A Makespan Lower Bound for the Scheduling of the Tiled Cholesky Factorization based on ALAP Schedule

Olivier Beaumont[1,2] Julien Langou[3] Willy Quach[4]
Alena Shilova[1,2]

(1) Inria Bordeaux – Sud-Ouest, Bordeaux, France
(2) Université de Bordeaux, Bordeaux, France
(3) Univ. of Colorado Denver, USA
(4) Northeastern Univ., USA

February 21, 2020

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. It is also implemented in mainstream dense linear algebra libraries and is used as a testbed for runtime systems. However, we note that 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. Based on the relative costs of involved kernels, we prove that only two different situations must be considered, typically corresponding to CPUs and GPUs. 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 the number of busy processors at any time (as degree 2 polynomials). We then use this information to find a closed form formula for the minimum time to schedule a tiled Cholesky factorization of size n on P processors. We show that this bound outperforms classical bounds from the literature. We also prove 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 proving both the effectiveness of ALAP(P) schedule and of the lower bound on the makespan.

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 of particular interest to optimize and understand their parallel behavior. In this paper, we consider the problem of the dense tiled Cholesky factorization on any type of architectures, in particular both for CPUs and GPUs. The algorithm first splits the initial matrix into square sub-matrices, or *tiles* of the same size. The tile size is chosen so as to achieve a good efficiency on the target architecture. A large tile size also favors the overlapping of communications by computations, since 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 for task-based schedulers [8, 3, 1, 16, 5], it is indeed possible to almost completely overlap communications and computations.

The tiled Cholesky factorization algorithm has recently received plenty of attention, either as an algorithm in itself [12, 15] or as a case study for task-based schedulers [8, 3, 21, 1, 16, 5]. Examples of task-based schedulers which have produced papers about the scheduling of tiled Cholesky factorization are for example DAGuE [7], KAAPI [11], QUARK [22], StarPU [4, 9], SMPSs [18], and SuperMatrix [19]. 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, FLAME, and PLASMA. Recently, the practical design of good static schedule for heterogeneous resources has been considered in [2] and extensions to incomplete factorization [14], sparse matrices [13] have also been proposed.

Our goal 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. 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 (was 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.

It is therefore 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 [2] in the context of StarPU. In this paper, we assume homogeneous processing units. While we acknowledge that the heterogeneous setting is more general, establishing theoretical bounds in this case is much more difficult (see [6] for a recent survey in the case of two types of resources). As previously mentioned, we also assume that the tile size is large enough so that it is possible to overlap communications and computations. As already mentioned, this assumption has been shown to be realistic when using dynamic schedulers and sufficiently large tile sizes. 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 Section 6.

We can relate our work to the recent work of Agullo et al. [1] 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. Another 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.

We can also relate our work to the work of Cosnard, Marrakchi, Robert, and Trystram [10, 20, 17]. In this work, the authors study the Gaussian elimination. The main difference is that they concentrate on the BLAS algorithm that works on the columns of the matrix, which is an easier problem. This algorithm was popular due to vectorization, but nowadays tiled algorithms are much

more relevant.

The rest of the paper is organized as follows. In Section 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 (Section 3) and to the GPU case(Section 4), 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, we prove that in the case of P processors, this analysis can be used to design efficient and tight bounds in Section 5. In Section 6, we prove using simulations that the makespan (the length) of the ALAP schedule with P processors is very close to the theoretical bound, even for a small number of tiles, what proves that the ALAP schedule is very efficient and that the bound is tight. Concluding remarks and perspectives are finally proposed in Section 7.

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 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} ← POTRF(A_{k,k})            {C_k}
  for i = k + 1 to n − 1 do
    A_{i,k} ← TRSM(A_{k,k}, A_{i,k})  {T_{i,k}}
  end for
  for j = k + 1 to n − 1 do
    A_{j,j} ← SYRK(A_{j,k}, A_{j,j})  {S_{j,k}}
    for i = j + 1 to n − 1 do
      Ai, j ← GEMM(A_{i,k}, A_{j,k})  {G_{i,j,k}}
    end for
  end for
end for
```

In the remainder of this paper, the tasks corresponding to POTRF kernels will be denoted as C_i with $1 \leq i \leq n$, the tasks corresponding to TRSM kernels will be denoted as $T_{i,j}$ with $1 \leq j < i \leq n$, the tasks corresponding to SYRK kernels will be denoted as $S_{i,j}$ with $1 \leq j < i \leq n$, and finally the the tasks corresponding to GEMM kernels will be denoted as $G_{i,j,k}$ with $1 \leq k < j < i \leq n$. The dependencies between the tasks are given by

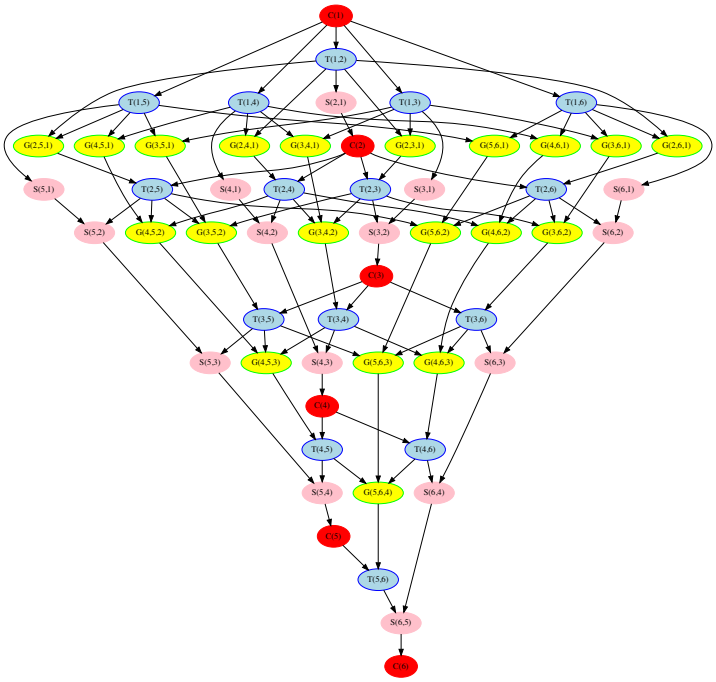

Figure 1: DAG of a 6×6 Cholesky factorization

Table 1: Number of tasks

Type of task	C	S	T	G
Number of tasks	n	$\frac{n(n-1)}{2}$	$\frac{n(n-1)}{2}$	$\frac{n(n-1)(n-2)}{6}$

- $C_j \rightarrow T_{i,j}, j < i \leq n$;
- $T_{i,j} \rightarrow S_{i,j}, j < i \leq n$;
- $T_{i,j} \rightarrow G_{i,k,j}, j < k < i \leq n$;
- $T_{i,j} \rightarrow G_{k,i,j}, j < i < k \leq n$;
- $S_{i,j} \rightarrow S_{i,j+1}, j + 1 < i \leq n$;
- $S_{i,i-1} \rightarrow C_i, 1 < i \leq n$;
- $G_{i,j,j-1} \rightarrow T_{i,j}, 1 < j < i \leq n$;
- $G_{i,j,k} \rightarrow G_{i,j,k+1}, k + 1 < j < i \leq n$.

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 Table 1.

Figure 2: Speeds of the different kernels on typical CPU and GPU nodes

	gemm	potrf	syrk	trsm
GPU	1.733	11.55	1.277	3.420
CPU	87.60	11.27	47.76	44.02
ratio	50.5	0.975	37.4	12.9

2.2 Kernel Performance

We will consider two main cases in our analysis, according to the relative values of $S+C$ and G. Both cases have a practical interest. Indeed, the case $S + C \leq G$ is consistant with what is observed on CPU nodes, since for a block size s, the number of operations induced by POTRF, TRSM, SYRK and GEMM are respectively $1/3s^3 + O(s^2)$, s^3, $s^3 + O(s^2)$ and $2s^3 + O(s^2)$. On a typical CPU, the ratios between the different task duration are close to the ratios between the number of induced flops. Table 2.2 describes the duration of individual tasks when $s = 960$ on an Intel Xeon E5-2680 and the relative duration of TRSM, SYRK and GEMM with respect to POTRF is typically given by $3, 3, 6$, what will be used as a toy example in this paper. On a typical GPU, it turns out that POTRF, although it induces less flops, achieves lower GPU performance, and the durations of the different kernels on a GPU are given in Table 2.2 for $s = 960$ on an Nvidia GK110BGL GPU unit. 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). Therefore, $S + C \leq G$ is typically what can be observed on CPU nodes, and $S + C > G$ on GPU nodes.

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

Based on 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.

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).

