Scheduling malleable task trees

Loris Marchal, Frédéric Vivien, Bertrand Simon

Scheduling malleable task trees

Bertrand SIMON, Loris MARCHAL, Frédéric VIVIEN

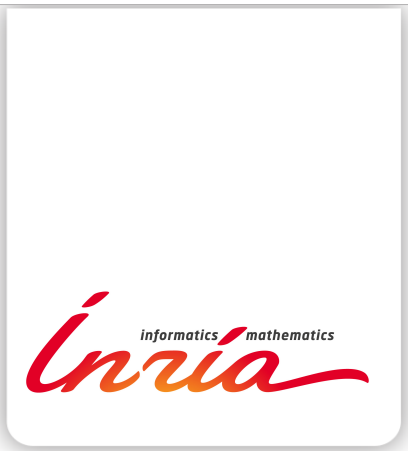
RESEARCH REPORT

N° 8587

September 2014

Project-Team ROMA

ISSN 0249-6399 ISRN INRIA/RR--8587--FR+ENG



Scheduling malleable task trees

Bertrand SIMON*, Loris MARCHAL†, Frédéric VIVIEN ‡

Project-Team ROMA

Research Report n° 8587 — September 2014 — 29 pages

Abstract: Solving sparse linear systems can lead to processing tree workflows on a platform of processors. In this study, we use the model of malleable tasks motivated in [1, 9] in order to study tree workflow schedules under two contradictory objectives: makespan minimization and memory minization. First, we give a simpler proof of the result of [8] which allows to compute a makespan-optimal schedule for tree workflows. Then, we study a more realistic speed-up function and show that the previous schedules are not optimal in this context. Finally, we give complexity results concerning the objective of minimizing both makespan and memory.

Key-words: Scheduling, Malleable tasks, Task trees, Makespan Minimization, Memory minimization

* Bertrand SIMON is with ENS de Lyon, France. E-mail: bertrand.simon@ens-lyon.fr
† Loris MARCHAL is with CNRS, France. E-mail: loris.marchal@ens-lyon.fr
‡ Frédéric VIVIEN is with Inria, France. E-mail: frederic.vivien@inria.fr

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Ordonnancement de tâches malléables

Résumé : Résoudre des systèmes linéaires creux peut nécessiter le traitement d'arbres de tâches sur une plate-forme de processeurs. Dans cette étude, nous utilisons le modèle de tâches malléables motivé dans [1, 9] pour étudier l'ordonnancement d'arbre de tâches sous plusieurs perspectives. Tout d'abord, nous proposons une preuve plus simple d'un résultat de [8] qui permet de calculer simplement un ordonnancement optimal d'un arbre de tâches malléables pour le temps de complétion. Nous étudions également un modèle de fonction d'accélération plus réaliste, et montrons que l'ordonnancement précédent n'est plus optimal dans ce contexte. Enfin, nous proposons ensuite des résultats de complexité pour le problème de la minimisation simultanée du temps de complétion et de la mémoire.

Mots-clés : Ordonnancement, Tâches malléables, Arbre de tâches, Minimisation du makespan, Minimisation de la mémoire

Contents

1 Introduction

Solving sparse linear systems requires fast sparse matrix factorizations, which then require execution of tree workflows on a platform of processors. A tree workflow is a rooted tree where each node represents a task to be executed, and in which a node cannot be executed before its children. There are two sources of parallelism in the processing of these workflows: the tree parallelism which allows tasks independent from each other (such as siblings) to be processed concurrently, and the task parallelism which allows a task to be processed on several processors. This study aims at improving the execution of such tree workflows, by theoretical studies. Formally, the purpose is to investigate the problem of scheduling a tree-shaped task graph of malleable tasks.

A malleable task is a task that can be parallelized (the more processors we allocate to it, the faster it will be completed) and the share of processors allocated to it can vary over time. As each node of the tree represents a parallelizable task, the problem consists not only in computing an order of execution between the various tasks of the graph, but also in computing the share of processors that is allocated to each task.

The computing platform studied is composed, unless otherwise specified, of homogeneous processors, and no communication or memory issues between different processors are treated. In other words, it can be seen as a processing power that can be divided between several tasks in any way. Moreover, we allow the allocation of real non-negative shares of processors (e.g., 3.5 processors) to each task. In Section 5, we will add memory constraints, assuming that there exists a unique shared memory capacity that cannot be exceeded.

As said above, the main motivation of this problem comes from scientific computing, and more precisely from tree workflows that arise during LU or Cholesky factorization of sparse matrices. An example of such a tree workflow is the elimination tree, which represents the computational dependencies and storage requirements in those factorizations, see [5] for details. Authorising non-integer processor allocation is motivated by actual runtime techniques, because time-sharing can indeed be used to process several tasks on one processor [1, 9].

One difficulty resides in the fact that the parallelization is not ideal and thus if too much processors are given to a task, losses of work and then of makespan will occur. The speedup of a task is a factor represented by a function f depending on a number of processors p. It is defined as the time necessary to complete the task with p processors divided by the time to complete it with 1 processor. In the ideal case, $f(p)$ would be equal to p, but it is actually smaller because of imperfect parallelizations (communication costs or inner scheduling issues). We assume in this work that the speedup function is the same for all tasks and its expression is known. Unless otherwise specified, we will assume $f(p) = p^\alpha$, with $\alpha \in]0,1[$. This model has been inferred from actual behaviors of matrix operations when $p > 1$ [9]. We do not study the general case of an unknown speedup function.

Both extreme cases of $\alpha \in \{0,1\}$ are not studied neither here. If $\alpha = 1$, this is the perfect case where no losses occur in parallelization and then any schedule that uses all the processors at any time is makespan-minimizing: it suffices to schedule a node after its children. On the other hand, if $\alpha = 0$, no parallelization is beneficial, and it is then natural to impose that at least one processor must be allocated to execute a task. This comes back to the sequential case, as studied in [7].

The remainder of this report is organized as follows. We first deal in Section 4 with the problem of computing makespan-minimizing schedules under two models of the speedup function. The first one is the one described above, and the second one is a modification of this model when $p < 1$ that is closer to reality but more difficult to study. Then, we present some complexity results for the problem combining makespan and memory minimization, assuming the speedup is equal to p^α.

