Scheduling Trees of Malleable Tasks for Sparse Linear Algebra

Abdou Guermouche, Loris Marchal, Bertrand Simon, Frédéric Vivien

Scheduling Trees of Malleable Tasks for Sparse Linear Algebra

Abdou Guermouche[1], Loris Marchal[2], Bertrand Simon[2], and Frédéric Vivien[2]

[1] University of Bordeaux and INRIA,
200 rue de la vieille Tour, Talence, France
abdou.guermouche@labri.fr
[2] CNRS, INRIA and University of Lyon,
LIP, ENS Lyon, 46 allée d'Italie, Lyon, France
{loris.marchal,bertrand.simon,frederic.vivien}@ens-lyon.fr

Abstract. Scientific workloads are often described by directed acyclic task graphs. This is in particular the case for multifrontal factorization of sparse matrices —the focus of this paper— whose task graph is structured as a tree of parallel tasks. Prasanna and Musicus [19,20] advocated using the concept of *malleable* tasks to model parallel tasks involved in matrix computations. In this powerful model each task is processed on a time-varying number of processors. Following Prasanna and Musicus, we consider malleable tasks whose speedup is p^α, where p is the fractional share of processors on which a task executes, and α ($0 < \alpha \leq 1$) is a task-independent parameter. Firstly, we use actual experiments on multicore platforms to motivate the relevance of this model for our application. Then, we study the optimal time-minimizing allocation proposed by Prasanna and Musicus using optimal control theory. We greatly simplify their proofs by resorting only to pure scheduling arguments. Building on the insight gained thanks to these new proofs, we extend the study to distributed (homogeneous or heterogeneous) multicore platforms. We prove the NP-completeness of the corresponding scheduling problem, and we then propose some approximation algorithms.

1 Introduction

Parallel workloads are often modeled as directed acyclic task graphs, or DAGs, where nodes represent tasks and edges represent dependencies between tasks. Task graphs arise from many scientific domains, such as image processing, genomics, and geophysical simulations. In this paper, we focus on task graphs coming from sparse linear algebra, and especially from the factorization of sparse matrices using the multifrontal method. Liu [18] explains that the computational dependencies and requirements in Cholesky and LU factorization of sparse matrices using the multifrontal method can be modeled as a task tree, called the *assembly tree*. We therefore focus on dependencies that can be modeled as a tree.

This work was supported by the ANR SOLHAR project funded by the French National Research Agency.

In the abundant existing literature, several variants of the task graph scheduling problem are addressed, depending on the ability to process a task in parallel: tasks are either *sequential* (not amenable to parallel processing), *rigid* (requesting a given number of processors), *moldable* (able to cope with any fixed number of processors) or even *malleable* (processed on a variable number of processors) in the terminology of Drozdowski [6, chapter 25]. When considering moldable and malleable tasks, one has to define how the processing time of a task depends on the number of allocated processors. Under some general assumptions, Jansen and Zhang [14] derive a 3.29 approximation algorithm for arbitrary precedence constraints, which is improved in a 2.62 approximation in the particular case of a series-parallel precedence graph by Lepere et al. [16]. However, although polynomial, these algorithms relies on complex optimization techniques, which makes them difficult to implement in a practical setting.

In this study, we consider a special case of malleable tasks, where the speedup function of each task is p^α, where p is the number of processors allocated to the task, and $0 < \alpha \leq 1$ is a global parameter. In particular, when the share of processors p_i allocated to a task T_i is constant, its processing time is given by L_i/p_i^α, where L_i is the sequential duration of T_i. The case $\alpha = 1$ represents the unrealistic case of a perfect linear speed-up, and we rather concentrate on the case $\alpha < 1$ which takes into consideration the cost of the parallelization. In particular $\alpha < 1$ accounts for the cost of intra-task communications, without having to decompose the tasks in smaller granularity sub-tasks with explicit communications, which would make the scheduling problem intractable. This model has been advocated by Prasanna and Musicus [20] for matrix operations, and we present some new motivation for this model in our context. As in [20], we also assume that it is possible to allocate non-integer shares of processors to tasks. This amounts to assume that processors can share their processing time among tasks. When task A is allocated 2.6 processors and task B 3.4 processors, one processor dedicates 60% of its time to A and 40% to B. Note that this is a realistic assumption, for example, when using modern task-based runtime systems such as StarPU [3], KAAPI [9], or PaRSEC [4]. This allows to simplify the scheduling problem and to derive optimal allocation algorithms.