- Case of POTRF$(i), 1 \leq i \leq n$ node: the critical path starting from POTRF(i) follows a sequence of TRSM and GEMM tasks until reaching TRSM$(n-1, n) \rightarrow$ SYRK$(n, n-1) \rightarrow$ POTRF(n). 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

$$\text{CP} = 2C + T + S + (n-2)(T+G)$$

  and

$$L(C, i) = \text{CP} - (i-1)(T+G).$$

- Case of $\mathrm{TRSM}(i,j), 1 \leq i < j \leq n$ node: the critical path starting from $\mathrm{TRSM}(i,j)$ follows a sequence of GEMM and TRSM tasks until reaching $\mathrm{SYRK}(n,n-1) \rightarrow \mathrm{POTRF}(n)$. The critical path from $\mathrm{TRSM}(i,j)$ is given by $\mathrm{TRSM}(i,j) \rightarrow (\mathrm{GEMM}(i+1,j,i) \rightarrow \mathrm{TRSM}(i+1,j)) \rightarrow \ldots \rightarrow (\mathrm{GEMM}(j-1,j,j-2) \rightarrow \mathrm{TRSM}(j-1,j)) \rightarrow (\mathrm{GEMM}(j,n,j-1) \rightarrow \mathrm{TRSM}(j,n)) \rightarrow \ldots \rightarrow (\mathrm{GEMM}(n-1,n,n-2) \rightarrow \mathrm{TRSM}(n-1,n)) \rightarrow \mathrm{SYRK}(n,n-1) \rightarrow \mathrm{POTRF}(n)$. Its length is given by $T+(n-i-1)(T+G)+S+C$. Note that above quantity does not depend on j and can be expressed as

$$L(T,i,j) = \mathrm{CP} - C - (i-1)(T+G).$$

- Case of $\mathrm{SYRK}(i,j), 1 \leq j < i < n$ node: the critical path starting from $\mathrm{SYRK}(i,j)$ follows a sequence of SYRK tasks until reaching $\mathrm{SYRK}(i,i-1) \rightarrow \mathrm{POTRF}(i)$ and then the critical path from $\mathrm{POTRF}(i)$. Its length is given by $(i-j)S+C+(n-i-1)(T+G)+T+S+C$ and can be expressed as

$$L(S,i,j) = \mathrm{CP} - (i-1)(T+G) + (i-j)S.$$

- Case of $\mathrm{SYRK}(n,j), 1 \leq j < n$ node: the critical path starting from $\mathrm{SYRK}(n,j)$ follows of sequence of SYRK tasks until reaching $\mathrm{SYRK}(n,n-1) \rightarrow \mathrm{POTRF}(n)$. Its length is given by

$$L(S,n,j) = (n-j)S + C.$$

- Case of $\mathrm{GEMM}(i,j,k), 1 \leq k < i < j \leq n$ node: the critical path starting from $\mathrm{GEMM}(i,j,k)$ follows a sequence of GEMM tasks until reaching $\mathrm{GEMM}(i,j,i-1) \rightarrow \mathrm{TRSM}(i,j)$ and then the critical path from $\mathrm{TRSM}(i,j)$. Its length is given by $(i-k)G+T+(n-i-1)(T+G)+S+C$. Note that above quantity does not depend on j. and can be expressed as

$$L(G,i,j,k) = \mathrm{CP} - C + G + T - iT - kG.$$

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 Table 2.2. In this case, in particular $S + C + T \geq G + T$, so that $\mathrm{SYRK}(i+1,i) \rightarrow \mathrm{POTRF}(i+1)$ are now used in critical paths.

- Case of $\mathrm{POTRF}(i), 1 \leq i < n$ node: the critical path starting from $\mathrm{POTRF}(i)$ follows a sequence of TRSM, SYRK and POTRF tasks until reaching $\mathrm{POTRF}(n)$. Its length is given by $L(C,i) = C + (n-i)(T+S+C)$. Thus, the total critical path can be calculated by the formula:

$$\mathrm{CP} = C + (n-1)(T+S+C).$$

- Case of $\mathrm{TRSM}(i,j), 1 \leq i < j \leq n$ node: the critical path starting from $\mathrm{TRSM}(i,j)$ follows a sequence of GEMM and TRSM tasks until reaching $\mathrm{TRSM}(j-1,j) \rightarrow \mathrm{SYRK}(j,j-1) \rightarrow \mathrm{POTRF}(j)$ and then the critical path from $\mathrm{POTRF}(j)$. Its length is given by $L(T,i,j) = (j-i-1)(T+G) + (n-j+1)(T+S+C)$.

- Case of $\mathrm{SYRK}(i,j), 1 \leq j < i \leq n$ node: the critical path starting from $\mathrm{SYRK}(i,j)$ follows a sequence of SYRK tasks until reaching $\mathrm{SYRK}(i,i-1) \rightarrow \mathrm{POTRF}(i)$ and then the critical path from $\mathrm{POTRF}(i)$. Its length is given by $L(S,i,j) = (i-j)S + C + (n-i)(T+S+C)$.

Figure 3: $n = 8$. ALAP schedule without resource limitations on 8×8 tiles with 1,3,3,6 weights. Colors are the same as in Figure 1

- Case of GEMM$(i, j, k), 1 \leq k < i < j \leq n$ node: the critical path starting from GEMM(i, j, k) follows a sequence of GEMM tasks until reaching GEMM$(i, j, i-1) \rightarrow$ TRSM(i, j) and then the critical path from TRSM(i, j). Its length is given by $L(G, i, j, k) = (i - k)G + (j - i - 1)(T + G) + (n - j + 1)(T + S + C)$.

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 limitations will be considered in Section 6). In the ALAP schedule without resource limitation, we consider the Cholesky graph from the end, *i.e.* we reverse the task graph depicted in Figure 1 and we schedule tasks in this order as soon as they 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 limitations, then every task starts its execution at a distance exactly equal to its critical path (as defined in Sections 2.3 and) to the end of the schedule. 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. In Sections 3 (CPU case) and 4 (GPU case), we precisely evaluate the number of tasks of each type running at any instant of the ALAP schedule without resource limitations, and then we use these bounds to compute a lower bound on the execution time of any schedule in Section 5. Figure 3 depicts the execution of an ALAP schedule (without resource limitations)on a 8×8 tiled Cholesky factorization, with the time on the x-axis.

3 ALAP Schedule Analysis without resource limitations when $S + C \leq G$

In the ALAP Schedule without resource limitations, 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 (respectively denoted as #GEMM(d), #TRSM(d), #SYRK(d) and #POTRF(d)) that are being processed at this instant CP $- d$ using ALAP schedule. We also

determine an upper bound on the work performed by tasks of each type (respectively denoted as $W_{\mathrm{GEMM}}(d), W_{\mathrm{TRSM}}(d), W_{\mathrm{SYRK}}(d)$ and $W_{\mathrm{POTRF}}(d)$) 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.

3.1 Case of POTRF tasks

Clearly, at any instant, at most one POTRF task can be running since there is a dependency path $\mathrm{POTRF}(i) \longrightarrow \mathrm{TRSM}(i,i+1) \longrightarrow \mathrm{SYRK}(i+1,i) \longrightarrow \mathrm{POTRF}(i+1)$, therefore

$$\forall d \geq 0,\ \#\mathrm{POTRF}(d) \leq 1.$$

Then, POTRF tasks that terminate their execution after instant $\mathrm{CP}-d$ are characterized by the equation

$$(n-i-1)(T+G)+T+S+C \leq d \Leftrightarrow i \geq (n-1) - \frac{d-(T+S+C)}{T+G}$$

so that the total amount of work done after $\mathrm{CP}-d$ is defined by

$$\forall d \geq 0,\ W_{\mathrm{POTRF}}(d) \leq C^{\mathrm{POTRF},W} d + D^{\mathrm{POTRF},W},$$

where $C^{\mathrm{POTRF},W} = \frac{C}{T+G}$ and $D^{\mathrm{POTRF},W} = \frac{C(2G+T-S-C)}{T+G}$.

3.2 Case of TRSM tasks

$\mathrm{TRSM}(i,j)$ runs at all instants such that

$$\mathrm{CP} - C - (i-1)(T+G) - T \leq d \leq \mathrm{CP} - C - (i-1)(T+G),$$

so that in particular

$$\frac{\mathrm{CP}-d-C+G}{T+G} \leq i \leq \frac{\mathrm{CP}-d-C+T+G}{T+G}$$

and as the difference between the lower bound and the upper bound is less than 1, therefore if there exists an integer number in this interval then

$$i = \left\lfloor \frac{\mathrm{CP}-d-C+T+G}{T+G} \right\rfloor = n - \left\lceil \frac{d-S-C+G}{T+G} \right\rceil.$$

The amount of the TRSM tasks running at the time $\mathrm{CP}-d$ is either 0 or

$$\#\mathrm{TRSM}(d) = \left\lceil \frac{d-S-C+G}{T+G} \right\rceil \leq C^{\mathrm{TRSM}} d + D^{\mathrm{TRSM}},$$

where

$$C^{\mathrm{TRSM}} = \frac{1}{T+G} \text{ and } D^{\mathrm{TRSM}} = \frac{2G-S-C+T}{T+G}.$$

Then, TRSM tasks that terminate their execution after instant $\mathrm{CP}-d$ are characterized by the equation $(n-i-1)(T+G)+S+C \leq d \Leftrightarrow i \geq (n-1) - \frac{d-(S+C)}{T+G}$. Moreover, since $1 \leq i < j \leq n$, if $i = n-l$, then j can take l different values so that for any $d \geq 0$

$$W_{\mathrm{TRSM}}(d) \leq B^{\mathrm{TRSM},W} d^2 + C^{\mathrm{TRSM},W} d + D^{\mathrm{TRSM},W},$$

where $B^{\mathrm{TRSM},W} = \frac{T}{2(T+G)^2}$, $C^{\mathrm{TRSM},W} = \frac{T(3T+3G-2S-2C)}{2(T+G)^2}$ and $D^{\mathrm{TRSM},W} = \frac{T(T+G-S-C)(2T+2G-S-C)}{2(T+G)^2}$.

3.3 Case of SYRK tasks

Clearly, at any instant, at most one SYRK(n, j) task can be running since there is a dependency path SYRK$(n, j) \longrightarrow$ 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 SYRK(i, j) for $1 \leq j < i < n$. In this case,

$$L(S, i, j) = \text{CP} - (i - 1)(T + G) + (i - j)S = \text{CP} + (T + G) - i(T + G - S) - jS$$

so that 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 which follows that

$$j = \left\lfloor \frac{\text{CP} - d + (T + G) - i(T + G - S)}{S} \right\rfloor.$$

In order to determine how many pairs (i, j) correspond to a tasks SYRK(i, j) running at time CP $- d$, we need to check to consider the constraints on (i, j) valid pairs, *i.e.* $1 \leq j < i < n$.

- $j \geq 1$ can be rewritten as

$$\text{CP} - d + (T + G) - i(T + G - S) \geq S \Leftrightarrow i \leq n - \left\lceil \frac{d + G - 2C - nS}{T + G - S} \right\rceil$$

- $j < i$ can be rewritten as

$$\text{CP} - d + (T + G) - i(T + G - S) \leq iS \Leftrightarrow i \geq n - \left\lfloor \frac{d + G - 2C - S}{T + G} \right\rfloor.$$

  Moreover, we can observe that $\lfloor \frac{d+G-2C-S}{T+G} \rfloor \geq 0$ as soon as $d \geq 2C + S - G$ (otherwise the number of SYRK tasks is 0 as can be observed in Figure 1).