2 Related work

The first problem that we study is to compute the makespan-minimizing schedule of a tree of malleable tasks with the same given speedup function. The concept of malleable tasks is a classical formalism to model parallel computations, see a survey in [2]. A similar problem problem with arbitrary speedup functions has already been studied in [6], but only dealing with integer shares of processors. In this context, the authors achieve to design a polynomial 2.6-approximation for series-parallel graphs and 5.2 for arbitrary precedence constraints. Concerning non-integer shares, this problem has been examined by Prasanna and Musicus for series-parallel graphs in [9, 8]. Their work uses optimal control theory to derive general theorems for any strictly increasing speedup function. For the particular case of the speedup equal to p^α, they proved properties characterizing exactly the unique optimal schedule, and allowing to compute it efficiently. In this report, we prove the same properties on this latter case but using only pure-scheduling arguments. Moreover, we study afterwards an other model of the speedup function that is more realistic.

The second problem studied, that deals with both makespan and memory minimization, has already been studied in [7], but without allowing task parallelization. For a complete review of related work on this subject, we refer the interested reader to the related work section of [7]. We prove here similar results concerning NP-completeness and inapproximations for malleable tasks in a more constraint model close to the pebble game model used in [7]. These results are then, by extension, also valid for the general model.

3 Model and notations

The most elementary structure used in this report is a malleable task, as presented is Section 1. Concerning notations, such a task will be indexed by T, I is a set of indices such that the set of tasks is $\{T_i \mid i \in I\}$ and L_i is the length of task T_i. The length of a task is the amount of work that is needed to complete this task.

The tasks are structured in a precedence rooted in-tree G, which means that the processing of a task cannot begin before all its children are completed. Therefore, at the outset, only leaves can be processed until an internal node has all its children completed, and at the end, only the root is processed. The set of children of T_i is denoted by $Children(T_i)$ and its parent by $parent(T_i)$.

The total amount of processors available at time t is defined by the step function $p(t) \in \left(\mathbb{R}^+\right)^{\mathbb{R}^+}$, called the processor profile.

In order to define the time needed to complete a task knowing how many processors are allocated to it, the speedup function has to be defined. It depends on a non-negative real number p, which is the share of processors allocated to a task, is denoted by $f(p)$ and, unless otherwise specified, will be equal to p^α with $\alpha \in]0,1[$ being a fixed parameter.

A solution schedule $\mathscr{S}$ is defined by a time limit τ and a set of non-negative piecewise continuous functions $\{p_i(t) \mid i \in I\}$ defined on $[0,\tau]$. We can suppose that τ is tight, i.e., $\exists i \in I \mid p_i(\tau) > 0$. The function $p_i(t)$ represents the share allocated to task i and because the tasks are malleable, it can vary over time. Its signification is that 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 it has received since the outset is equal to its length L_i. We define $w_i(t)$ as the ratio of work of the task T_i that is done during the time interval $[0,t]$: $w_i(t) = \int_0^t p_i(x)^\alpha \, \mathrm{d}x / L_i$. The ratio of processors given to a task is $p_i(t)/p(t)$. This quantity is relevant as it will be proved constant under certain conditions. We call a *clean interval* with regard to $\mathscr{S}$ an interval during which no task is completed.

A schedule is a valid solution if and only if it does not use more processors than available, completes all the tasks and respects the precedence constraints. More formally it is valid if and only if it respects the following constraints:

- $\forall t \in [0,\tau], \sum_{i \in I} p_i(t) \leq p(t)$
- $\forall i \in I, w_i(\tau) = 1$
- For all $i \in I$ and $t \in [0,\tau]$, if $p_i(t) > 0$ then, for all $j \in I$ such that T_j is a child of T_i in G, $w_j(t) = 1$

The makespan of a graph G following $\mathscr{S}$ is τ, and the optimal makespan of G is the minimum makespan among every valid schedule.

In the first section, we will actually not study rooted trees but a more general structure that is series-parallel graphs (or SP graphs). A SP graph is recursively

defined by being a single task, a series composition of two SP graphs, or a parallel composition of two SP graphs. The two subgraphs forming a parallel composition are called branches. Series composition are ordered so that it is clear which subgraph should be executed first. Then, we can naturally define the parents and children of nodes as direct successors and predecessors. Similarly, if two nodes have the same set of parents, they are called siblings. A tree can be easily transformed into an SP graph by joining the leaves according to its structure (see Figure 1), 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 the series composition. Then, the SP graph of Figure 1 can be represented as $\Big(\Big(((4 \parallel 5) \parallel 6)\,;2\Big) \parallel 3\Big)\,;\,1\Big)$. Therefore, trees can be considered as a subset of SP graphs, so we will study in Section 4 SP graphs as they are more general and more convenient for this purpose, but the results will stay valid for trees, which is the main objective of this study.

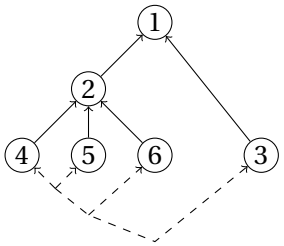


Figure 1: Example of a tree completed in an SP graph.

4 Makespan-minimizing schedules at a fixed speedup

In this section, we do not focus on memory consumption but only on makespan minimization. Therefore, *optimal* should be understood here as *minimizing the makespan*.

As explained above, the speedup is fixed in this section, which means that we design algorithms to a specific speedup, and we do not study the scheduling problem for generic speedup functions, as it was done in [8]. We will first study the model where $f(p) = p^\alpha$, as motivated in [8], before designing a more realistic model for the special case where $p < 1$.

4.1 The speedup is equal to $f(p) = p^\alpha$

The purpose of this section 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.

Moreover, we prove that in an optimal schedule, all siblings terminate at the same time. In addition, the ratio of processors allocated to each task T_i, defined

by $r_i(t) = p_i(t)/p(t)$, is constant and equal to r_i from the moment at which T_i is initiated to the moment at which it is terminated. Similarly, for each branch of a parallel composition, the total ratio of processors allocated to the set of tasks composing this branch is constant.

These properties imply that the optimal schedule is unique and obeys to a *flow conservation* property: on a tree, ratios of processors are given to the leaves at the beginning of the schedule. Then, all the children of a node T_i terminate at the same time, and its ratio r_i becomes the sum of its children ratios.