Our objective is to minimize the total processing time of a tree of malleable tasks. Initially, we consider a homogeneous platform composed of p identical processors. To achieve our goal, we take advantage of two sources of parallelism: the *tree parallelism* which allows tasks independent from each others (such as siblings) to be processed concurrently, and the *task parallelism* which allows a task to be processed on several processors. A solution to this problem describes both in which order tasks are processed and which share of computing resources is allocated to each task.

In [19,20], the same problem has been addressed by Prasanna and Musicus for series-parallel graphs (or SP-graphs). Such graphs are built recursively as series or parallel composition of two smaller SP-graphs. Trees can be seen as a special-case of series-parallel graphs, and thus, the optimal algorithm proposed in [19,20] is also valid on trees. They use optimal control theory to derive general

theorems for any strictly increasing speedup function. For the particular case of the speedup function p^α, Prasanna and Musicus prove some properties of the unique optimal schedule which allow to compute it efficiently. Their results are powerful (a simple optimal solution is proposed), but to obtain these results they had to transform the problem in a shape which is amenable to optimal control theory. Thus, their proofs do not provide any intuition on the underlying scheduling problem, yet it seems tractable using classic scheduling arguments.

In this paper, our contributions are the following:

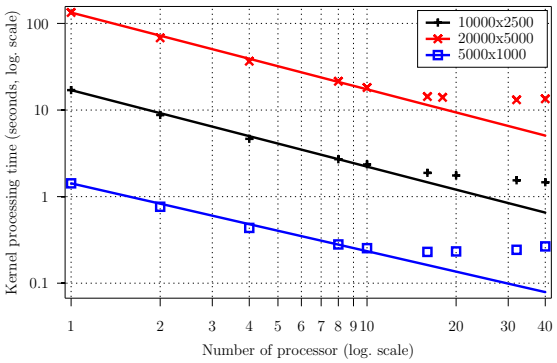
- In Sect. 2, we show that the model of malleable tasks using the p^α speed-up function is justified in the context of sparse matrix factorization.
- In Sect. 4, we propose a new and simpler proof for the results of [19,20] on series-parallel graphs, using pure scheduling arguments.
- In Sect. 5, we extend the previous study on distributed memory machines, where tasks cannot be distributed across several distributed nodes. We provide NP-completeness results and approximation algorithms.

2 Validation of the Malleable Task Model

In this section, we evaluate the model proposed by Prasanna and Musicus in [19,20] for our target application. This model states that the instantaneous speedup of a task processed on p processors is p^α. Thus, the processing time of a task T_i of size L_i which is allocated a share of processors $p_i(t)$ at time t is equal to the smallest value C_i such that $\int_0^{C_i} (p_i(t))^\alpha \, dt \geq L_i$, where α is a task-independent constant. When the share of processors p_i is constant, $C_i = L_i/p_i^\alpha$. Our goal is (i) to find whether this formula well describes the evolution of the task processing time for various shares of processors and (ii) to check that different tasks of the same application have the same α parameter. We target a modern multicore platform composed of a set of nodes each including several multicore processors. For the purpose of this study we restrict ourselves to the single node case for which the communication cost will be less dominant. In this context, $p_i(t)$ denotes the number of *cores* dedicated to task T_i at time t.

We consider applications having a tree-shaped task graph constituted of parallel tasks. This kind of execution model can be met in sparse direct solvers where the matrix is first factorized before the actual solution is computed. For instance, either the multifrontal method [7] as implemented in `MUMPS` [1] or `qr_mumps` [5], or the supernodal approach as implemented in `SuperLU` [17] or in `PaStiX` [12], are based on tree-shaped task graphs (namely the assembly tree [2]). Each task in this tree is a partial factorization of a dense sub-matrix or of a sparse panel. In order to reach good performance, these factorizations are performed using tiled linear algebra routines (BLAS): the sub-matrix is decomposed into 2D tiles (or blocks), and optimized BLAS kernels are used to perform the necessary operations on each tile. Thus, each task can be seen as a task graph of smaller granularity sub-tasks.

As computing platforms evolve quickly and become more complex (e.g., because of the increasing use of accelerators such as GPUs or Xeon Phis), it be-



(a) Timings and model (lines) with 1D partitioning

matrix	1D	2D
5000x1000	0.78	0.93
10000x2500	0.88	0.95
20000x5000	0.89	0.94

(b) Values of α

Fig. 1. Timings and α values for `qr_mumps` frontal matrix factorization kernel

comes interesting to rely on an optimized dynamic runtime system to allocate and schedule tasks on computing resources. These runtime systems (such as StarPU [3], KAAPI [9], or PaRSEC [4]) are able to process a task on a prescribed subset of the computing cores that may evolve over time. This motivates the use of the malleable task model, where the share of processors allocated to a task vary with time. This approach has been recently used and evaluated [13] in the context of the `qr_mumps` solver using the StarPU runtime system.