Let us now prove that (except for very large values of d, close to CP) that

$$n - \left\lceil \frac{d + G - 2C - nS}{T + G - S} \right\rceil \geq n - \left\lfloor \frac{d + G - 2C - S}{T + G} \right\rfloor.$$

Indeed, above equation is true as soon as

$$\frac{d + G - 2C - S}{T + G} \geq \frac{d + G - 2C - nS}{T + G - S} + 1 \Leftrightarrow d \geq \text{CP} + T + G - 2S - \frac{(T + G)(T + G - S)}{S}.$$

The last constraint $i < n$ or $i \leq n - 1$ can dominate over $i \leq n - \left\lceil \frac{d+G-2C-nS}{T+G-S} \right\rceil$ in cases when $\left\lceil \frac{d+G-2C-nS}{T+G-S} \right\rceil \leq 1$, *i.e.* when

$$d \leq d_S = (n - 1)S + 2C + T,$$

which leads to the following situations:

- If $d \leq (n-1)S + 2C + T = d_S$, then i must be between $n - \lfloor \frac{d+G-2C-S}{T+G} \rfloor$ and $n-1$ and

$$\#\mathrm{SYRK}(d) = \left\lfloor \frac{d+G-2C-S}{T+G} \right\rfloor$$

and for any $d \leq d_S$

$$\#\mathrm{SYRK}(d) \leq C_1^{\mathrm{SYRK}} d + D_1^{\mathrm{SYRK}},$$

where $C_1^{\mathrm{SYRK}} = \frac{1}{T+G}$ and $D_1^{\mathrm{SYRK}} = \frac{G-2C-S}{T+G}$.

To estimate the overall work induced by SYRK tasks, we need to add the component related to $\mathrm{SYRK}(n,j)$ tasks, that are processed in ALAP schedule for $C \leq d \leq C + (n-1)S$. Noticing that $C + (n-1)S - d_S = -C - T \leq 0$, the overall work produced by SYRK tasks in the last d time units is given by $\min(d - C, (n-1)S)$.

In order to estimate $W_{\mathrm{SYRK}}(d)$, we rely for other SYRK tasks on the integral of $\#\mathrm{SYRK}(t)$ between 0 and d so that

$$W_{\mathrm{SYRK}}(d) \leq B_1^{\mathrm{SYRK},W} d^2 + C_1^{\mathrm{SYRK},W} d + D_1^{\mathrm{SYRK},W},$$

where $B_1^{\mathrm{SYRK},W} = \frac{C_1^{\mathrm{SYRK}}}{2}$, $C_1^{\mathrm{SYRK},W} = D_1^{\mathrm{SYRK}} + 1$ and $D_1^{\mathrm{SYRK},W} = -C$.

- If $d > (n-1)S + 2C + T = d_S$, then i must be between $n - \lfloor \frac{d+G-2C-S}{T+G} \rfloor$ and $n - \lceil \frac{d+G-2C-nS}{T+G-S} \rceil$, so that

$$\#\mathrm{SYRK}(d) = \left\lfloor \frac{d+G-2C-S}{T+G} \right\rfloor - \left\lceil \frac{d+G-2C-nS}{T+G-S} \right\rceil + 1$$

and

$$\forall d > nS + 2C - G, \#\mathrm{SYRK}(d) \leq C_2^{\mathrm{SYRK}} d + D_2^{\mathrm{SYRK}},$$

where $C_2^{\mathrm{SYRK}} = \frac{-S}{(T+G)(T+G-S)}$ and $D_2^{\mathrm{SYRK}} = 1 + \frac{S((n-1)(T+G)-G+2C+S)}{(T+G)(T+G-S)}$.

As previously, to estimate the overall work induced by SYRK tasks, we need to add the component related to $\mathrm{SYRK}(n,j)$ tasks, that are processed in ALAP schedule for $C \leq d \leq C + (n-1)S$. Noticing that $C + (n-1)S \leq d_S$, we know that the total work produced by $\mathrm{SYRK}(n,j)$ tasks is exactly $(n-1)S$.

In order to estimate $W_{\mathrm{SYRK}}(d)$ for the other SYRK tasks, we rely on the integral of $\#\mathrm{SYRK}(t)$ between d_S and d plus $W_{\mathrm{SYRK}}(d_S)$ so that

$$W_{\mathrm{SYRK}}(d) \leq B_2^{\mathrm{SYRK},W} d^2 + C_2^{\mathrm{SYRK},W} d + D_2^{\mathrm{SYRK},W},$$

where $B_2^{\mathrm{SYRK},W} = \frac{C_2^{\mathrm{SYRK}}}{2}$, $C_2^{\mathrm{SYRK},W} = D_2^{\mathrm{SYRK}}$ and $D_2^{\mathrm{SYRK},W} = (B_1^{\mathrm{SYRK},W} - B_2^{\mathrm{SYRK},W}) d_S^2 + (C_1^{\mathrm{SYRK},W} - C_2^{\mathrm{SYRK},W}) d_S + (n-1)S$.

3.4 Case of GEMM tasks

Let us now concentrate on GEMM tasks. $\mathrm{GEMM}(i,j,k)$ runs at all instants such that

$$\mathrm{CP} - C + T - iT - kG \leq d \leq \mathrm{CP} - C + T - iT - kG + G,$$

so that in particular

$$\frac{\mathrm{CP}-d-C+T-iT}{G} \leq k \leq \frac{\mathrm{CP}-d-C+T-iT}{G}+1$$

so that at most one value of k is possible, for a fixed pair (i,d) and

$$k = \left\lceil \frac{\mathrm{CP}-d-C+T}{G} - \frac{iT}{G} \right\rceil.$$

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.* $1 \leq k < i < j \leq n$.

- The first constraint states that

$$k \geq 1 \Leftrightarrow \frac{\mathrm{CP}-d-C+T}{G} \geq \frac{iT}{G} \Leftrightarrow i \leq \frac{\mathrm{CP}-d-C+T}{T}.$$

  This constraint can be rewritten as

$$i \leq n + \frac{nG+C+S-2G-d}{T}.$$

  Note that in particular, when d is small enough, *i.e.* $d \leq nG+C+S-2G$, then above constraint becomes trivial and can be replaced by $i \leq n$. Otherwise, if $d \geq nG+C+S-2G$, then the constraint becomes

$$i \leq n - \left\lceil \frac{d-(nG+C+S-2G)}{T} \right\rceil.$$

- The second constraint states that

$$k < i \Leftrightarrow \frac{\mathrm{CP}-d-C+T}{G} - \frac{iT}{G} \leq (i-1) \Leftrightarrow \mathrm{CP}-d-C+T+G \leq i(G+T).$$

  This constraint can be rewritten as

$$\begin{aligned}(n-i-2)(T+G) \leq d-(C+G+S+2T) \Leftrightarrow i \geq (n-1) - \frac{d-(C+S+T)}{T+G} \\ \Leftrightarrow i \geq n - \left\lceil \frac{d-(C+S+T)}{T+G} \right\rceil.\end{aligned}$$

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

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

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

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

where $B_1^{\mathrm{GEMM}} = \frac{1}{2(G+T)^2}$, $C_1^{\mathrm{GEMM}} = \frac{(3G+T-2C-2S)}{2(G+T)^2}$ and $D_1^{\mathrm{GEMM}} = \frac{(G-C-S)(2G+T-C-S)}{2(G+T)^2}$.
In order to estimate $W_{\mathrm{GEMM}}(d)$, we rely on the integral of $\#\mathrm{GEMM}(t)$ between 0 and d so that

$$W_{\mathrm{GEMM}}(d) \leq A_1^{\mathrm{GEMM},W} d^3 + B_1^{\mathrm{GEMM},W} d^2 + C_1^{\mathrm{GEMM},W} d,$$

where $A_1^{\mathrm{GEMM},W} = \frac{B_1^{\mathrm{GEMM}}}{3}$, $B_1^{\mathrm{GEMM},W} = \frac{C_1^{\mathrm{GEMM}}}{2}$ and $C_1^{\mathrm{GEMM},W} = D_1^{\mathrm{GEMM}}$.

