GEMM}}}{3}(d^3 - d_1^3) + \frac{C_2^{\text{GEMM}}}{2}(d^2 - d_1^2) + D_2^{\text{GEMM}}(d - d_1) + \frac{B_1^{\text{GEMM}}}{3} d_1^3 + \frac{C_1^{\text{GEMM}}}{2} d_1^2 + D_1^{\text{GEMM}} d_1$, where $d_1 = nG + S + C - 2G$.

4 Lower Bound for Cholesky with P Resources

4.1 CPU Case, $S + C \leq G$

Using the above bounds on the number of tasks, we can bound, for any distance d to CP the number of tasks that would be processed simultaneously using the ALAP schedule without resource limitation. The upper bound on the overall number of tasks $f_\#(t)$ processed at any instant t, $0 \leq t \leq \text{CP}$ is therefore given as a degree 2 polynomial, whose coefficients depend on whether $t \leq \text{CP} - d_G$, $\text{CP} - d_G < t \leq \text{CP} - d_S$ and $t > \text{CP} - d_S$ (where d_G and d_S are defined in Sect. 3.1). Similarly, let us denote by $f_W(t)$ the upper bound on the work performed by ALAP schedule after instant t. $f_W(t)$ is given as a degree 3 polynomial, whose coefficients depend on whether $t \leq \text{CP} - d_G$, $\text{CP} - d_G < t \leq \text{CP} - d_S$ and $t > \text{CP} - d_S$.

Figure 3 displays the upper bound on the overall number of tasks processed at any instant t, $0 \leq t \leq \text{CP}$, and the same information for each type of task, GEMM, TRSM, SYRK. All plots correspond to the case where $G = 6$, $T = S = 3$ and $C = 1$, that corresponds to our sample model for a CPU node. Due to lack of space, we refer the interested reader to companion research report [7] to find the counterparts of Fig. 3 in the GPU case.

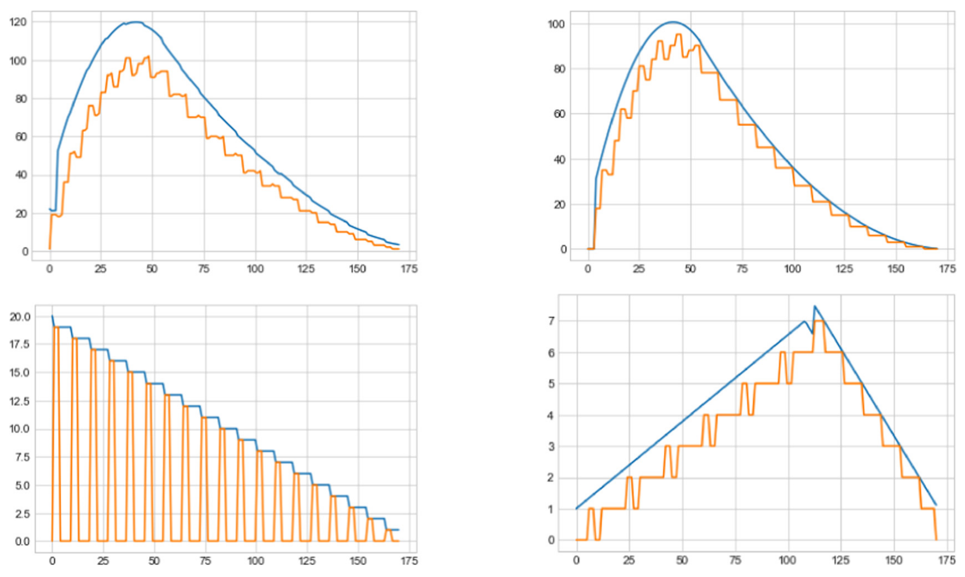

Fig. 3. Evolution of the number of tasks of each type with d, $n = 20$, CPU case, for all instants between 0 and 170 (the length of the critical path). Subfigures depict the overall number of tasks (left, up), GEMM tasks (right, up), TRSM tasks (left, down) and SYRK tasks (right, down)

4.2 Lower Bounds

Let us define t_P as the largest instant such that $f_{\#}(t) \leq P$ for any $t \geq t_P$, or $t_P = \text{CP}$ if the number of resources is large enough. This instant can be determined easily by studying $f_{\#}(t)$, which is described as a degree 2 polynomial on several intervals. As we have seen above, both $f_{\#}(t)$ and the set of intervals to be considered depend only whether $S + C \leq G$ (CPU case) or $S + C \geq G$ (GPU case).