In order to assess whether tasks used within sparse direct solvers fit the model introduced by Prasanna and Musicus in [20] we conducted an experimental study on several dense linear algebra tasks. We used a test platform composed of 4 Intel E7-4870 processors having 10 cores each clocked at 2.40 GHz and having 30 MB of L3 cache for a total of 40 cores. The platform is equipped with 1 TB of memory with uniform access. We considered dense operations which are representative of what can be met in sparse linear algebra computations, namely the standard frontal matrix factorization kernel used in the `qr_mumps` solver. We used either block-columns of size 32 (1D partitioning) or square blocks of size 256 (2D partitioning). All experiments were made using the StarPU runtime.

Figure 1(a) presents the timings obtained when processing the `qr_mumps` frontal matrix factorization kernel on a varying number of processors. The logarithmic scales show that the p^α speedup function models well the timings, except for small matrices when p is large. In those cases, there is not enough parallelism in tasks to exploit all available cores. We performed linear regressions on the portions where $p \leq 10$ to compute α for different task sizes (Fig. 1(b)). We performed the same test for 2D partitioning and computed the corresponding α values (using $p \leq 20$). We notice that the value of α does not vary significantly with the matrix size, which validates our model. The only notable exception is for the smallest matrix (5000x1000) with 1D partitioning: it is hard to efficiently use many cores for such small matrices. In all cases, when the number of processors is larger than a threshold the performance deteriorates and stalls. Our

speedup model is only valid below this threshold, which threshold increases with the matrix size. This is not a problem as the allocation schemes developed in the next sections allocate large numbers of processors to large tasks at the top of the tree and smaller numbers of processors for smaller tasks. In other words, we produce allocations that always respect the validity thresholds of the model. Finally, note that the value of α depends on the parameters of the problem (type of factorization, partitioning, block size, etc.). It has to be determined for each kernel and each set of blocking parameters.

3 Model and Notations

We assume that the number of available computing resources may vary with time: $p(t)$ gives the (possibly rational) total number of processors available at time t, also called the processor profile. For the sake of simplicity, we consider that $p(t)$ is a step function. Although our study is motivated by an application running on a single multicore node (as outlined in the previous section), we use the term *processor* instead of *computing core* in the following sections for readability and consistency with the scheduling literature.

We consider an in-tree G of n malleable tasks $T_1, \ldots, T_n$. L_i denotes the length, that is the sequential processing time, of task T_i. As motivated in the previous section, we assume that the speedup function for a task allocated p processors is p^α, where $0 < \alpha \leq 1$ is a fixed parameter. A schedule $\mathcal{S}$ is a set of nonnegative piecewise continuous functions $\left\{p_i(t) \;\middle|\; i \in I\right\}$ representing the time-varying share of processors allocated to each task. During a time interval Δ, the task T_i performs an amount of work equal to $\int_\Delta p_i(t)^\alpha \, \mathrm{d}t$. Then, T_i is completed when the total work performed is equal to its length L_i. The completion time of task T_i is thus the smallest value C_i such that $\int_0^{C_i} p_i(t)^\alpha dt \geq L_i$. We define $w_i(t)$ as the ratio of the work of the task T_i that is done during the time interval $[0, t]$: $w_i(t) = \int_0^t p_i(x)^\alpha \, \mathrm{d}x / L_i$. A schedule is a valid solution if and only if:

- it does not use more processors than available: $\forall t, \sum_{i \in I} p_i(t) \leq p(t)$;
- it completes all the tasks: $\exists \tau, \ \forall i \in I, \quad w_i(\tau) = 1$;
- and it respects precedence constraints: $\forall i \in I, \forall t$, if $p_i(t) > 0$ then, $\forall j \in I$, if j is a child of i, $w_j(t) = 1$.

The makespan τ of a schedule is computed as $\min\{t \mid \forall i \, w_i(t) = 1\}$. Our objective is to construct a valid schedule with optimal, i.e., minimal, makespan.

Note that because of the speedup function p^α, the computations in the following sections will make a heavy use of the functions $f : x \mapsto x^\alpha$ and $g : x \mapsto x^{(1/\alpha)}$. We assume that we have at our disposal a polynomial time algorithm to compute both f and g. We are aware that this assumption is very likely to be wrong, as soon as $\alpha < 1$, since f and g produce irrational numbers. However, without these functions, it is not even possible to compute the makespan of a schedule in polynomial time and, hence, the problem is not in NP. Furthermore, this allows us to avoid the complexity due to number computations, and to concentrate on the most interesting combinatorial complexity, when proving NP-completeness

results and providing approximation algorithms. In practice, any implementation of f and g with a reasonably good accuracy will be sufficient to perform all computations including the computation of makespans.