In [8], the authors used optimal control theory to achieve a similar result when $p(t)$ is constant. We generalize here this result when $p(t)$ is a step function and give a proof using pure-scheduling arguments. This generalization is motivated by modifications of the processing power available.

We first need to define the length $\mathscr{L}_G$ 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 theorem.

Definition 1. *We recursively define the length $\mathscr{L}_G$ depending on a SP graph G:*

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

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

Proof. If a schedule does not use some processors during a certain time interval Δ, it is possible to increase the work performed during Δ for all the tasks being executed during Δ by allocating these processors equitably among those tasks. Then, the work performed during Δ is achieved in a smaller interval because the speedup is a strictly increasing function. It now suffices to remove the processors possibly allocated in surplus (if too much work is done for certain tasks) and move backward the later part of the schedule to reduce the makespan. □

Lemma 2. *We have to process n tasks in parallel with a constant number of processors p. A schedule that does not allocate a constant number of processors per task on clean intervals is not optimal.*

Proof. By contradiction, we consider an optimal schedule $\mathscr{P}$ with makespan M, and we suppose that one task j does not have a constant number of processors allocated to it on a clean interval $\Delta = [t_1, t_2]$. By definition of a clean interval, no task completes during Δ. By abuse of notations, we will also use Δ to represent the length of this interval.

From now on, the index j will refer to this particular task and the index i will refer to any task. Let $p_i(t)$ denote the share of processors allocated to task T_i at time $t \in \Delta$.

The purpose is to build a valid schedule $\mathscr{R}$ with a makespan smaller than M. To achieve it, we will define three intermediate and not necessarily valid

schedules $\mathscr{Q}_{\{1,2,3\}}$, which will nevertheless respect the resource constraint (no more than $p(t)$ processors are used at time t). These schedules will be equal to $\mathscr{P}$ except on Δ where successive modifications will eventually allow to do the same work as $\mathscr{P}$ before t_2, and then lead to the schedule $\mathscr{R}$.

The constant share of processors allocated to task T_i on Δ in $\mathscr{Q}_1$ is defined by $q_i = \frac{1}{\Delta}\int_\Delta p_i(t)dt$. For all t, we have $\sum_{i\in I} p_i(t) = p$. We get $\sum_{i\in I} q_i = p$. So $\mathscr{Q}_1$ respects the resource constraint.

Let $W_i^\Delta(\mathscr{P})$ (resp. $W_i^\Delta(\mathscr{Q}_1)$) denote the work on T_i during Δ under schedule $\mathscr{P}$ (resp. $\mathscr{Q}_1$).

We have

$$W_i^\Delta(\mathscr{P}) = \int_\Delta p_i(t)^\alpha dt = \Delta \int_{[0,1]} p_i(t\Delta)^\alpha dt$$

$$\text{and} \qquad W_i^\Delta(\mathscr{Q}_1) = \int_\Delta \left(\frac{1}{\Delta}\int_\Delta p_i(t)dt\right)^\alpha dx = \Delta\left(\int_{[0,1]} p_i(t\Delta)dt\right)^\alpha$$

As $\alpha < 1$, the function $x \mapsto x^\alpha$ is strictly concave and then, by Jensen inequality, $W_i^\Delta(\mathscr{P}) \leq W_i^\Delta(\mathscr{Q}_1)$. Moreover, again by Jensen inequality, as $p_j(t)$ is not constant, we have $W_j^\Delta(\mathscr{P}) < W_j^\Delta(\mathscr{Q}_1)$.

We will now partition a part the surplus of processors allocated to task j (as $W_j^\Delta(\mathscr{P}) < W_j^\Delta(\mathscr{Q}_1)$) to the other tasks so that each one completes its work before t_2. Then, we will stop the schedule just before t_2 and so perform at least as much work in a smaller makespan.

As $W_j^\Delta(\mathscr{P}) < W_j^\Delta(\mathscr{Q}_1)$, there exists $\varepsilon > 0$ such that with a constant share of processors equal to $q_j - (n-1)\varepsilon$ (with $n = |I|$) allocated to T_j during Δ, the work on T_j is still strictly larger than the work on T_j under $\mathscr{P}$. We create the schedule $\mathscr{Q}_2$ based on $\mathscr{Q}_1$: the modification is that on Δ, task j is allocated a share of $q_j-(n-1)\varepsilon$ processors, and any task $i \neq j$ is allocated a share of $q_i+\varepsilon$ processors. This still respects the resource constraint as the sum of the shares has not been modified.

Then, the work of each task during Δ is strictly larger under $\mathscr{Q}_2$ than under $\mathscr{P}$.

Therefore, there exists $t_2' < t_2$ such that for all i, $W_i^{[t_1,t_2]}(\mathscr{P}) < W_i^{[t_1,t_2']}(\mathscr{Q}_2)$. We note $d = t_2 - t_2'$ and we create another schedule $\mathscr{Q}_3$ from $\mathscr{Q}_2$. The modification is that the share of processors allocated to task i drops to 0 at the time $t^i < t_2'$ where $W_i^{[t_1,t_2]}(\mathscr{P}) = W_i^{[t_1,t^i]}(\mathscr{Q}_2)$.

We construct a last schedule $\mathscr{R}$ defined on $[0, M-d]$. $\mathscr{R}$ is equal to $\mathscr{P}$ on $[0,t_1]$ and to $\mathscr{Q}_3$ on $[t_1,t_2']$. On $[t_2', M-d]$, $\mathscr{R}(t)$ is equal to $\mathscr{P}(t+d)$. For each task, the work performed with $\mathscr{R}$ is equal to the work performed with $\mathscr{P}$. So $\mathscr{R}$ is a valid schedule and completes every task with a smaller makespan. Therefore, $\mathscr{P}$ is not optimal. □

Lemma 3. *Let G be the parallel-composition of tasks T_1 and T_2. Let $r_1(t) = p_1(t)/p(t)$ (resp. $r_2(t)$) be the ratio of processors allocated to T_1 (resp. T_2) in a*

schedule. If $p(t)$ is a step function, in every optimal schedule until the graph is terminated, $r_1(t)$ is constant and equal to $\pi_1 = 1/\left(1+(L_2/L_1)^{1/\alpha}\right) = L_1^{1/\alpha} \Big/ \mathscr{L}_{1\|2}^{1/\alpha}$.