Lemma 1. *Let us denote by $\mathcal{S}$ any valid schedule with P processors. Then, $\mathcal{S}$ cannot perform more work between* $\textsc{Makespan}(\mathcal{S}) - (\text{CP} - t_P)$ *and* $\textsc{Makespan}(\mathcal{S})$ *than ALAP(P) and this amount of work is upper bounded by $f_W(t_P)$.*

Proof. Intuitively, no schedule can perform more tasks during the last $\text{CP} - t_P$ instants. Indeed, during these instants, all the tasks whose critical path is less than t_P are processed using ALAP. Moreover, no other task can start as close to the CP in any schedule. $f_{\#}(t)$ (resp. $f_W(t)$) and is an upper bound on the number of tasks (resp. the overall work) processed simultaneously at time t by ALAP schedule without resource limitation. Moreover $\text{CP} - t_P$ is the largest instant where the ALAP schedule without resource limitation and with at most P processors coincide, so that we can upper bound the work performed by any schedule (by optimality of ALAP after $\text{CP} - t_P$) by $f_W(t_P)$. This finishes the proof of the lemma. $\square$

Theorem 1. *A makespan lower bound for any schedule is* $(\mathrm{CP} - t_P) + \frac{W - f_W(t_P)}{P}$.

Proof. The overall work W to perform for Cholesky factorization is given by $W = nC + \frac{n(n-1)}{2}(S + T) + \frac{n(n-1)(n-2)}{6}G$. In any schedule $\mathcal{S}$, we have proved in Lemma 1 that an upper bound for the amount of work W_{END} that can be processed during the last $\mathrm{CP} - t_P$ time units is $f_W(t_P)$. Similarly, a trivial upper bound for the amount of work W_{BEGIN} processed during the first $\textsc{Makespan}(\mathcal{S}) - (\mathrm{CP} - T_p)$ time units is $P(\textsc{Makespan}(\mathcal{S}) - (\mathrm{CP} - T_p))$, so that $W = nC + n(n-1)/2 * (S + T) + n(n-1)(n-2)/6 * G = W_{\text{BEGIN}} + W_{\text{END}} \leq f_W(t_P) + P(\textsc{Makespan}(\mathcal{S}) - (\mathrm{CP} - T_p))$ and

$$\textsc{Makespan}(\mathcal{S}) \leq \frac{W - f_W(t_P)}{P} + (\mathrm{CP} - T_p) \quad \square$$

$\square$

5 Simulation Results

In the above sections, we have established a theoretical lower bound on the time necessary to achieve a Cholesky factorization on an homogeneous platform consisting of P GPUs or P CPUs. This lower bound was established using a detailed analysis of the ALAP schedule and we expect this bound to be close to the makespan achieved by ALAP. Our goal in this section is to establish through simulation our intuition.

We performed simulations with different problem sizes ($n = 30$ or 40 and two different configurations of tasks lengths corresponding either to the CPU case ($G = 6$, $C = 1$, $S = T = 3$) or to the GPU case ($G = 2$, $C = 12$, $S = 1$ and $T = 3$) in Fig. 4. In this simulation, we plot the speedup achieved by the different heuristics against theoretical bounds. The first theoretical (trivial) bound on the achievable speedup on P processors is $\min(P, W/CP)$ **(red)**. The second bound is the one established in Sect. 4, based on a detailed analysis of ALAP schedule for Cholesky factorization **(green)**.