In the next section, following Prasanna and Musicus, we will not consider trees but more general graphs: *series-parallel graphs* (or SP graphs). An SP graph is recursively defined as a single task, the series composition of two SP graphs, or the parallel composition of two SP graphs. A tree can easily be transformed into an SP graph by joining the leaves according to its structure, the resulting graph is then called a *pseudo-tree*. We will use $(i \parallel j)$ to represent the parallel composition of tasks T_i and T_j and $(i\,;j)$ to represent their series composition. Thanks to the construction of pseudo-trees, an algorithm which solves the previous scheduling problem on SP-graphs also gives an optimal solution for trees.

4 Optimal Solution for Shared-Memory Platforms

The purpose of this section is to give a simpler proof of the results of [19,20] using only scheduling arguments. We consider an SP-graph to be scheduled on a shared-memory platform (each task can be distributed across the whole platform). We assume that $\alpha < 1$ and prove the uniqueness of the optimal schedule.

Our objective is to prove that any SP graph G is *equivalent* to a single task T_G of easily computable length: for any processor profile $p(t)$, graphs G and T_G have the same makespan. We prove that the ratio of processors allocated to any task T_i, defined by $r_i(t) = p_i(t)/p(t)$, is constant from the moment at which T_i is initiated to the moment at which it is terminated. We also prove that in an optimal schedule, the two subgraphs of a parallel composition terminate at the same time and each receives a constant total ratio of processors throughout its execution. We then prove that these properties imply that the optimal schedule is unique and obeys to a *flow conservation* property: the shares of processors allocated to two subgraphs of a series composition are equal. When considering a tree, this means that the whole schedule is defined by the ratios of processors allocated to the leaves. Then, all the children of a node T_i terminate at the same time, and its ratio is the sum of its children ratios.

We first need to define the length $\mathcal{L}_G$ associated to a graph G, which will be proved to be the length of the task T_G. Then, we state a few lemmas before proving the main theorem. We only present here sketches of the proofs, the detailed versions can be found in [10].

Definition 1 *We recursively define the length $\mathcal{L}_G$ associated to a SP graph G:*

- $\mathcal{L}_{T_i} = L_i$
- $\mathcal{L}_{G_1\,;\,G_2} = \mathcal{L}_{G_1} + \mathcal{L}_{G_2}$
- $\mathcal{L}_{G_1 \parallel G_2} = \left(\mathcal{L}_{G_1}^{1/\alpha} + \mathcal{L}_{G_2}^{1/\alpha}\right)^{\alpha}$

Lemma 1. *An allocation minimizing the makespan uses all the processors at any time.*

We call a *clean interval* with regard to a schedule $\mathcal{S}$ an interval during which no task is completed in $\mathcal{S}$.

Lemma 2. *When the number of available processors is constant, any optimal schedule allocates a constant number of processors per task on any clean interval.*

Proof. By contradiction, we assume that there exists an optimal schedule $\mathcal{P}$ of makespan M, a task T_j and a clean interval $\Delta = [t_1, t_2]$ such that T_j is not allocated a constant number of processors on Δ. By definition of clean intervals, no task completes during Δ. $|\Delta| = t_2 - t_1$ denotes the duration of Δ, I the set of tasks that receive a non-empty share of processors during Δ, and p the constant number of available processors.

We want to show that there exists a valid schedule with a makespan smaller than M. To achieve this, we define an intermediate and not necessarily valid schedule $\mathcal{Q}$, which nevertheless respects the resource constraints (no more than p processors are used at time t). This schedule is equal to $\mathcal{P}$ except on Δ. The constant share of processors allocated to task T_i on Δ in $\mathcal{Q}$ is defined by $q_i = \frac{1}{|\Delta|}\int_\Delta p_i(t)dt$. For all t, we have $\sum_{i\in I} p_i(t) = p$ because of Lemma 1. We get $\sum_{i\in I} q_i = p$. So $\mathcal{Q}$ respects the resource constraints. Let $W_i^\Delta(\mathcal{P})$ (resp. $W_i^\Delta(\mathcal{Q})$) denote the work done on T_i during Δ under schedule $\mathcal{P}$ (resp. $\mathcal{Q}$). We have

$$W_i^\Delta(\mathcal{P}) = \int_\Delta p_i(t)^\alpha dt = |\Delta| \int_{[0,1]} p_i(t_1 + t|\Delta|)^\alpha dt$$
$$W_i^\Delta(\mathcal{Q}) = \int_\Delta \left(\frac{1}{|\Delta|}\int_\Delta p_i(t)dt\right)^\alpha dx = |\Delta| \left(\int_{[0,1]} p_i(t_1 + t|\Delta|)dt\right)^\alpha$$