– If $d \geq \mathrm{CP} - C - T$, then there is no GEMM task to perform (only TRSMs and one POTRF remain). This corresponds to the case when

$$\left\lceil \frac{d-(C+S+T)}{T+G} \right\rceil < \left\lceil \frac{d-((n-2)G+C+S)}{T} \right\rceil,$$

i.e. the instant when the lower bound on i becomes smaller than the upper bound. In this case,

$$\#\mathrm{GEMM}(d) = 0.$$

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

$$\#\mathrm{GEMM}(d) = \sum_{l=\left\lceil \frac{d-(nG+C+S-2G)}{T} \right\rceil}^{\left\lceil \frac{d-(C+S+T)}{T+G} \right\rceil} l$$

and after simplifying it we obtain

$$\#\mathrm{GEMM}(d) \leq \frac{1}{2}\left(\frac{d+G-C-S}{T+G}\right)^2 - \frac{(d-((n-2)G+C+S))^2}{2T^2} + \frac{d-(C+S+T)}{2(T+G)} + \frac{d-((n-2)G+}{2T}$$

so that

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

where $B_2^{\mathrm{GEMM}} = \frac{1}{2(G+T)^2} - \frac{1}{2T^2}$, $C_2^{\mathrm{GEMM}} = \frac{1}{2(T+G)} + \frac{1}{2T} - \frac{C+S-G}{(T+G)^2} + \frac{(n-2)G+C+S}{T^2}$and $D_2^{\mathrm{GEMM}} = 1 - \frac{C+S+T}{2(T+G)} - \frac{(n-2)G+C+S}{2T} + \frac{(C+S-G)^2}{2(T+G)^2} - \frac{((n-2)G+C+S)^2}{2T^2}$
In order to estimate $W_{\mathrm{GEMM}}(d)$, we rely on the integral of $\#\mathrm{GEMM}(t)$ between d_G and d plus $W_{\mathrm{GEMM}}(d_G)$ so that

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

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

4 ALAP Schedule Analysis without resource limitations when $S+C \geq G$

4.1 Case of POTRF tasks

Similarly to CPU case, at any instant at most one POTRF task can be running since there is a dependency path $\mathrm{POTRF}(i) \longrightarrow \mathrm{TRSM}(i,i+1) \longrightarrow \mathrm{SYRK}(i+1,i) \longrightarrow \mathrm{POTRF}(i+1)$, therefore

$$\forall d \geq 0,\ \#\mathrm{POTRF}(d) \leq 1.$$

Then, POTRF tasks that terminate their execution after instant $\mathrm{CP}-d$ are characterized by the equation

$$(n-i)(T+S+C) \leq d \Leftrightarrow i \geq n - \left\lfloor \frac{d}{T+S+C} \right\rfloor$$

so that the total amount of work done after $\mathrm{CP}-d$ is defined by

$$\forall d \geq 0,\ W_{\mathrm{POTRF}}(d) \leq C^{\mathrm{POTRF},W} d + D^{\mathrm{POTRF},W},$$

where

$$C^{\mathrm{POTRF},W} = \frac{C}{T+S+C} \text{ and } D^{\mathrm{POTRF},W} = C.$$

4.2 Case of TRSM tasks

TRSM(i,j) runs at all instants such that

$$\begin{aligned}(j-i-1)(T+G)+S+C+(n-j)(C+S+T) &\leq d \\ &\leq (j-i-1)(T+G)+(n-j+1)(C+S+T)\end{aligned}$$

so that in particular

$$j-1+\frac{(n-j+1)(C+S+T)-d}{T+G} \leq i \leq j-1+\frac{(n-j+1)(C+S+T)-d-T}{T+G}$$

and as the difference between the lower bound and the upper bound is less than 1, therefore if there exists an integer number in this interval then

$$i = \left\lfloor j-1+\frac{(n-j+1)(C+S+T)-d}{T+G} \right\rfloor.$$

Now, if there exist some TRSM tasks at time d from the critical path, then it should comply with $1 \leq i < j \leq n$. The first condition $i \leq 1$ implies

$$j-1+\frac{(n-j+1)(C+S+T)-d}{T+G} \geq 1 \Leftrightarrow (C+S-G)j \leq n(C+S+T)+C+S-T-2G-d$$

$$j \leq n+1-\frac{d-(n-1)(T+G)}{C+S-G} \Leftrightarrow j \leq n - \left\lfloor \frac{d-(n-1)(T+G)}{C+S-G} \right\rfloor.$$

The second condition $i < j$ leads to

$$j - 2 + \frac{(n-j+1)(C+S+T)-d}{T+G} \leq j-1 \Leftrightarrow (C+S+T)j \geq n(C+S+T)+C+S-G-d$$

$$j \geq n - \left\lfloor \frac{d+G-C-S}{C+S+T} \right\rfloor.$$

Depending on value of d either $j \leq n$ or $j \leq n - \left\lfloor \frac{d-(n-1)(T+G)}{C+S-G} \right\rfloor$ should prevail. The condition $j \leq n$ will dominate if

$$n - \left\lfloor \frac{d-(n-1)(T+G)}{C+S-G} \right\rfloor \geq n \Leftrightarrow d = d_T \leq (n-1)(T+G)+C+S-G.$$

Thus, we consider two different zones for TRSMs:

- When $d \leq d_T$, then the number of TRSMs could be found by

$$\#\text{TRSM}(d) = \left\lfloor \frac{d+G-C-S}{C+S+T} \right\rfloor + 1 \leq \frac{d+G+T}{C+S+T} = C_1^{\text{TRSM}} d + D_1^{\text{TRSM}}$$

  where $C_1^{\text{TRSM}} = \frac{1}{C+S+T}$ and $D_1^{\text{TRSM}} = \frac{G+T}{C+S+T}$.

- When $d \geq d_T$, then

$$\begin{aligned}\#\text{TRSM}(d) &= \left\lfloor \frac{d+G-C-S}{C+S+T} \right\rfloor - \left\lfloor \frac{d-(n-1)(T+G)}{C+S-G} \right\rfloor + 1 \\ &\leq \frac{d+G-C-S}{C+S+T} - \frac{d-(n-1)(T+G)}{C+S-G} + 2 \leq C_2^{\text{TRSM}} d + D_2^{\text{TRSM}}\end{aligned}$$

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

The total amount of TRSM work that is finished after $\text{CP}-d$ could be obtained by integrating the above inequalities. Thus, for $d \leq d_T$ we have

$$W_{\text{TRSM}}(d) \leq \frac{C_1^{\text{TRSM}}}{2} d^2 + D_1^{\text{TRSM}} d$$

and for $d \geq d_T$

$$W_{\text{TRSM}}(d) \leq B^{\text{TRSM},W} d^2 + C^{\text{TRSM},W} d + D^{\text{TRSM},W}$$

where $B^{\text{TRSM},W} = \frac{C_2^{\text{TRSM}}}{2}$, $C^{\text{TRSM},W} = D_2^{\text{TRSM}}$ and $D^{\text{TRSM},W} = \frac{C_1^{\text{TRSM}} - C_2^{\text{TRSM}}}{2} d_T^2 + (D_1^{\text{TRSM}} - D_2^{\text{TRSM}}) d_T$.

4.3 Case of SYRK tasks

$\forall 1 \leq j < i \leq n$, then $\text{SYRK}(i,j)$ runs at all instants d such that

$$-jS + C + n(C+S+T) - i(C+T) - S \leq d \leq -jS + C + n(C+S+T) - i(C+T)$$

so that in particular there is only one possible value of j for each valid i value.

$$j = \lfloor \frac{C + n(C+S+T) - i(C+T) - d}{S} \rfloor$$

The additional constraints to define the validity domain for i are $j \geq 1$ and $j < i$

- $j \geq 1 \iff \frac{C+n(C+S+T)-i(C+T)-d}{S} \geq 1$ and finally $i \leq \left\lfloor \frac{\mathrm{CP}-d-S}{T+C} \right\rfloor = n - \left\lceil \frac{d-C-(n-1)S}{T+C} \right\rceil$.
- $j+1 \leq i \iff \frac{C+n(C+S+T)-i(C+T)-d}{S} \leq i$ and finally $i \geq \left\lceil \frac{\mathrm{CP}-d}{T+C+S} \right\rceil = n - \left\lfloor \frac{d-C}{T+S+C} \right\rfloor$.

Therefore, the maximum number of possible values of i (and therefore the maximum number of SYSRKs) is bounded by

$$\forall d \geq 0,\ \#\mathrm{SYRK}(d) \leq C^{\mathrm{SYRK}} d + C^{\mathrm{SYRK}},$$

where $C^{\mathrm{SYRK}} = \frac{-S}{(C+S+T)(C+T)}$ and $D^{\mathrm{SYRK}} = 1 + \frac{(n-1)S}{(C+T)} + \frac{CS}{(C+S+T)(C+T)}$.

The total amount of work that is done after $\mathrm{CP} - d$ can be defined by integrating the upper bound given above. In the end,

$$W_{\mathrm{SYRK}} \leq \frac{C^{\mathrm{SYRK}}}{2} d^2 + C^{\mathrm{SYRK}} d.$$

4.4 Case of GEMM tasks

$\forall 1 \leq k < i < j \leq n$, then $\mathrm{GEMM}(i,j,k)$ runs at all instants d such that

$$(n+1)(S+C+T)-(T+G)-iT-j(S+C-G)-G \leq d \leq (n+1)(S+C+T)-(T+G)-iT-j(S+C-G)$$

so that in particular there is only one possible value of k for each valid (i,j) pair, that is given by

$$k = \lfloor \frac{(n+1)(S+C+T)-(T+G)-iT-j(S+C-G)-d}{G} \rfloor.$$

In order to determine the set of valid (i,j) pairs, we have to take into account the constraints $1 \leq k < i < j \leq n$.

- $k \leq 1$ that can be rewritten $(n+1)(S+C+T)-(T+G)-iT-j(S+C-G)-d \geq G$. Let us now consider the constraint $j \leq n$. We can observe that the value of j decreases with i so that it is of particular interest to consider the value $i = 1$. When $i = 1$, $j \leq n$ can be rewritten as $d \geq n(T+G)+S+C-2G-T$. Since d can be as large as $n(T+S+C)+C$ and $G \leq S+C$, $j \leq n$ will be automatically satisfied when d is large enough, but must be enforced otherwise.
- $k < i$ that can be rewritten $(n+1)(S+C+T)-(T+G)-iT-j(S+C-G)-d \geq iG$. Let us now consider the constraint $j \leq n$. We can observe that the value of j decreases with i so that it is of particular interest to consider the value $i = 1$. When $i = 1$, $j \leq n$ can be as previously be rewritten as $d \geq n(T+G)+S+C-2G-T$.

Therefore, the different possible situations are depicted in Figure 4, Figure 5 and Figure 6 where constraint $i < j$ has been added, and the goal is to evaluate the number of integer (i,j) pairs in the shaded green area. In the case when $d \leq n(T+G)+S+C-2G-T$, it is of particular interest to focus on the relative positions of the intersections of constraints corresponding to $k \leq 1$ and $k < i$ with the constraint $i < j$ in order to precisely define the shape of the validity constraints.

- Let us denote by $i_{k<i}$ the intersection of constraint $k < i$ with $j = n$. The, $i_{k<i} \leq n$ can be rewritten as $d \geq S+C-G$, so that we can assume that it always holds true.