We consider the following heuristics:

- **ALAP (blue)** is the heuristic that we described in Sect. 2.5 when there is no resource limitation. In the presence of resource limitations, when at a given moment, the number of available tasks is greater than the number of available resources, we define the highest priority task as the one that maximizes the length of the longest path between POTRF(1) and this task.
- **ASAP (yellow)** As Soon As Possible, is the dual heuristic with respect to ALAP. Tasks are processed as soon as they become ready when there is no resource limitation. In the presence of resource limitations, when at a given moment, the number of available tasks is greater than the number of available resources, we define the highest priority task as the one that maximizes the length of the longest path between this task and POTRF(n).

– **LAPACK (purple)** corresponds to the Cholesky factorization implemented in the LAPACK library. It consists in n bulk-synchronized steps. During step i, POTRF(i) is first performed, then all TRSM(i, j) tasks and finally all SYRK(j, i) tasks and all GEMM(j, k, i) tasks can be interleaved and can be executed concurrently if enough resources are available.

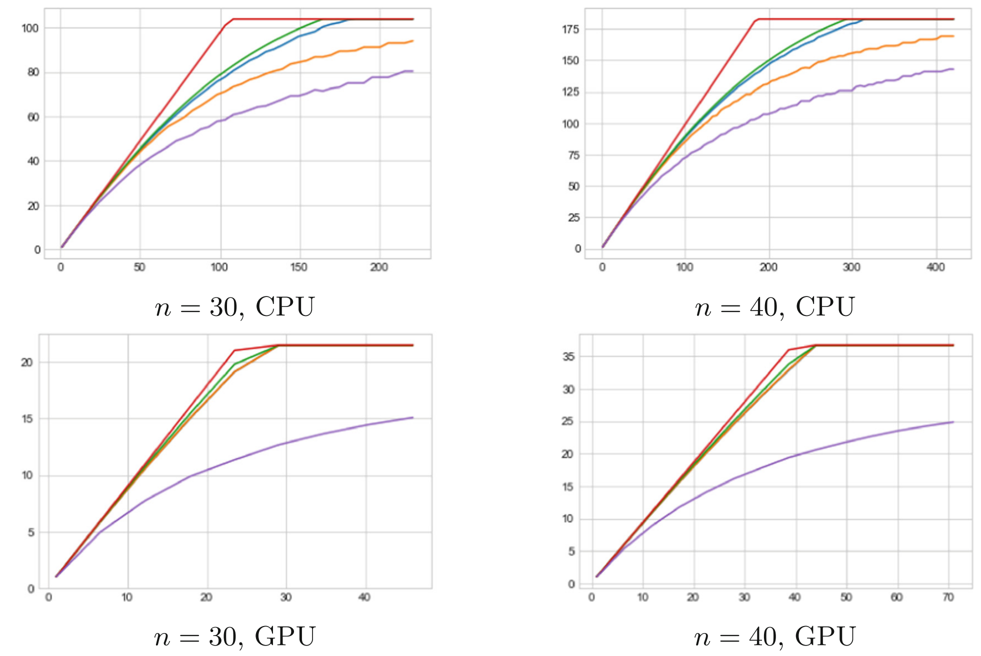

Fig. 4. Evolution of speedup with the number of processors P with CPU weights: ($C = 1$, $S = 3$, $T = 3$, $G = 6$) on the top, and GPU weights: ($C = 12$, $S = 1$, $T = 3$, $G = 2$) on the bottom. (Color figure online)

We note that, since we plot speedup, our lower bounds on the makespan become upper bounds on the speedup. We see that our new upper bound on the speedup (green) is lower than the trivial bound (red). Another observation is that the length of the ALAP schedule (blue) and our new lower bound (green) are always extremely close. This confirms the tightness of our analysis and the excellent performance of the ALAP schedule. In the companion research report [7], the reader will find more results ($n = 20$ in particular) and also an asymptotic analysis which suggests that ALAP is uniformly asymptotically optimal as the problem size becomes larger. In other words, as n increases, the maximum ratio over all P between the ALAP schedule makespan and our lower bound uniformly tends to 1. For example, we can observe that, using either our model GPU weights or our model CPU weights as soon as n gets larger than 40, for any processor count P, ALAP is at least 5% optimal. And, as n increases, ALAP approaches optimality (for any processor count P).