As $\alpha < 1$, the function $x \mapsto x^\alpha$ is concave and then, by Jensen inequality [11], $W_i^\Delta(\mathcal{P}) \leq W_i^\Delta(\mathcal{Q})$. Moreover, as $x \mapsto x^\alpha$ is *strictly* concave, this inequality is an equality if and only if the function $t \mapsto p_i(t_1 + t|\Delta|)$ is equal to a constant on $[0, 1[$ except on a subset of $[0, 1[$ of null measure [11]. Then, by definition, p_j is not constant on Δ, and cannot be made constant by modifications on a set of null measure. We thus have $W_j^\Delta(\mathcal{P}) < W_j^\Delta(\mathcal{Q})$. Therefore, T_j is allocated too many processors under $\mathcal{Q}$. It is then possible to distribute this surplus among the other tasks during Δ, so that the work done during Δ in $\mathcal{P}$ can be terminated earlier. This remark implies that there exists a valid schedule with a makespan smaller than M; hence, the contradiction. □

We recall that $r_i(t) = p_i(t)/p(t)$ is the instantaneous ratio of processors allocated to a task T_i .

Lemma 3. *Let G be the parallel composition of two tasks, T_1 and T_2. If $p(t)$ is a step function, in any optimal schedule $r_1(t)$ is constant and equal to $\pi_1 = 1 \Big/ \left(1 + (L_2/L_1)^{1/\alpha}\right) = L_1^{1/\alpha} \Big/ \mathcal{L}_{1\parallel 2}^{1/\alpha}$ up to the completion of G.*

Proof. First, we prove that $r_1(t)$ is constant on any optimal schedule.

We consider an optimal schedule $\mathcal{S}$, and two consecutive time intervals A and B such that $p(t)$ is constant and equal to p on A and q on B, and $\mathcal{S}$ does not complete before the end of B. Suppose also that $|A|p^\alpha = |B|q^\alpha$ (shorten one interval otherwise), where $|A|$ and $|B|$ are the durations of intervals A and

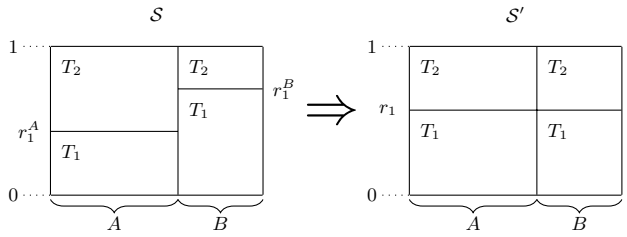


Fig. 2. Schedules $\mathcal{S}$ and $\mathcal{S}'$ on $A \cup B$. The abscissae represent the time and the ordinates the ratio of processing power

B. By Lemma 2, $r_1(t)$ has constant values r_1^A on A and r_1^B on B. Suppose by contradiction that $r_1^A \neq r_1^B$.

We want to prove that $\mathcal{S}$ is not optimal, and so that we can do the same work as $\mathcal{S}$ does on $A \cup B$ in a smaller makespan. We set $r_1 = \left(r_1^A + r_1^B\right)/2$. We define the schedule $\mathcal{S}'$ as equal to $\mathcal{S}$ except on $A \cup B$ where the ratio allocated to T_1 is r_1 (see Fig. 2).
The work W_1 on task T_1 under $\mathcal{S}$ and W_1' under $\mathcal{S}'$ during $A \cup B$ are equal to:

$$W_1 = |A|p^\alpha \left(r_1^A\right)^\alpha + |B|q^\alpha \left(r_1^B\right)^\alpha \qquad W_1' = r_1^\alpha \left(|A|p^\alpha + |B|q^\alpha\right)$$

Then, with the concavity inequality and the fact that $|B|q^\alpha = |A|p^\alpha$, we can deduce that $W_1' > W_1$ and symmetrically that $W_2' > W_2$.

Therefore, $\mathcal{S}'$ performs strictly more work for each task during $A \cup B$ than $\mathcal{S}$. Thus, as in Lemma 2, $\mathcal{S}$ is not optimal. So $r_1(t)$ is constant in optimal schedules.