Proof. First, we prove that $r_1(t)$ is constant on any optimal schedule. Therefore, as by Lemma 1 we have $r_2(t) = 1 - r_1(t)$, $r_2(t)$ will also be proved constant. This results implies in particular that both tasks terminate simultaneously as the ratios never get null until the graph is terminated.

We consider an optimal schedule $\mathscr{S}$, 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 the graph is not terminated. By abuse of notations, we will also use A and B to represent the lengths of these intervals. By Lemma 2, the ratio of processors $r_1(t)$ allocated to T_1 in $\mathscr{S}$ has a constant value r_1^A on A and r_1^B on B (these values can potentially be 0 or 1 if one task is terminated). We want to prove that $r_1^A = r_1^B$. Suppose by contradiction that $r_1^A \neq r_1^B$. We can assume without loss of generality that $r_1^A < r_1^B$.

We want to prove that $\mathscr{S}$ is not optimal, and so that we can do the same work than $\mathscr{S}$ on $A \cup B$ in a smaller makespan.

We can assume without loss of generality that $Ap^\alpha = Bq^\alpha$ (otherwise, we can truncate one interval to decrease A or B). We set $r_1 = \left(r_1^A + r_1^B\right)/2$ and we define the schedule $\mathscr{S}'$ equal to $\mathscr{S}$ except on $A \cup B$ where the rate allocated to T_1 is r_1, see Figure 2.

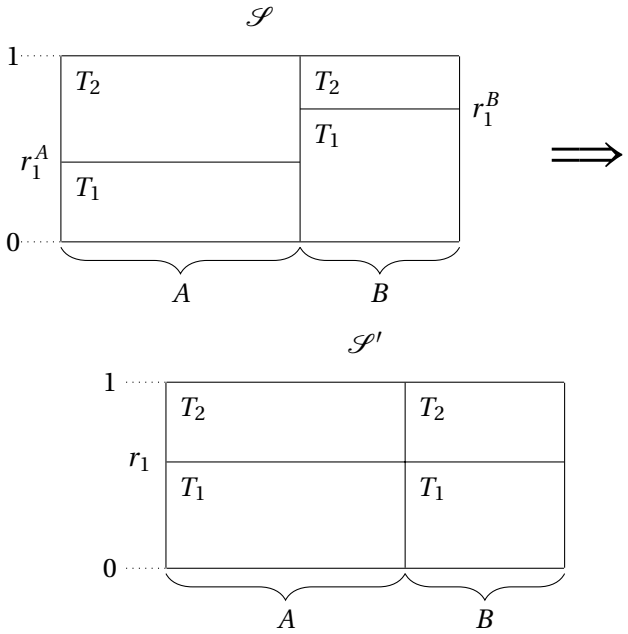


Figure 2: Schedules $\mathscr{S}$ and $\mathscr{S}'$ on $A \cup B$.
The abscissae represent the time and the ordinates the ratio of processing power.

The work of task T_1 under $\mathscr{S}$ and under $\mathscr{S}'$ during $A \cup B$ are

$$W_1 = Ap^\alpha \left(r_1^A\right)^\alpha + Bq^\alpha \left(r_1^B\right)^\alpha$$

$$W_1' = r_1^\alpha \left(Ap^\alpha + Bq^\alpha\right)$$

By concavity, we have:

$$\begin{aligned}
& \frac{\left(r_1^B\right)^\alpha - (r_1)^\alpha}{r_1^B - r_1} < \frac{(r_1)^\alpha - \left(r_1^A\right)^\alpha}{r_1 - r_1^A} \\
\Longrightarrow \quad & Bq^\alpha \left(\left(r_1^B\right)^\alpha - (r_1)^\alpha\right) < Ap^\alpha \left((r_1)^\alpha - \left(r_1^A\right)^\alpha\right) \\
\Longrightarrow \quad & Ap^\alpha \left(r_1^A\right)^\alpha + Bq^\alpha \left(r_1^B\right)^\alpha < r_1^\alpha \left(Ap^\alpha + Bq^\alpha\right) \\
\Longrightarrow \quad & W_1 < W_1'
\end{aligned}$$

We have the same properties for T_2 as the hypotheses are symmetric between T_1 and T_2.

Therefore, $\mathscr{S}'$ does strictly more work for both tasks during $A \cup B$, and so can be modified as in Lemma 2 to do the same work in a smaller makespan. Then, $\mathscr{S}$ is not optimal.

Now, we want to prove that in an optimal schedule $\mathscr{S}$, $r_1(t)$ must be equal to π_1 and hence that the optimal schedule is unique. As $p(t)$ is a step function, we define the sequences $(A_i)_{i>0}$ and $\left(p_i\right)_{i>0}$ such that A_i is the duration of the i-th step of the function $p(t)$ and $p(t) = p_i > 0$ on A_i. Therefore, the sum of A_i's is the makespan of $\mathscr{S}$.

Then, as $\mathscr{S}$ completes both T_1 and T_2 with constant rates, if we note $V = \sum_i A_i p_i^\alpha$ and r_1 the value of $r_1(t)$, we have:

$$L_1 = \sum_i A_i r_1^\alpha p_i^\alpha = r_1^\alpha V \qquad \text{and} \qquad L_2 = \sum_i A_i (1-r_1)^\alpha p_i^\alpha = (1-r_1)^\alpha V$$

Then,

$$L_2 = (1-r_1)^\alpha \frac{L_1}{r_1^\alpha} \qquad \text{thus} \qquad r_1 = \frac{1}{1 + \left(\frac{L_2}{L_1}\right)^{1/\alpha}} = \pi_1$$

So the lemma is proved. □

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

Proof. We characterize $p(t)$ by the sequences $(A_i)_{i>0}$ and $(p_i)_{i>0}$ as in the proof of Lemma 3. Let Δ be the domain of definition of $\mathscr{S}$, so that $\Delta = [0, \tau]$. As defined in Section 3, we define the functions $w_i(t)$ representing the ratio of work done during $[0, t]$: $w_i(t) = \int_0^t p_i(x)^\alpha \, \mathrm{d}x / L_i$, such that $w_1(0) = 0$ and $w_1(\tau) = 1$. For