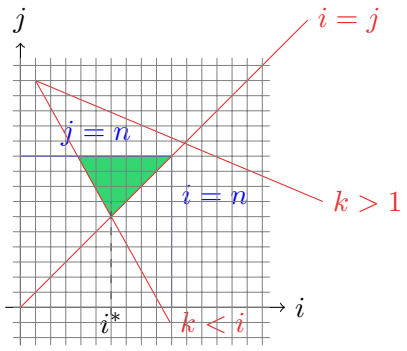

Figure 4: Number of GEMMs in the GPU when $d \geq nG + S + C - 2G$.

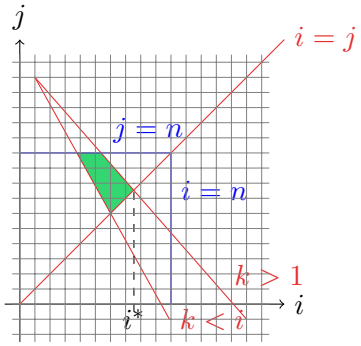

Figure 5: Number of GEMMs in the GPU when $nG + S + C - 2G \leq d \leq n(G + T) + S + C - 2G$.

- Let us denote by $i_{k \geq 1}$ the intersection of constraint $k \geq 1$ with $j = n$. The, $i_{k \geq 1} \leq n$ can be rewritten as $d \geq S + C - G$, so that it holds true as soon as $d \geq nG + S + C - 2G$.

As a partial conclusion, we are left with 3 different situations, depending on the value of d.

- When $d \geq nG + S + C - 2G$, the situation is depicted in Figure 4 and we are left with the problem of estimating the number of integer (i, j) points in the triangle defined by $k > 1$, $j \leq n$ and $j > i$.
- When $nG + S + C - 2G \leq d \leq n(G + T) + S + C - 2G$, the situation is depicted in Figure 5 and we are left with the problem of estimating the number of integer (i, j) points in the quadrilateral defined by $k > 1$, $i > k$, $j \leq n$ and $j > i$. To estimate this number of pairs, we will consider the set of points that lie in the triangle defined by $k > 1$, $j \leq n$ and $j > i$ minus the set of points that lie in the triangle defined by $k < i$, $j \leq n$ and $j > i$.
- When $d \geq n(G + T) + S + C - 2G$, the situation is depicted in Figure 6 and we are left with the problem of estimating the number of integer (i, j) points in the triangle defined by $k > 1$, $k < i$ and $j > i$.

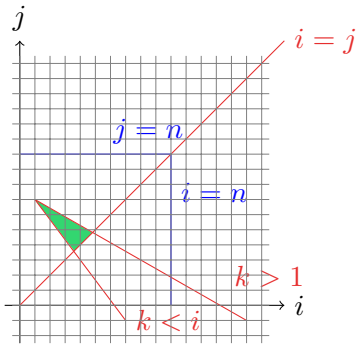

Figure 6: Number of GEMMs in the GPU when $d \geq n(G+T)+S+C-2G$.

4.4.1 Case of Figure 4

The point at the intersection of constraints $k > 1$ and $i = j$ is defined by $j* = n - \frac{d}{T+S+C} + \frac{S+C-G}{T+S+C}$ and then, for each j value between $\lceil j* \rceil$ and n, all integer i values for i between $\frac{(n+1)(S+C+T)-(T+G)}{(T+G)} - j\frac{(S+C-G)}{(T+G)}$ and $j-1$, that can be upper bounded by

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

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 $D_1^{\text{GEMM}} = 1$.

4.4.2 Case of Figure 5

As already mentioned, we will first compute the number of points that lie in the triangle defined by constraints $k < i$, $j \leq n$ and $j > i$ and then remove them from the set of points in the triangle defined $k > 1$, $j \leq n$ and $j > i$, that corresponds to the case of Figure 4 that we just analyzed.

The point at the intersection of constraints $k < i$ and $i = j$ is defined by $j* = n(1 + \frac{G}{T+S+C-G}) + \frac{S+C-2G-d}{T+S+C-G}$ and then, for each j value between $\lceil j* \rceil$ and n, all integer i values for i between $\frac{(n+1)(S+C+T)-(T+2G)-d}{T} - j\frac{(S+C-G)}{T}$ and $j-1$, that can be lower bounded by $\leq B_2'^{\text{GEMM}} d^2 + C_2'^{\text{GEMM}} + D_2'^{\text{GEMM}}$, where $B_2'^{\text{GEMM}} = \frac{1}{2(T+S+T-G)(T)}$, $C_2'^{\text{GEMM}} = \frac{nG}{2(T+S+C-G)(T)}$ and $D_2'^{\text{GEMM}} = \frac{n^2G^2}{2(T+S+C-G)(T)}$ so that the overall number of (i,j) pairs can be bounded by

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

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 $D_2^{\text{GEMM}} = 1 - \frac{n^2G^2}{2(T+S+C-G)(T)}$.

We can observe that the maximal number of $\#\text{GEMM}(d)$ tasks is obtained in the interval of d values corresponding to Figure 5. Therefore, since we are interested in finding the smallest value d such that ALAP occupies more than a given number of processors, we can safely ignore the case corresponding to Figure 6.

Therefore, the overall work that can be performed by ALAP without resource limitations after instant $\text{CP} - d$ can be defined by integrating the upper bounds given above. In the end,

- If $d \leq nG + S + C - 2G$, then

$$W_{\text{SYRK}} \leq \frac{B_1^{\text{GEMM}}}{3} d^3 + \frac{C_1^{\text{GEMM}}}{2} d^2 + D_1^{\text{GEMM}} d$$

.

- If $nG + S + C - 2G \leq d \leq n(G+T) + S + C - 2G$, then

$$W_{\text{SYRK}} \leq \frac{B_2^{\text{GEMM}}}{3}(d^3 - d_1^3) + \frac{C_2^{\text{GEMM}}}{2}(d^2 - d_1^2) + D_2^{\text{GEMM}}(d - d_1) + \frac{B_1^{\text{GEMM}}}{3} d_1^3 + \frac{C_1^{\text{GEMM}}}{2} d_1^2 + D_1^{\text{GEMM}} d_1,$$

where $d_1 = nG + S + C - 2G$.

5 Lower Bound for Cholesky Factorization with P resources.

Using 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.

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

For SYRK and GEMM tasks, we have two different formulas depending on the value of d. Indeed, the number of GEMM tasks is given by

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

if $d \leq d_G = (n-2)G + C + S + T$ and

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

otherwise.

Similarly, If $d \leq (n-1)S + 2C + T = d_S$, then

$$\#\text{SYRK}(d) \leq (C_1^{\text{SYRK}} d + D_1^{\text{SYRK}})$$

and

$$\#\text{SYRK}(d) \leq (C_2^{\text{SYRK}} d + D_2^{\text{SYRK}})$$

otherwise.

We can remark that $d_G \geq d_S \iff (n-2)G + C + S + T \geq (n-1)S + 2C + T \iff (n-2)(G - S) - C \geq 0$ so that $d_G \geq d_S$ as soon as $n \geq 3$, what we will assume. 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 $t > \text{CP} - d_S$.

Figure 12 displays the upper bound on the overall number of tasks processed at any instant t, $0 \leq t \leq \text{CP}$, whereas Figure 13 14 15 and Figure 16 display the same information for each type of task, GEMM, TRSM, SYRK and POTRF respectively. All plots correspond to the case where $G = 6$, $T = S = 3$ and $C = 1$, that typically corresponds to the situation on a CPU node.

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 $t > \text{CP} - d_S$.