6 Conclusion and Perspectives

In this paper, we have studied in detail the makespan of Cholesky's factorization on a homogeneous platform. For example, this platform can be made of GPUs only, or CPUs only, or anything really. We have obtained a sharp lower bound on the completion time of the factorization, regardless of the scheduling used, which is based on a detailed study of the ALAP schedule. In particular, this bound requires determining the number of simultaneous tasks of each type at any instant in the ALAP schedule.

This lower bound allows us to make several observations. First of all, ALAP scheduling behaves remarkably well in the case of CPUs as in the case of GPUs, always significantly better than the LAPACK schedule and better than ASAP scheduling in the case of CPUs. The proximity between the ALAP completion time and the lower bound, in all the scenarios, allows us to accurately estimate the time required for Cholesky factorization. Indeed, the ALAP completion time provides an upper bound on the time needed whereas the theoretical lower bound provides a lower bound. The proximity between the two thus guarantees both the quality of the approximation of the time needed to perform the factorization, the quality of the theoretical lower bound and the quality of the ALAP scheduling which provides the upper bound.

This work opens many perspectives. From a theoretical point of view, the generalization of the technique used in the case of Cholesky factorization to other types of task graphs, in linear algebra and elsewhere, is open. The techniques used in this paper are highly computational and the results are technically quite complex, but generalization and automation may be envisaged. Another interesting issue is the possibility to extend these results to heterogeneous platforms. Indeed, it has been observed using dynamic runtime schedulers, typically on Cholesky factorization, that heterogeneity allows an "optimal" use of resources, by executing tasks on the most suitable type of resources. Unfortunately, in the heterogeneous case, the known lower bounds are extremely coarse and do not allow to assess the closeness to optimality of a schedule. This raises the question on whether our approach can help.

References

1. Agullo, E., Hadri, B., Ltaief, H., Dongarra, J.: Comparative study of one-sided factorizations with multiple software packages on multi-core hardware. In: ACM/IEEE Conference on Supercomputing, Portland, OR, SC 2009, November 2009. https://doi.org/10.1145/1654059.1654080
2. Agullo, E., et al.: Bridging the gap between performance and bounds of Cholesky factorization on heterogeneous platforms. In: HCW 2015 (2015). https://doi.org/10.1109/IPDPSW.2015.35
3. Agullo, E., Beaumont, O., Eyraud-Dubois, L., Kumar, S.: Are static schedules so bad? A case study on Cholesky factorization. In: 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 1021–1030. IEEE (2016). https://doi.org/10.1109/IPDPS.2016.90