$t \in \Delta$, let $k(t)$ be the index such that $p(t)$ is in its $k(t)$-th step, and so $p(t) = p_{k(t)}$. And let $\bar{t}$ be the difference between t and the starting time of the $k(t)$-th step. By Lemma 3, we know that the ratio of processors allocated to T_1 is constant over Δ and equal to:

$$r_1 = \frac{L_1^{1/\alpha}}{L_1^{1/\alpha} + L_2^{1/\alpha}} = \left(L_1 / \mathscr{L}_{1\|2}\right)^{1/\alpha}$$

Then, we have:

$$w_1(t) = \frac{\bar{t}\left(p_{k(t)} r_1\right)^\alpha + \sum_{i<k(t)} A_i \left(p_i r_1\right)^\alpha}{L_1} = \frac{\bar{t}\, p_{k(t)}^\alpha + \sum_{i<k(t)} A_i p_i^\alpha}{\mathscr{L}_{1\|2}}$$

Similarly, for T_2, we have:

$$w_2(t) = \frac{\bar{t}\left(p_{k(t)}(1-r_1)\right)^\alpha + \sum_{i<k(t)} A_i \left(p_i(1-r_1)\right)^\alpha}{L_2} = \frac{\bar{t}\, p_{k(t)}^\alpha + \sum_{i<k(t)} A_i p_i^\alpha}{\mathscr{L}_{1\|2}}$$

We define $w(t)$ as the ratio of work that is done for the equivalent task T_G of length $\mathscr{L}_{1\|2}$ under the processor profile $p(t)$, until the task is terminated.

We have:

$$w(t) = \frac{t\, p_{k(t)}^\alpha + \sum_{i<k(t)} A_i p_i^\alpha}{\mathscr{L}_{1\|2}} = w_1(t) = w_2(t)$$

The three ratios are identical, so they all reach 1 at time τ. Then, G and T_G have the same optimal makespan under any step-function $p(t)$. □

Theorem 1. *For every graph G, if $p(t)$ is a step function, G has the same optimal makespan than the equivalent task T_G of length $\mathscr{L}_G$. 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, it is implicitly the optimal makespan.

First, we have to 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 prove the result by induction.

- G is a single task. The result is immediate, as by Lemma 1, all the processors have to be used.
- G is the series composition of G_1 and G_2. By induction, G_1 (resp. G_2) has the same makespan than task T_1 (resp. T_2) of length $\mathscr{L}_1$ (resp. $\mathscr{L}_2$) under any processor profile. Therefore, the makespan of the series composition of T_1 and T_2 is equal to the makespan of G. Then, it is equal to the makespan of the task of length $\mathscr{L}_G = \mathscr{L}_{1;2} = \mathscr{L}_1 + \mathscr{L}_2$.

By induction, the unique optimal schedules of G_1 and G_2 under $p(t)$ processors can be computed, so there is a unique optimal schedule of G under $p(t)$ processors: the concatenation of these two schedules.

- G is the parallel composition of G_1 and G_2. By induction, G_1 (resp. G_2) has the same makespan than task T_1 (resp. T_2) of length $\mathscr{L}_1$ (resp. $\mathscr{L}_2$) under any processor profile.

  Consider an optimal schedule $\mathscr{S}$ of G and let $p_1(t)$ be the processor profile allocated to G_1. Let $\tilde{S}$ be the schedule of $(T_1 \parallel T_2)$ that allocates $p_1(t)$ processors to T_1. $\tilde{S}$ is optimal and gives the same makespan as $\mathscr{S}$ for G because T_1 and G_1 (resp. T_2 and G_2) have the same makespan under any processor profile. Then, by Lemma 4, $\tilde{S}$ (so $\mathscr{S}$) gives the makespan equal to the optimal makespan of $\mathscr{L}_{1\parallel 2} = \mathscr{L}_G$.

  Moreover, by Lemma 3 applied on $(T_1 \parallel T_2)$, 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, in order to determine the optimal schedule of a graph G, it suffices to inductively build the equivalent task of each parallel branch and use the rate π_1 described in the proof to compute the processing power that should be given to each branch.

This makespan-minimizing problem is then solved as the optimal schedule can be computed efficiently. A critic, however, can be made on the validity of the speedup value when $p < 1$. Indeed, the speedup should be linear in this domain as no parallelization issue occurs and so the model overestimates it. Moreover, because of the strict concavity of the function, more work is done by a processor when it is split between several tasks, which is unrealistic. This motivated the following modification of the model, for applications where less than one processor can be used on a task.

4.2 The speedup is equal to $f(p) = p$ when $p < 1$ and $f(p) = p^\alpha$ otherwise

As motivated above, we modify in this section the speedup function when $p < 1$, so that $f(p) = p$ when $p < 1$ and $f(p) = p^\alpha$ otherwise. See Figure 3 for an illustration. We only study here the case where $p(t)$ is constant and equal to p.

This modification complicates the model as optimal schedules do no longer necessarily obey the structure depicted in the previous section, as proved in Property 2. Therefore, a heuristic has been designed to approach the optimal makespan, but no performance guarantee has been proved yet.

In order to clarify the results, we define two classes of schedules, which can also be called processor allocations, as they are completely defined by the share

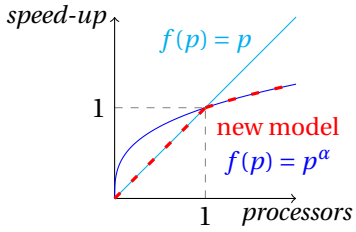


Figure 3: Speed-up in function of the processing power in different models

of processors allocated to each task and the precedence constraints, and not by temporal parameters. First, the *PM allocation* (for **P**rasanna-**M**usicus allocation) of a graph is the allocation that associates to each task the share computed by the formulas of the previous section. Then, a *PFC allocation* (**P**rocessor **F**low **C**onservation) is an allocation that associates a constant ratio of processors to each branch at every parallel node. By definition, PM allocations are necessarily PFC allocations. The motivation of the latter class is to preserve a simple structure while allowing other shares of processors than the ones defined in Subsection 4.1. Indeed, they seem general enough to offer better solutions than the PM allocation, and behaved well in our simulations. We will focus here on schedules belonging to this class, as they are easier to manipulate.