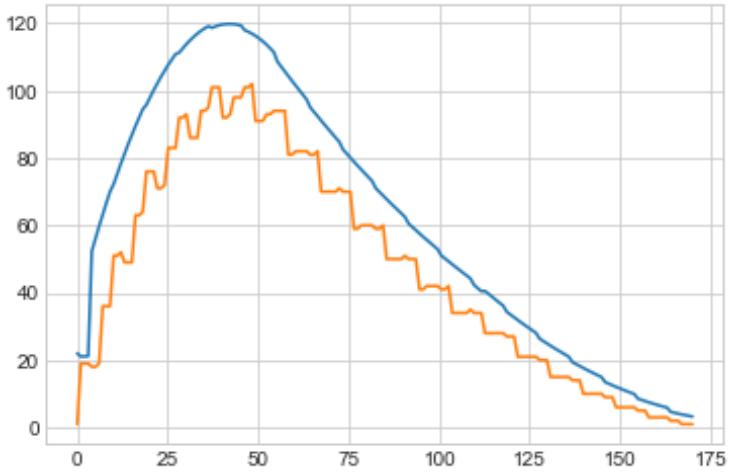

Figure 7: Overall number of tasks $f_{\#}(t),\ (n = 40)$, CPU case

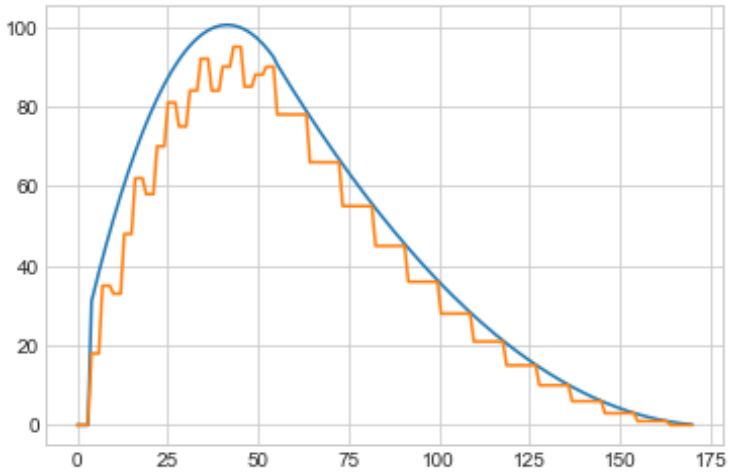

Figure 8: Overall number of GEMM tasks $f_{\#}(t),\ (n = 40)$, CPU case

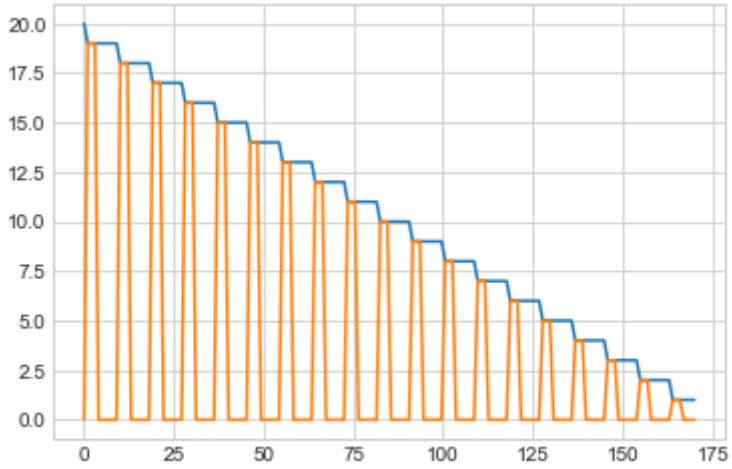

Figure 9: Overall number of TRSM tasks $f_{\#}(t),\ (n = 40)$, CPU case

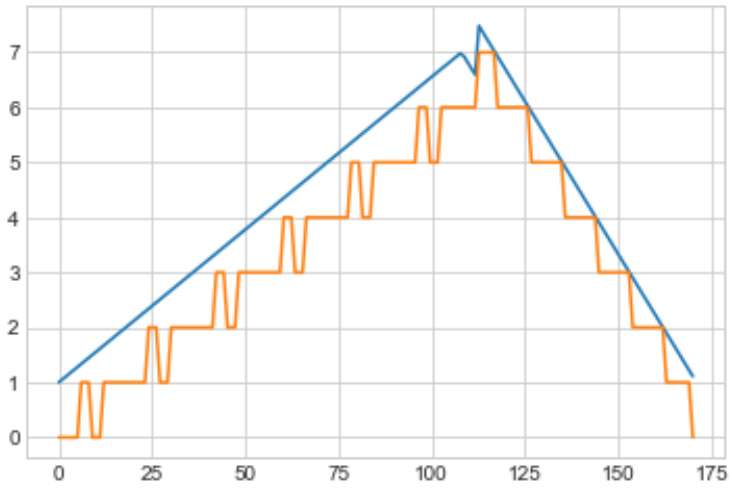

Figure 10: Overall number of SYRK tasks $f_{\#}(t),\ (n = 40)$, CPU case

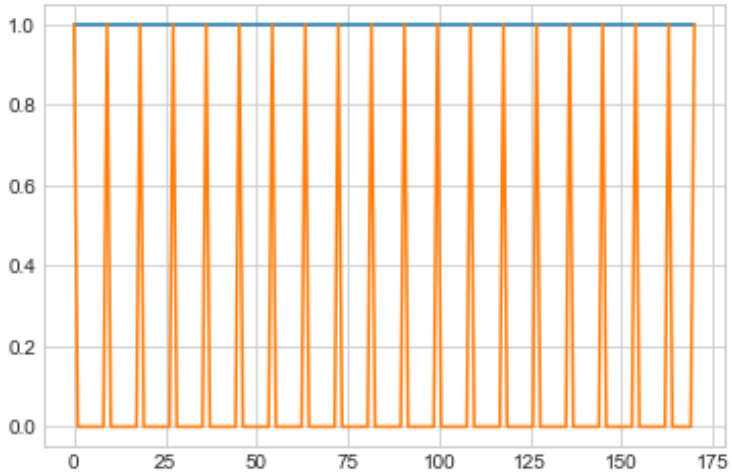

Figure 11: Overall number of POTRF tasks $f_{\#}(t)$, $(n = 40)$, CPU case

5.2 GPU Case, $S + C \geq G$

In the GPU case, for TRSM and GEMM tasks, we have two different formulas depending on the value of d.

As previously, 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 $t > \text{CP} - d_S$.

Figure 12 displays the upper bound on the overall number of tasks processed at any instant t, $0 \leq t \leq \text{CP}$, whereas Figure 13 14 15 and Figure 16 display the same information for each type of task, GEMM, TRSM, SYRK and POTRF respectively. All plots correspond to the case where $G = 2$, $T = 3$, $S = 1$ and $C = 12$, that typically corresponds to the situation on a GPU node.

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 $t > \text{CP} - d_S$.

5.3 Overall Number of Tasks

Let us define t_P as the largest instant such that $f_{\#}(t) \leq P$ for any $t \geq t_P$. 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 $S + C \geq G$ (CPU case) or $S + C \leq G$ (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

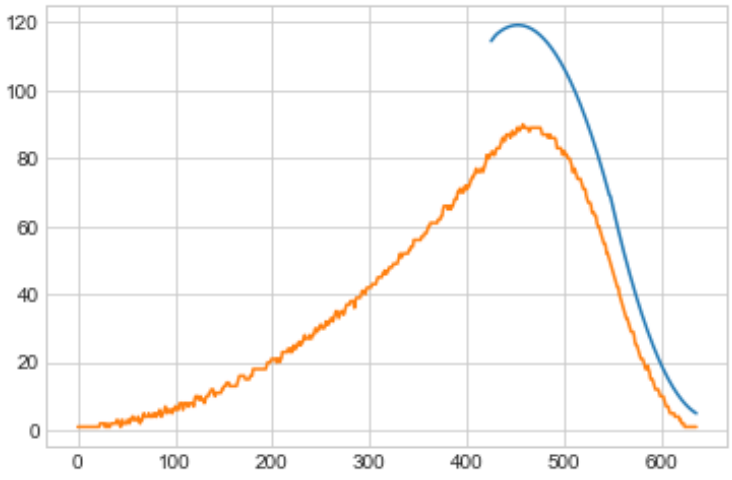

Figure 12: Overall number of tasks $f_{\#}(t),\ (n = 40)$, GPU case

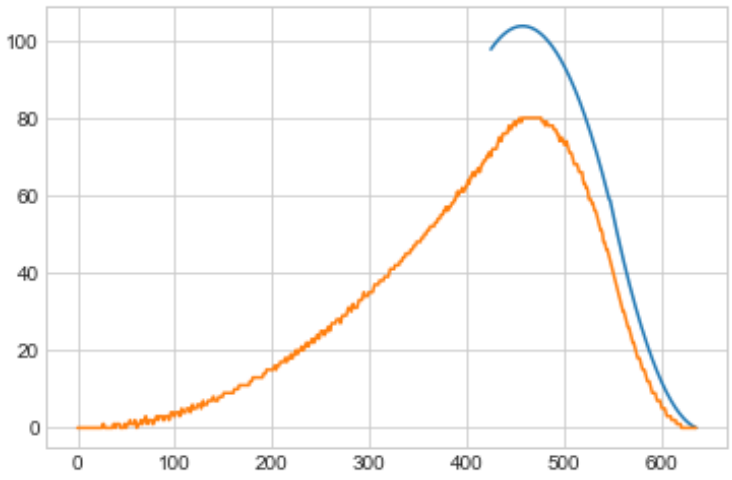

Figure 13: Overall number of GEMM tasks $f_{\#}(t),\ (n = 40)$, GPU case

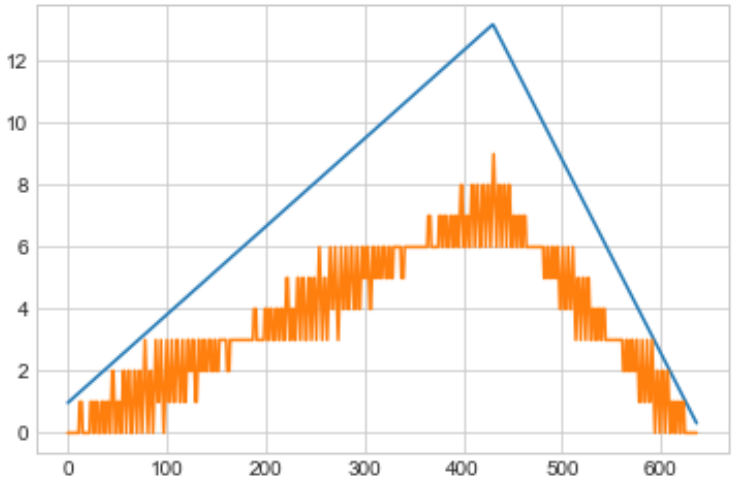

Figure 14: Overall number of TRSM tasks $f_{\#}(t),\ (n = 40)$, GPU case

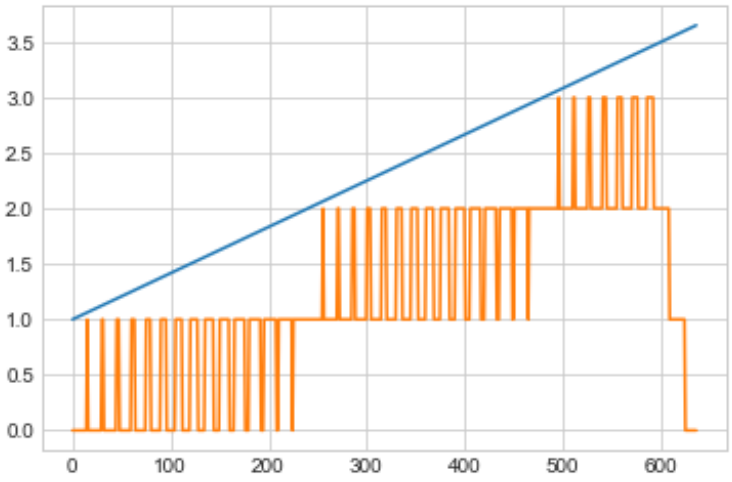

Figure 15: Overall number of SYRK tasks $f_{\#}(t),\ (n = 40)$, GPU case

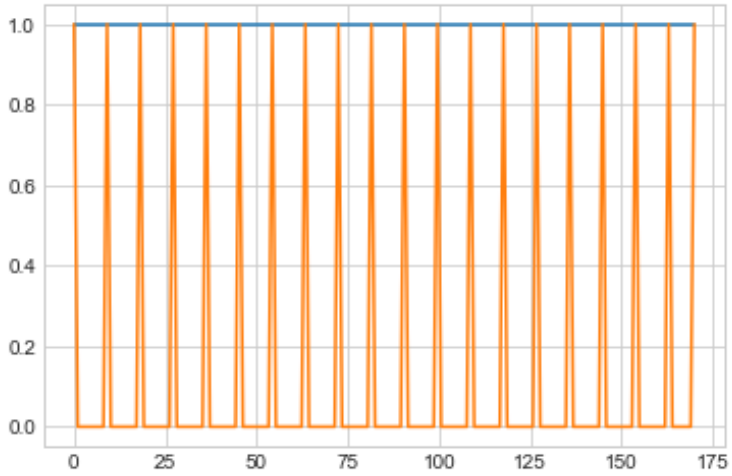

Figure 16: Overall number of POTRF tasks $f_{\#}(t)$, $(n = 40)$, GPU case

an upper bound on the number of tasks (resp. the overall work) processed simultaneously at time t by ALAP schedule without resource limitation. Moreover $\mathrm{CP} - t_P$ is the largest instant where the ALAP schedules 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 $\mathrm{CP} - t_P$) by $f_W(t_P)$, what achieves the proof of the lemma. □

Theorem 1. *The makespan of any schedule is lower bounded by* $(\mathrm{CP} - t_P) + \frac{W - f_W(t_P)}{P}$.