There remains to prove that in an optimal schedule $\mathcal{S}$, $r_1(t) = \pi_1$; hence, the optimal schedule is unique. As $p(t)$ is a step function, we define the sequences (A_k) and (p_k) such that A_k is the duration of the k-th step of the function $p(t)$ and $p(t) = p_k > 0$ on A_k. The sum of the durations of the A_k's is the makespan of $\mathcal{S}$. Then, if we note $V = \sum_k |A_k| p_k^\alpha$ and r_1 the value of $r_1(t)$, we have:

$$L_1 = \sum_k |A_k| r_1^\alpha p_k^\alpha = r_1^\alpha V \qquad \text{and} \qquad L_2 = \sum_k |A_k| (1 - r_1)^\alpha p_k^\alpha = (1 - r_1)^\alpha V$$

Then, $r_1 = 1 \Big/ \left(1 + (L_2/L_1)^{1/\alpha}\right) = \pi_1$. □

Lemma 4. *Let G be the parallel composition of tasks T_1 and T_2, with $p(t)$ a step function, and $\mathcal{S}$ an optimal schedule. Then, the makespan of G under $\mathcal{S}$ is equal to the makespan of the task T_G of length $\mathcal{L}_G = \mathcal{L}_{1\,\|\,2}$.*

Proof. We characterize $p(t)$ by the sequences (A_k) and (p_k) as in the proof of Lemma 3. We know by Lemma 3 that the share allocated to T_1 is constant and equal to $\pi_1 p_k$ on each interval A_k. Then, by summing the work done on each interval for both tasks, one can prove that they are completed simultaneously, and that this completion time is the same as that of task T_G under the same processor profile. □

Theorem 2. *For every graph G, if $p(t)$ is a step function, G has the same optimal makespan as its equivalent task T_G of length $\mathcal{L}_G$ (computed as in Definition 1). Moreover, there is a unique optimal schedule, and it can be computed in polynomial time.*

Proof. In this proof, we only consider optimal schedules. Therefore, when the makespan of a graph is considered, this is implicitly its optimal makespan. We first remark that in any optimal schedule, as $p(t)$ is a step function and because of Lemma 2, only step functions are used to allocate processors to tasks, and so Lemma 4 can be applied on any subgraph of G without checking that the processor profile is also a step function for this subgraph. We now prove the result by induction on the structure of G.

- G is a single task. The result is immediate.
- G is the series composition of G_1 and G_2. By induction, G_1 (resp. G_2) has the same makespan as task T_{G_1} (resp. T_{G_2}) of length $\mathcal{L}_{G_1}$ (resp. $\mathcal{L}_{G_2}$) under any processor profile. Therefore, the makespan of G is equal to $\mathcal{L}_G = \mathcal{L}_{G_1\,;\,G_2} = \mathcal{L}_{G_1} + \mathcal{L}_{G_2}$. The unique optimal schedule of G under $p(t)$ processors is the concatenation of the optimal schedules of G_1 and G_2.
- G is the parallel composition of G_1 and G_2. By induction, G_1 (resp. G_2) has the same makespan as task T_{G_1} (resp. T_{G_2}) of length $\mathcal{L}_{G_1}$ (resp. $\mathcal{L}_{G_2}$) under any processor profile. Consider an optimal schedule $\mathcal{S}$ of G and let $p_1(t)$ be the processor profile allocated to G_1. Let $\tilde{\mathcal{S}}$ be the schedule of $(T_{G_1} \parallel T_{G_2})$ that allocates $p_1(t)$ processors to T_{G_1}. $\tilde{\mathcal{S}}$ is optimal and achieves the same makespan as $\mathcal{S}$ for G because T_{G_1} and G_1 (resp. T_{G_2} and G_2) have the same makespan under any processor profile. Then, by Lemma 4, $\tilde{\mathcal{S}}$ (so $\mathcal{S}$) achieves the same makespan as the optimal makespan of the task T_G of length $\mathcal{L}_{G_1 \parallel G_2} = \mathcal{L}_G$. Moreover, by Lemma 3 applied on $(T_{G_1} \parallel T_{G_2})$, we have $p_1(t) = \pi_1 p(t)$. By induction, the unique optimal schedules of G_1 and G_2 under respectively $p_1(t)$ and $(p(t) - p_1(t))$ processors can be computed. Therefore, there is a unique optimal schedule of G under $p(t)$ processor: the parallel composition of these two schedules.

Therefore, there is a unique optimal schedule for G under $p(t)$. Moreover, it can be computed in polynomial time. We describe here the algorithm to compute the optimal schedule of a tree G, but it can be extended to treat SP-graphs. The length of the equivalent task of each subtree of G can be computed in polynomial time by a depth-first search of the tree (assuming that raising a number to the power α or $1/\alpha$ can be done in polynomial time). Hence, the ratios π_1 and π_2 for each parallel composition can also be computed in polynomial time. Finally, these ratios imply the computation in linear time of the ratios of the processor profile that should be allocated to each task after its children are completed, which describes the optimal schedule. □

5 Extensions to Distributed Memory

The objective of this section is to extend the previous results to the case where the computing platform is composed of several nodes with their own private

memory. In order to avoid the large communication overhead of processing a task on cores distributed across several nodes, we forbid such a multi-node execution: the tasks of the tree can be distributed on the whole platform but each task has to be processed on a single node. We prove that this additional constraint, denoted by $\mathcal{R}$, renders the problem much more difficult. We concentrate first on platforms with two homogeneous nodes and then with two heterogeneous nodes.