The first result, which is proved in Appendix A, shows that:

Property 1. *There exists a unique PFC allocation of minimum makespan, the one in which every siblings terminate simultaneously.*

However, Property 2 states that this schedule is not always optimal among all possible schedules.

Property 2. *The best PFC allocation is not always the optimal schedule. This statement is still valid on pseudo-trees and for moldable tasks (tasks whose share of processor cannot vary until termination).*

Proof. We present an example in which no PFC schedule is optimal, which proves the proposition. The goal is to parallelize the tree-shaped graph G of Figure 4 with $p = 4$ processors. One possible allocation is to allocate one processor to each of the four tasks of length 1, and then two processors to both remaining tasks. The makespan is then $M = 2$, and the schedule is not PFC.

In an optimal PFC schedule, a certain share $0 < x < 4$ is dedicated to subgraph G_2. We have $x < 3$ because with $x = 3$, G_2 is completed before G_1. The makespan of G_1 is $M_1(x) = \frac{1+2^\alpha}{(4-x)^\alpha}$ and is increasing with x.

Each of the $b_{1,i}$ tasks must receive equal shares of $\frac{x}{3} < 1$ processors so the makespan of G_2 is $M_2(x) = \frac{3}{x} + \left(\frac{2}{x}\right)^\alpha$ and is decreasing with x. Then, suppose that such a schedule is makespan-optimal. There exists x_e such that $M_1(x_e) = M_2(x_e) < M$. Then, for the value x_1 such that $M_1(x_1) = M$, we must

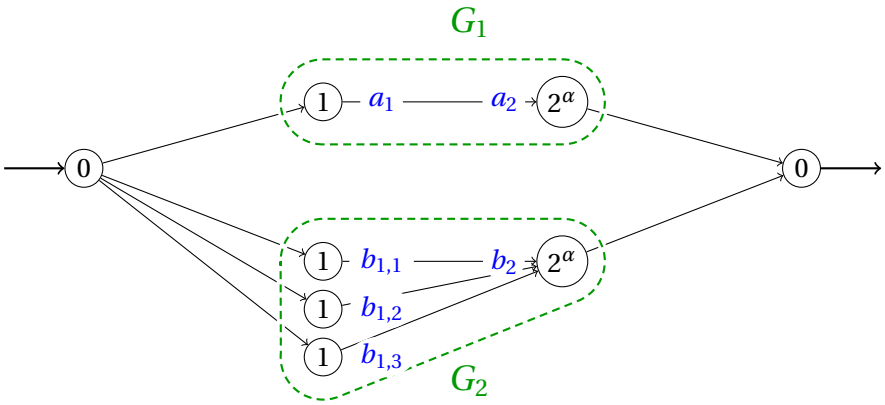


Figure 4: Series-parallel graph of the counter example.

have $M_2(x_1) < M$, as $x_1 > x_e$ by the increasing of M_1 and M_2 is decreasing. But in this case, we have:

$$x_1 = 4 - \left(\frac{1+2^\alpha}{2}\right)^{1/\alpha}$$

Then, we numerically verified that $M_2 > M$ for any $0 < \alpha < 1$. So no PFC schedule is optimal. □

The following result shows that PM allocations are not relevant approximations for the makespan, and so motivates the choice to study PFC allocations, which are more general.

Property 3. *The PM allocation is not a constant ratio approximation for the makespan (proof: Appendix C).*

Indeed, the PFC-optimal allocation seems more promising. For indication, the worst ratio observed between the makespan of the PFC-optimal allocation and another allocation is around 1.09 (see Appendix C). Therefore, in spite of the lack of established approximation ratio, this schedule seems to be a relevant approximation.

However, this schedule is again difficult to compute. We can nevertheless characterize it by a quantity $\Delta_{\mathscr{P}}$ associated to a graph G and a PFC allocation $\mathscr{P}$ which represents the largest difference of makespan between two parallel branches and is inductively defined on G with respects to $\mathscr{P}$:

- $\Delta_{\mathscr{P}}(T) = 0$
- $\Delta_{\mathscr{P}}(G_1 ; G_2) = \max(\Delta_{\mathscr{P}}(G_1), \Delta_{\mathscr{P}}(G_2))$
- $\Delta_{\mathscr{P}}(G_1 \parallel G_2) = \max(\Delta_{\mathscr{P}}(G_1), \Delta_{\mathscr{P}}(G_2), |M_{\mathscr{P}}(G_1) - M_{\mathscr{P}}(G_2)|)$ where $M_{\mathscr{P}}(G_i)$ is the makespan of G_i under schedule $\mathscr{P}$.

The characterization is the following.

Property 4. *A PFC allocation $\mathscr{P}$ is optimal if and only if $\Delta_{\mathscr{P}} = 0$ (Proof: Appendix A).*

A way to compute this schedule is to start from the PM allocation, which underestimates the makespan of tasks to whom less than 1 processor has been allocated. Then, the idea is to rebalance the allocation of the deepest parallel connections whose branches have different makespans. However, this strategy needs to predict the makespan of a subgraph for a given processor power, in order to balance large branches, which cannot be done efficiently.

Therefore, we design the heuristic depicted in Algorithm 1. The idea is, instead of rebalancing the allocation, to artificially modify the length of underestimated tasks in order to counterbalance the errors made by the PM allocation.

If L_i is the length of an underestimated task and $p_i < 1$ the share allocated to it, the updated length $\bar{L}$ should verify $\bar{L}/p_i^\alpha = L_i/p_i$ so that both makespans are equal: the one expected by the PM allocation, $\bar{L}/p_i^\alpha$ and the real one, L_i/p_i. Therefore, we have $\bar{L} = L_i\, p_i^{\alpha-1}$. However, in the updated allocation, p_i will likely take a different value, either larger or smaller. That's why the heuristic needs to iterate this step, until convergence.

The principle is to permanently mark at each step tasks with $p_i < 1$, and update their length before computing the new PM allocation. If some p_i become greater than 1, the length should then be decreased as the PM allocation now overestimates this makespan, hence the *if-then* statement.