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 the amount of work W_{END} that can be processed during the last $\mathrm{CP} - t_P$ time units is upper bounded by $f_W(t_P)$. Similarly, the amount of work W_{BEGIN} that can be processed during the first $\textsc{Makespan}(\mathcal{S}) - (\mathrm{CP} - T_p)$ time units is trivially upper bounded by $P(\textsc{Makespan}(\mathcal{S}) - (\mathrm{CP} - T_p))$.

Therefore

$$W = nC + \frac{n(n-1)}{2}(S+T) + \frac{n(n-1)(n-2)}{6}G = W_{\text{BEGIN}} + W_{\text{END}} \leq f_W(t_P) + P(\textsc{Makespan}(\mathcal{S}) - (\mathrm{CP} - T_p))$$

so that

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

□

6 Simulation Results

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

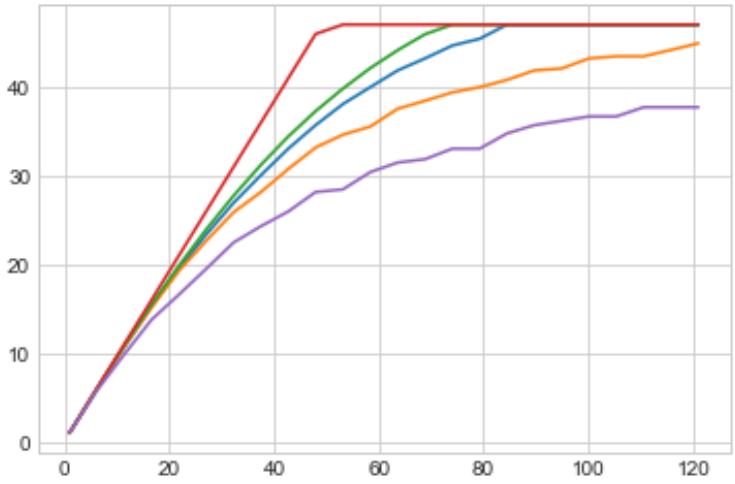

Figure 17: Evolution of speedup with the number of processors P, CPU case, $n = 20$

6.1 Comparison of different heuristics and bounds

In a first set of simulation, we plot the achieved speedup achieved by the different heuristics against theoretical bounds. The first theoretical (trivial) bound on the achievable speedup on P processors $\min(P, W/CP)$. The second bound is the one established in Section 5, based on a detailed analysis of ALAP schedule for Cholesky factorization. We consider the following heuristics:

- ALAP is the heuristic that we described in Section 2.5 when there is no resource limitations. In presence of resource limitations, when several tasks are ready, we define the highest priority task as the one that maximizes the length of the longest path between POTRF(1) and this task.
- ASAP (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 limitations. In presence of resource limitations, when several tasks are ready, we define the highest priority task as the one that maximizes the length of the longest path between this task and POTRF(n).
- Lapack corresponds to the Cholesky factorization implemented in the Lapack library. It consists in n 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.

We performed simulations with different problem sizes ($N = 20$, 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 all Figures 17, 18, 19, 20, 21 and 22, the red plot corresponds to the trivial lower bound, the green plot to our new lower bound, the blue plot to ALAP, the yellow plot to ASAP and the purple plot to Lapack.

The first observation is that the length of the ALAP schedule and the lower bound are always extremely close, what confirms the tightness of our analysis and the excellent performance of ALAP

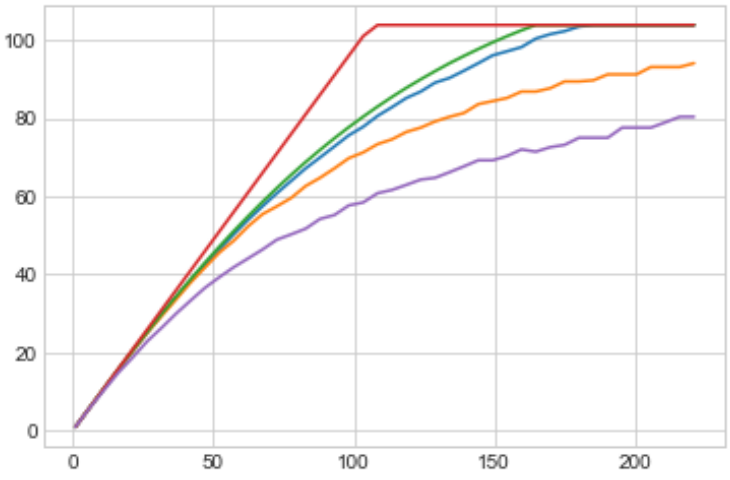

Figure 18: Evolution of speedup with the number of processors P, CPU case, $n = 30$

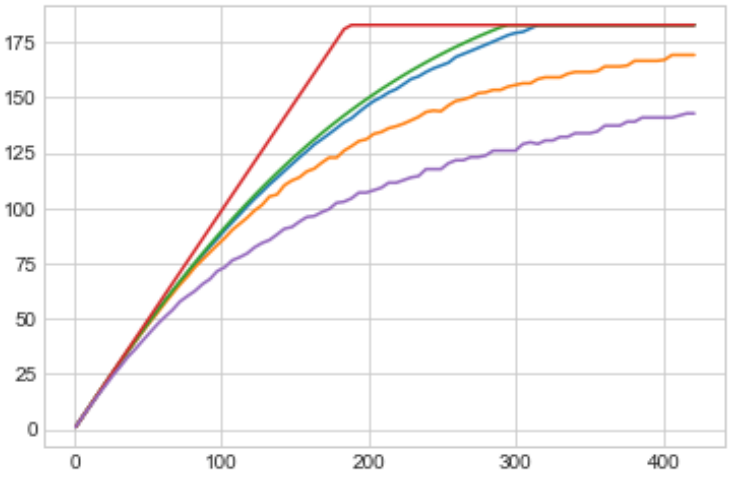

Figure 19: Evolution of speedup with the number of processors P, CPU case, $n = 40$

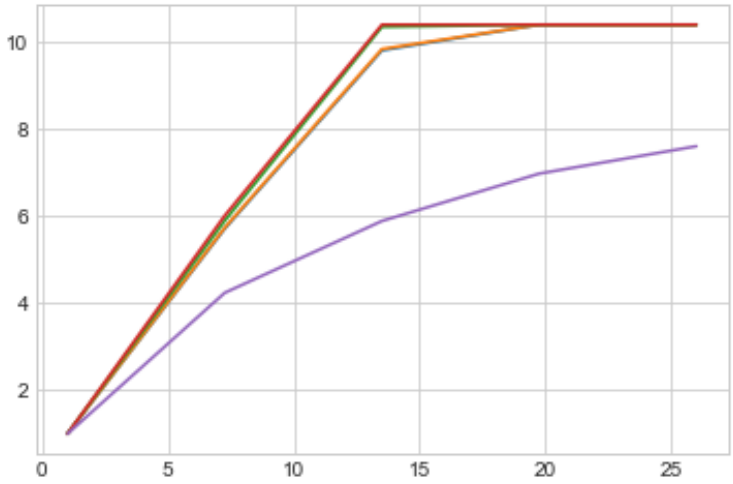

Figure 20: Evolution of speedup with the number of processors P, GPU case, $n = 20$

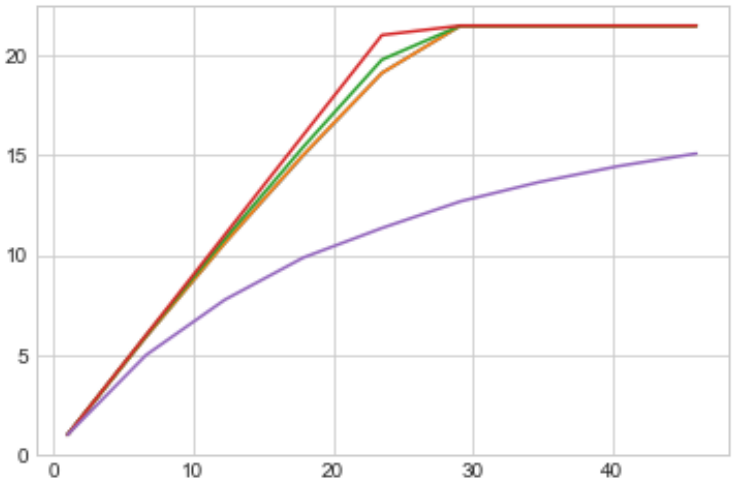

Figure 21: Evolution of speedup with the number of processors P, GPU case, $n = 30$

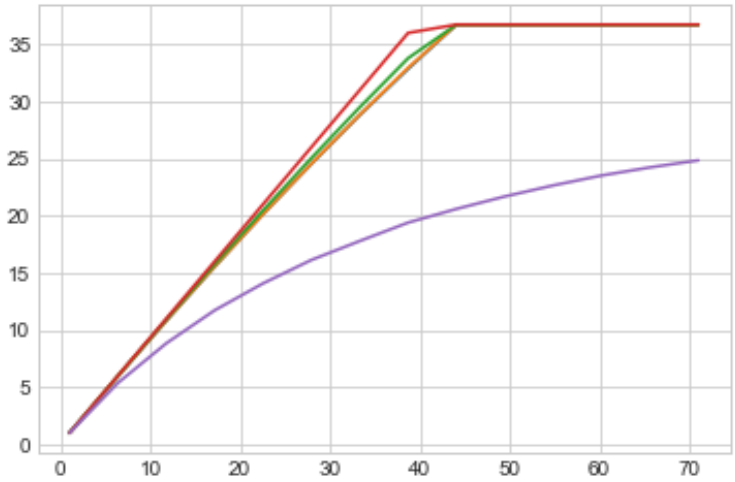

Figure 22: Evolution of speedup with the number of processors P, GPU case, $n = 40$

schedule. Then, we can observe that the GPU case is much easier than the CPU case. Indeed, in the GPU case, both ALAP and ASAP perform remarkably well (their performance is undistinguishable on the plots), whereas in the CPU case, only ALAP performs well. On the other hand, in the GPU case, we can observe that LapackV1 performs poorly. Indeed, in the GPU case, POTRF tasks perform significantly slower that GEMM tasks such that it is crucial to perform other tasks in parallel with POTRF tasks.