5.1 Two Homogeneous Multicore Nodes

In this section, we consider a multicore platform composed of two equivalent nodes having the same number of computing cores p. We also assume that all the tasks T_i have the same speedup function p_i^α on both nodes. We first show that finding a schedule with minimum makespan is weakly NP-complete, even for independent tasks:

Theorem 3. *Given two homogenous nodes of p processors, n independent tasks of sizes $L_1, ..., L_n$ and a bound T, the problem of finding a schedule of the n tasks on the two nodes that respects $\mathcal{R}$, and whose makespan is not greater than T, is (weakly) NP-complete for all values of the α parameter defining the speedup function.*

The proof relies on the Partition problem, which is known to be weakly (i.e., binary) NP-complete [8], and uses tasks of length $L_i = a_i^\alpha$, where the a_i's are the numbers from the instance of the Partition problem. We recall that we assume that functions $x \mapsto x^\alpha$ and $x \mapsto x^{1/\alpha}$ can be computed in polynomial time. Details can be found in the companion research report [10].

We also provide a constant ratio approximation algorithm. We recall that a ρ-approximation provides on each instance a solution whose objective z is such that $z \leq \rho z*$, where $z*$ is the optimal value of the objective on this instance.

Theorem 4. *There exists a polynomial time $\left(\frac{4}{3}\right)^\alpha$-approximation algorithm for the makespan minimization problem when scheduling a tree of malleable tasks on two homogenous nodes.*

Due to lack of space, we refer the interested reader to the companion research report for the complete description of the algorithm and proof [10]. The proof of the approximation ratio consists in comparing the proposed solution to the optimal solution on a single node made of $2p$ processors, denoted $\mathcal{S}_{\text{PM}}$. Such an optimal solution can be computed as proposed in the previous section, and is a lower bound on the optimal makespan on 2 nodes with p processors. The general picture of the proposed algorithm is the following. First, the root of the tree is arbitrarily allocated to the p processors of one of the two nodes. Then, the subtrees S_i's rooted at the root's children are considered. If none of these subtrees is allocated more than p processors in $\mathcal{S}_{\text{PM}}$, then we show how to "pack" the subtrees on the two nodes and bound the slow-down by $\left(\frac{4}{3}\right)^\alpha$. On the contrary, if one of the S_i's is allocated more than p processors in $\mathcal{S}_{\text{PM}}$, then we allocate p processors to its root, and recursively call the algorithm on its children and on the remaining subtrees.

5.2 Two Heterogeneous Multicore Nodes

We suppose here that the computing platform is made of two processors of different processing capabilities: the first one is made of p cores, while the second one includes q cores. We also assume that the parameter α of the speedup function is the same on both processors. As the problem gets more complicated, we concentrate here on n independent tasks, of lengths $L_1, ..., L_n$. Thanks to the homogenous case presented above, we already know that scheduling independent tasks on two nodes is NP-complete.

This problem is close to the Subset Sum problem. Given n numbers, the optimization version of Subset Sum considers a target K and aims at finding the subset with maximal sum smaller than or equal to K. There exists many approximation schemes for this problem. In particular, Kellerer et al. [15] propose a fully polynomial approximation scheme (FPTAS). Based on this result, an approximation scheme can be derived for our problem.

Theorem 5. *There exists an FPTAS for the problem of scheduling independent malleable tasks on two heterogeneous nodes, provided that, for each task, $L_i^{1/\alpha}$ is an integer.*

The proof is complex and detailed in [10]. The assumption on the $L_i^{1/\alpha}$s is needed to apply the FPTAS of Subset Sum, which is valid only on integers.

6 Conclusion

In this paper, we have studied how to schedule trees of malleable tasks whose speedup function on multicore platforms is p^α. We have first motivated the use of this model for sparse matrix factorizations by actual experiments. When using factorization kernels actually used in sparse solvers, we show that the speedup follows the p^α model for reasonable allocations. On the machine used for our tests, α is in the range 0.85–0.95. Then, we proposed a new proof of the optimal allocation derived by Prasanna and Musicus [19,20] for such trees on single node multicore platforms. Contrarily to the use of optimal control theory of the original proofs, our method relies only on pure scheduling arguments and gives more intuitions on the scheduling problem. Based on these proofs, we proposed several extensions for two multicore nodes: we prove the NP-completeness of the scheduling problem and propose a $\left(\frac{4}{3}\right)^\alpha$-approximation algorithm for a tree of malleable tasks on two homogeneous nodes, and an FPTAS for independent malleable tasks on two heterogeneous nodes.