4. Augonnet, C., Thibault, S., Namyst, R., Wacrenier, P.A.: StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. Concurr. Comput. Pract. Exp. **23**, 187–198 (2011). https://doi.org/10.1002/cpe.1631. Special Issue: Euro-Par 2009
5. Badia, R.M., Herrero, J.R., Labarta, J., Pérez, J.M., Quintana-Ortí, E.S., Quintana-Ortí, G.: Parallelizing dense and banded linear algebra libraries using SMPSs. Concurr. Comput. Pract. Exp. **21**(18), 2438–2456 (2009). https://doi.org/10.1002/cpe.1463
6. Beaumont, O., et al.: Scheduling on two types of resources: a survey. arXiv preprint arXiv:1909.11365 (2019)
7. Beaumont, O., Langou, J., Quach, W., Shilova, A.: A Makespan Lower Bound for the Scheduling of the Tiled Cholesky Factorization based on ALAP Schedule, February 2020. https://hal.inria.fr/hal-02487920, working paper or preprint
8. Bosilca, G., et al.: Flexible development of dense linear algebra algorithms on massively parallel architectures with DPLASMA. In: IEEE International Symposium on Parallel and Distributed Processing Workshops, pp. 1432–1441, May 2011. https://doi.org/10.1109/IPDPS.2011.299
9. Bosilca, G., Bouteiller, A., Danalis, A., Herault, T., Lemarinier, P., Dongarra, J.: DAGuE: a generic distributed DAG engine for high performance computing. Parallel Comput. **38**(1–2), 37–51 (2012). https://doi.org/10.1016/j.parco.2011.10.003
10. Buttari, A., Langou, J., Kurzak, J., Dongarra, J.: A class of parallel tiled linear algebra algorithms for multicore architectures. Parallel Comput. **35**(1), 38–53 (2009). https://doi.org/10.1016/j.parco.2008.10.002. http://www.sciencedirect.com/science/article/pii/S0167819108001117
11. Chameleon: a dense linear algebra software for heterogeneous architectures (2014). https://project.inria.fr/chameleon. Accessed June 2020
12. Chan, E., Van Zee, F.G., Bientinesi, P., Quintana-Ortí, E.S., Quintana-Ortí, G., van de Geijn, R.: SuperMatrix: a multithreaded runtime scheduling system for algorithms-by-blocks. In: PPoPP 2008, pp. 123–132. ACM (2008). https://doi.org/10.1145/1345206.1345227
13. Cojean, T., Guermouche, A., Hugo, A., Namyst, R., Wacrenier, P.A.: Resource aggregation for task-based Cholesky factorization on top of modern architectures. Parallel Comput. **83**, 73–92 (2019). https://doi.org/10.1016/j.parco.2018.10.007
14. Cosnard, M., Marrakchi, M., Robert, Y., Trystram, D.: Parallel Gaussian elimination on an MIMD computer. Parallel Comput. **6**(3), 275–296 (1988). https://doi.org/10.1016/0167-8191(88)90070-1
15. Gunnels, J.A., Gustavson, F.G., Henry, G.M., van de Geijn, R.A.: Flame: formal linear algebra methods environment. ACM Trans. Math. Softw. **27**(4), 422–455 (2001). https://doi.org/10.1145/504210.504213
16. Gustavson, F., Karlsson, L., Kågström, B.: Distributed SBP Cholesky factorization algorithms with near-optimal scheduling. ACM Trans. Math. Softw. **36**(2), 1–25 (2009). https://doi.org/10.1145/1499096.1499100
17. Jacquelin, M., Zheng, Y., Ng, E., Yelick, K.: An asynchronous task-based fan-both sparse Cholesky solver. SIAM J. Sci. and Stat. Comput. **9**(2), 327–340 (2016). https://doi.org/10.1137/1.9781611975215.8
18. Kim, K., Rajamanickam, S., Stelle, G., Edwards, H.C., Olivier, S.L.: Task parallel incomplete Cholesky factorization using 2D partitioned-block layout. arXiv preprint arXiv:1601.05871 (2016)

19. Kurzak, J., Buttari, A., Dongarra, J.: Solving systems of linear equations on the CELL processor using Cholesky factorization. IEEE Trans. Parallel Distrib. Syst. **19**(9), 1175–1186 (2008). https://doi.org/10.1109/TPDS.2007.70813
20. Kurzak, J., Ltaief, H., Dongarra, J., Badia, R.M.: Scheduling dense linear algebra operations on multicore processors. Concurr. Comput. Pract. Exp. **22**(1), 15–44 (2010). https://doi.org/10.1002/cpe.1467
21. Pérez, J.M., Badia, R.M., Labarta, J.: A flexible and portable programming model for SMP and multi-cores. Technical report, Barcelona Supercomputing Center - Centro Nacional de Supercomputacioń, June 2007
22. Quintana-Ortí, E.S., Quintana-Ortí, G., van de Geijn, R.A., Van Zee, F.G., Chan, E.: Programming matrix algorithms-by-blocks for thread-level parallelism. ACM Trans. Math. Softw. **36**(3) (2009). https://doi.org/10.1145/1527286.1527288
23. Song, F., YarKhan, A., Dongarra, J.: Dynamic task scheduling for linear algebra algorithms on distributed-memory multicore systems. In: SC 2009 (2009). https://doi.org/10.1145/1654059.1654079