Input: a graph G with the length L_i^0 of each task i, the parameter α and a processing power p
Output: A PFC processor allocation of G

compute the PM allocation into the p_i^0
$j \leftarrow 0$
while *not converging* **do**
  $j \leftarrow j+1$
  mark all tasks with $p_i^{j-1} < 1$
  for *all marked tasks* i **do**
    if $p_i^{j-1} < 1$ **then**
      $L_i^j \leftarrow L_i^0 \cdot \left(p_i^{j-1}\right)^{\alpha-1}$
    else
      $L_i^j \leftarrow 1/2\left(L_i^0 + L_i^{j-1}\right)$
  compute the PM allocation and save it in the p_i^j's

Algorithm 1: Heuristic to approach the optimal PFC allocation

The heuristic has been implemented and tested on a large variety of SP graphs, either designed manually or generated randomly among SP graphs including a thousand tasks. This allows to have a good intuition of the relevance of this heuristic. However, it is does not prove its correctness as the tested graphs

do not represent a significant part of this family and so critical cases may not have been scanned. Nevertheless, on every studied example, the heuristic converged towards the optimal PFC allocation whenever $\alpha > 1/2$. The convergence can indeed be verified by computing the value $\Delta_{\mathscr{P}}$, which tends towards 0 if and only if the heuristic converges towards the expected schedule.

A conjecture derived from these observations is the following.

Conjecture 1. *For $\alpha > 1/2$, during the execution of the algorithm, both sequences $\left(\Delta_{\mathscr{P}}^{2j}\right)_j$ and $\left(\Delta_{\mathscr{P}}^{2j+1}\right)_j$ decrease, and so have a limit. These limits are both equal to 0 and thus the algorithm converges to the optimal PFC schedule.*

The difficulty for proving this conjecture is that the whole allocation vary at each step, and the property links step j with step $j+2$, independently from step $j+1$. It has been proven for the particular case the parallel composition of two tasks (see Appendix B), but the proof seems difficult to generalize.

For $\alpha < 1/2$, the heuristic does not converge on various examples, which can be explained by a too large error made by the PM allocation. However, in practice, we should have $\alpha > 1/2$ so this is not a critical issue.

5 Makespan and memory minimizing schedules

In this section, we take into account the memory needed to execute a task tree. If the memory bound is not a crucial constraint on standard computers, it can become very restraining on platforms housing many processors. Indeed, some algorithms of sparse matrix factorizations, such as multifrontal methods, have high memory costs. Therefore, we need not only to minimize the makespan, but also to prevent an execution from exceeding a memory bound.

As explained in Section 2, we show here negative complexity results in a model similar to the one studied in [7] without task parallelization, but we do not achieve to use a fully homogeneous model as in that paper. Therefore, the theorems proved here are slightly weaker than their analogues from [7], because more flexibility in the input is necessary to achieve the complexity results. This difference is due to the freedom induced by the task parallelization which allows less control on optimal schedule behaviors with the same constraints.

5.1 Description of the model and preliminary lemma

In this section, the objective is to schedule a tree G under a fixed processor profile $p(t) = p$. Each node T_i has a length L_i and an output file of size f_i that is used as input by its parent if T_i is not the tree root. Actually, nodes can also have execution files n_i but these can be supposed null, as explained below. The speedup is equal to $f(p) = p^\alpha$, even when $p < 1$.

We have here an additional constraint when compared to the model without memory. During the execution of T_i, the memory must contain its execu-

tion, output and its input files (the output files of its children), which leads to a memory occupation for task T_i of:

$$Mem_i^{\text{ex}} = \left(\sum_{j \in Children(T_i)} f_j \right) + n_i + f_i$$

Moreover, the output file of a node must be kept in memory: it cannot be released until the termination of its parent. With this definition, n_i can indeed be set to 0 and simulated by an additional child of null length and output file equal to n_i. Therefore, we now assume that all n_i's are equal to 0.

The objective is to minimize the makespan, which has already been defined, and the memory peak Mem^{peak}. This latter quantity is the maximum amount of memory needed among all time steps. The memory needed at a time t, Mem_t, is equal to the sum of all the Mem_i^{ex} of the tasks T_i that are being executed at time t, plus all the output files that must remain in memory because a node has been processed but the processing parent is not initiated yet. More formally, we have:

$$Mem_t = \left(\sum_{i \mid 0<w_i(t)<1} Mem_i^{\text{ex}} \right) + \left(\sum_{i \,\middle|\, \left(w_{parent(T_i)}(t)=0 \text{ and } w_i(t)=1\right)} f_i \right)$$

In order to work with schedules obeying to such memory constraints, we first need the following general lemma which gives us a characterisation of makespan-optimal schedules when a bound on the maximum parallelization is imposed.

Lemma 5. *For all $x, n \in \mathbb{N}^{\times}$ and for all $p > n$, if we have xn independant tasks of length 1 to process in parallel with xp processors and we cannot execute more than x nodes at any time, a schedule is makespan-optimal if and only if it processes at any time x nodes in parallel using equal maximal shares of processors.*

Proof. Let $\mathcal{S}$ be a schedule with makespan M. We know by Lemma 2 that the shares follow step functions: on clean intervals, optimal schedules allocate constant ratios of processors to each task. Indeed, the constraint on the maximal number of nodes to process in parallel does not contradict the hypotheses of the lemma. Similarly, we know by Lemma 1 that makespan-optimal schedules use all processors at any time, so we only consider such schedules here. Therefore, when a schedule allocates equal shares to x processors, it allocates p/x processors to each task.

The purpose is to show that either $\mathcal{S}$ always processes x nodes in parallel with equal shares and has the same makespan than any other such schedule, or its makespan is larger than such a schedule. We first show an analogous property on a small interval where all the shares are constant before generalizing the result.

Let A be a time interval in which all the shares are constant. By abuse of notations, we will also use A to represent the length of this interval. Suppose the

distribution of the x involved tasks among the processors is represented by the shares $(a_i)_{1\leq i\leq x}$ (we can potentially have some of them equal to 0). Then, the total work performed during A is $W_A = A\sum_{i=1}^{x} a_i^\alpha$.