The perspectives to extend this work follow two main directions. First, it would be interesting to extend the approximations proposed for the heterogeneous case to a number of nodes larger than two, and to more heterogeneous nodes, for which the value of α differs from one node to another. This is a promising model for the use of accelerators (such as GPU or Xeon Phi). The second direction concerns an actual implementation of the PM allocation scheme in a sparse solver.

References

1. Amestoy, P., Buttari, A., Duff, I.S., Guermouche, A., L'Excellent, J., Uçar, B.: Mumps. In: Padua, D.A. (ed.) Encyclopedia of Parallel Computing, pp. 1232–1238. Springer (2011)
2. Ashcraft, C., Grimes, R.G., Lewis, J.G., Peyton, B.W., Simon, H.D.: Progress in sparse matrix methods for large linear systems on vector computers. Int. Journal of Supercomputer Applications 1(4), 10–30 (1987)
3. Augonnet, C., Thibault, S., Namyst, R., Wacrenier, P.A.: StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. Concurrency and Computation: Practice and Experience 23(2), 187–198 (2011)
4. Bosilca, G., Bouteiller, A., Danalis, A., Faverge, M., Herault, T., Dongarra, J.J.: PaRSEC: Exploiting heterogeneity for enhancing scalability. Computing in Science & Engineering 15(6), 36–45 (2013)
5. Buttari, A.: Fine granularity sparse QR factorization for multicore based systems. In: Int. Conf. on Applied Parallel and Scientific Computing. pp. 226–236 (2012)
6. Drozdowski, M.: Scheduling parallel tasks – algorithms and complexity. In: Leung, J. (ed.) Handbook of Scheduling. Chapman and Hall/CRC (2004)
7. Duff, I.S., Reid, J.K.: The multifrontal solution of indefinite sparse symmetric linear systems. ACM Transactions on Mathematical Software 9, 302–325 (1983)
8. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA (1979)
9. Gautier, T., Besseron, X., Pigeon, L.: Kaapi: A thread scheduling runtime system for data flow computations on cluster of multi-processors. In: International Workshop on Parallel Symbolic Computation. pp. 15–23 (2007)
10. Guermouche, A., Marchal, L., Simon, B., Vivien, F.: Scheduling trees of malleable tasks for sparse linear algebra. Tech. Rep. RR-8616, INRIA (Oct 2014)
11. Hardy, G., Littlewood, J., Pólya, G.: Inequalities, chap. 6.14. Cambridge Mathematical Library, Cambridge University Press (1952)
12. Hénon, P., Ramet, P., Roman, J.: PaStiX: A High-Performance Parallel Direct Solver for Sparse Symmetric Definite Systems. Par. Comp. 28(2), 301–321 (2002)
13. Hugo, A., Guermouche, A., Wacrenier, P.A., Namyst., R.: A runtime approach to dynamic resource allocation for sparse direct solvers. In: ICPP. pp. 481–490 (2014)
14. Jansen, K., Zhang, H.: Scheduling malleable tasks with precedence constraints. In: ACM Symp. on Par. in Algorithms and Architectures (SPAA). pp. 86–95 (2005)
15. Kellerer, H., Mansini, R., Pferschy, U., Speranza, M.G.: An efficient fully polynomial approximation scheme for the subset-sum problem. Journal of Computer and System Sciences 66(2), 349–370 (2003)
16. Lepère, R., Trystram, D., Woeginger, G.J.: Approximation algorithms for scheduling malleable tasks under precedence constraints. IJFCS 13(4), 613–627 (2002)
17. Li, X.S.: An overview of SuperLU: Algorithms, implementation, and user interface. ACM Transactions on Mathematical Software 31(3), 302–325 (September 2005)
18. Liu, J.W.H.: The role of elimination trees in sparse factorization. SIAM Journal on Matrix Analysis and Applications 11(1), 134–172 (1990)
19. Prasanna, G.N.S., Musicus, B.R.: Generalized multiprocessor scheduling and applications to matrix computations. IEEE TPDS 7(6), 650–664 (1996)
20. Prasanna, G.N.S., Musicus, B.R.: The optimal control approach to generalized multiprocessor scheduling. Algorithmica 15(1), 17–49 (1996)