6.2 Asymptotic Performance

In order to establish the asymptotic performance of ALAP and ASAP, we also performed another experiment, where the size of the problem varies. More specifically, for each problem size, we plot the maximum (over all possible values of P) of the relative difference

$$\frac{\textsc{Makespan}(\text{ALAP}) - \textsc{LowerBound}}{\textsc{Makespan}(\text{ALAP})}.$$

The CPU (resp. GPU) case is depicted in Figure 23 (resp. Figure 24). In both cases, we can observe that when n grows, the relative difference gets smaller, suggesting that ALAP is asymptotically optimal when the problem size becomes large.

7 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 very 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.

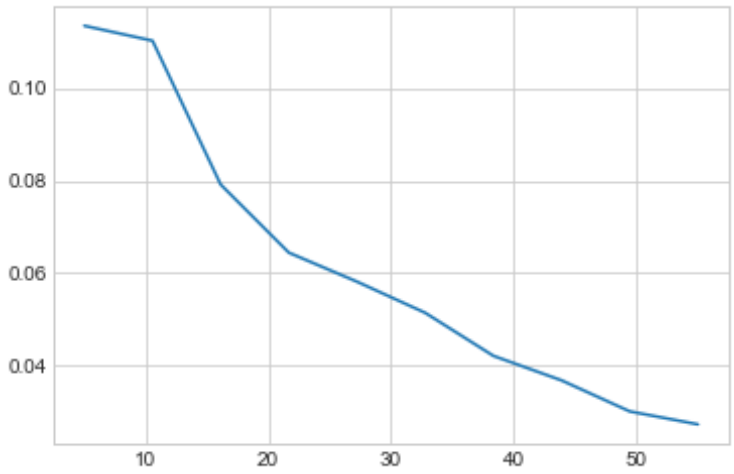

Figure 23: Relative Difference between ALAP and the Lower Bound, GPU case, for different sizes of n.

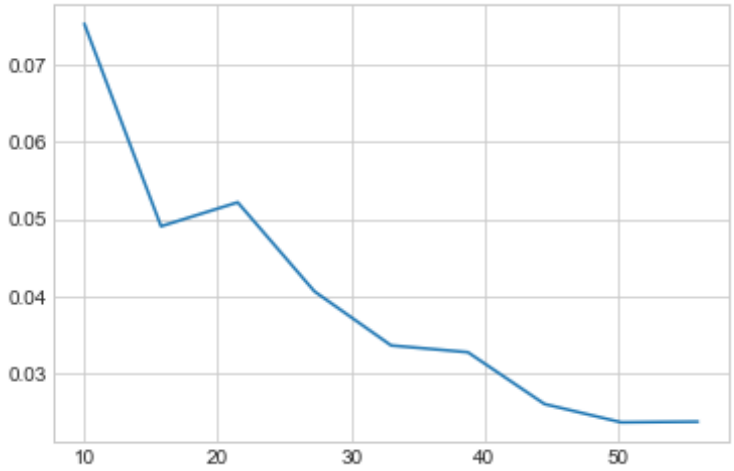

Figure 24: Relative Difference between ALAP and the Lower Bound, GPU case, for different sizes of n.

This 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 algorithm and better than ASAP scheduling in the case of CPUs. The proximity between the completion time of the ASAP algorithm and the bound, in all investigated situations, also allows us to validate the quality of the lower bound obtained.

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 bounds are extremely coarse and do not allow to assess the efficiency of the scheduling algorithms.

References

[1] Agullo, E., Beaumont, O., Eyraud-Dubois, L., Herrmann, J., Kumar, S., Marchal, L., and Thibault, S. Bridging the gap between performance and bounds of Cholesky factorization on heterogeneous platforms. In *HCW'15* (2015).

[2] Agullo, E., Beaumont, O., Eyraud-Dubois, L., and Kumar, S. Are static schedules so bad? a case study on cholesky factorization. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2016), IEEE, pp. 1021–1030.

[3] Agullo, E., Hadri, B., Ltaief, H., and Dongarra, J. Comparative study of one-sided factorizations with multiple software packages on multi-core hardware. In *SC'09. ACM/IEEE Conference on Supercomputing, Portland, OR, November* (2009).

[4] Augonnet, C., Thibault, S., Namyst, R., and Wacrenier, P.-A. StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. *Concurrency and Computation: Practice and Experience, Special Issue: Euro-Par 2009 23* (Feb. 2011), 187–198.

[5] Badia, R. M., Herrero, J. R., Labarta, J., Pérez, J. M., Quintana-Ortí, E. S., and Quintana-Ortí, G. Parallelizing dense and banded linear algebra libraries using SMPSs. *Concurrency and Computation: Practice and Experience 21*, 18 (2009), 2438–2456.

[6] Beaumont, O., Canon, L.-c., Eyraud-Dubois, L., Lucarelli, G., Marchal, L., Mommessin, C., Simon, B., and Trystram, D. Scheduling on two types of resources: a survey. *arXiv preprint arXiv:1909.11365* (2019).

[7] Bosilca, G., Bouteiller, A., Danalis, A., Herault, T., Lemarinier, P., and Dongarra, J. DAGuE: A generic distributed dag engine for high performance computing. *Parallel Computing 38*, (1-2) (2012), 37–51.

[8] Chan, E., Van Zee, F. G., Bientinesi, P., Quintana-Ortí, E. S., Quintana-Ortí, G., and van de Geijn, R. Supermatrix: a multithreaded runtime scheduling system for algorithms-by-blocks. In *PPoPP '08* (2008), ACM, pp. 123–132.

[9] Cojean, T., Guermouche, A., Hugo, A., Namyst, R., and Wacrenier, P.-A. Resource aggregation for task-based cholesky factorization on top of modern architectures. *Parallel Computing 83* (2019), 73–92.

[10] Cosnard, M., Marrakchi, M., Robert, Y., and Trystram, D. Parallel gaussian elimination on an MIMD computer. *Parallel Computing 6*, 3 (1988), 275–296.

[11] Gautier, T., Besseron, X., and Pigeon, L. KAAPI: A thread scheduling runtime system for data flow computations on cluster of multi-processors. In *PASCO'07* (London, Ontario, Canada, July 2007).

[12] Gustavson, F., Karlsson, L., and Kågström, B. Distributed SBP Cholesky factorization algorithms with near-optimal scheduling. *ACM T. Math. Software 36*, 2 (2009), 1–25.

[13] Jacquelin, M., Zheng, Y., Ng, E., and Yelick, K. An asynchronous task-based fan-both sparse cholesky solver. *arXiv preprint arXiv:1608.00044* (2016).

[14] Kim, K., Rajamanickam, S., Stelle, G., Edwards, H. C., and Olivier, S. L. Task parallel incomplete cholesky factorization using 2d partitioned-block layout. *arXiv preprint arXiv:1601.05871* (2016).

[15] Kurzak, J., Buttari, A., and Dongarra, J. Solving systems of linear equations on the CELL processor using Cholesky factorization. *IEEE Trans. Parallel Distrib. Syst. 19*, 9 (2008), 1175–1186.

[16] Kurzak, J., Ltaief, H., Dongarra, J., and Badia, R. M. Scheduling dense linear algebra operations on multicore processors. *Concurrency and Computation: Practice and Experience 22*, 1 (2010), 15–44.

[17] Marrakchi, M., and Robert, Y. Optimal algorithms for gaussian elimination on a MIMD computer. *Parallel Computing 12* (1989), 183–194.

[18] Pérez, J. M., Badia, R. M., and Labarta, J. A flexible and portable programming model for SMP and multi-cores. Tech. rep., Barcelona Supercomputing Center – Centro Nacional de Supercomputacioń, June 2007.

[19] Quintana-Ortí, E. S., Quintana-Ortí, G., van de Geijn, R. A., Van Zee, F. G., and Chan, E. Programming matrix algorithms-by-blocks for thread-level parallelism. vol. 36.

[20] Robert, Y., and Trystram, D. Optimal scheduling algorithms for parallel gaussian elimination. *Theoretical Computer Science 64*, 2 (1989), 159 – 173.

[21] Song, F., YarKhan, A., and Dongarra, J. Dynamic task scheduling for linear algebra algorithms on distributed-memory multicore systems. In *SC'09* (2009).

[22] YarKhan, A., Kurzak, J., and Dongarra, J. QUARK users' guide: Queueing and runtime for kernels. Tech. Rep. ICL-UT-11-02, University of Tennessee, Innovative Computing Laboratory, 2011.