Suppose that we have i, j such that $a_i < a_j$. Let $\tilde{a} = \frac{1}{2}(a_i + a_j)$ and $d_a = \frac{1}{2}(a_j - a_i)$. Let $\mathscr{S}'$ be the schedule defined on A that is equal to $\mathscr{S}$ on A except that it associates a share $\tilde{a}$ to both tasks i and j, and let W'_A be the total work performed during A by $\mathscr{S}'$. As the function $t \mapsto t^\alpha$ is strictly concave, we have:

$$\frac{W'_A - W_A}{A} \;=\; 2\tilde{a}^\alpha - a_i^\alpha - a_j^\alpha \;=\; d_a\left(\frac{\tilde{a}^\alpha - a_i^\alpha}{\tilde{a} - a_i} - \frac{a_j^\alpha - \tilde{a}^\alpha}{a_j - \tilde{a}}\right) \;>\; 0$$

Then, $\mathscr{S}'$ has a total work larger than $\mathscr{S}$ during A. Therefore, the unique schedule (up to a permutation of processors) that maximizes the total work performed during A, with the x involved tasks fixed, is the one that allocates the same share to all x tasks.

Let $\mathscr{S}_{\text{eq}}$ be a schedule that finishes every task and allocates equal shares to x tasks at any time. Let M_{eq} be its makespan and M_n be the minimum of M and M_{eq}. We can partition the interval of time $\Delta_n = [0, M_n]$ in time intervals $\{A_i\}$ such that both $\mathscr{S}$ and $\mathscr{S}_{\text{eq}}$ are constant on each A_i.

On A_i, if $\mathscr{S}$ allocates equal shares to the x tasks involved, the total work of $\mathscr{S}$ and $\mathscr{S}_{\text{eq}}$ is equal. Otherwise, we have shown that the total work of $\mathscr{S}_{\text{eq}}$ is larger than the total work of $\mathscr{S}$ on A_i. We then have two cases:

- if for all A_i, $\mathscr{S}$ allocates equal shares to the x tasks involved, then the total works of $\mathscr{S}$ and $\mathscr{S}_{\text{eq}}$ are equal on Δ_n. Therefore, $M = M_{\text{eq}}$.
- otherwise, let A_i be an interval that contradicts the previous statement. Let $W^{eq}_{\Delta_n}$ and W_{Δ_n} be the total work of both schedules during Δ_n. We have

$$W^{eq}_{\Delta_n} - W_{\Delta_n} \;=\; W^{eq}_{A_i} - W_{A_i} + \sum_{j\neq i}(W^{eq}_{A_j} - W_{A_j}) \;\geq\; W^{eq}_{A_i} - W_{A_i} \;>\; 0$$

  Then, $\mathscr{S}$ is not terminated at time M_n so $M_{\text{eq}} < M$ and $\mathscr{S}$ is not makespan-optimal.

To conclude, $\mathscr{S}_{\text{eq}}$ is makespan-optimal so $\mathscr{S}$ is makespan-optimal if and only if it processes at any time x nodes in parallel at equal maximal shares of processors. □

5.2 NP-completeness of the bi-objective problem

We now state the decision problem combining minimisation of both makespan and memory:

Definition 2 (BiObjectiveParallelTreeScheduling)**.** *Given a task tree G provided with memory weights and task durations, p processors, and two bounds B_C and B_{Mem}, is there a schedule of the task tree on the processors whose makespan is not larger than B_C and whose memory peak is not larger than B_{Mem}?*

This problem allows any parameters f_i and L_i. Actually, in order to show the NP-completeness of this problem, we can restrict ourselves to the case where all the output files f_i have the same size, 1, and the lengths L_i either have the same value, 1, or are null. Without task parallelisation, in [7], the difference is that the L_i can be restricted to have the same value, 1.

Theorem 2. *The BiObjectiveParallelTreeScheduling problem is NP-Complete in the following model:* $\forall i\ f_i = 1, n_i = 0, L_i \in \{0,1\}$.

Proof. First, the problem clearly belongs to NP.

Then, we reduce the problem from 3-PARTITION which is unary NP-Complete [4]. Let I_1 be the following instance of 3-PARTITION: let B be an integer and $a_1,\dots,a_{3m}$ be $3m$ integers such that $\sum_i a_i = mB$ and for all i, $B/4 < a_i < B/2$. The problem is to find whether there exists a partition of the a_i's in m subsets $S_1,\dots,S_m$ each of cardinal 3, such that for all k, $\sum_{i\in S_k} a_i = B$.

Consider the instance I_2 illustrated on Figure 5. The tree $\mathcal{T}$ contains a root r with $3m$ children $N_1,\dots,N_{3m}$, each corresponding to a value a_i. Each node N_i has $3m \times a_i$ children T^i_x, which are leaf nodes. The execution time of the leaves and the root are 0 and the execution time of the other nodes are 1. The question is to find a schedule of this tree on $p > 3$ processors, whose memory peak is not larger than $B_{Mem} = 3mB + 3m$ and whose makespan is not larger than $B_C = m(3/p)^\alpha$.

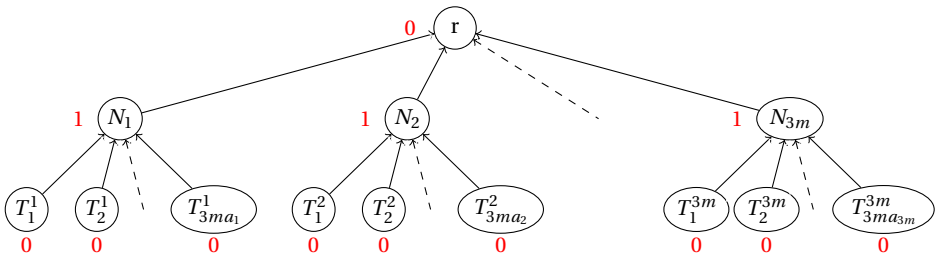


Figure 5: Tree $\mathcal{T}$ used for establishing Theorem 2, with execution times written in red.

Assume first that there exists a solution $S_1,\dots,S_m$ to I_1. We build the schedule $Opt(\{S_n\})$ for I_2, assuming that $S_n = \{a_{i_n}, a_{j_n}, a_{k_n}\}$:

- step 1: process the nodes $T^{i_1}_x, T^{j_1}_y, T^{k_1}_z$ for $1 \le x,y,z \le 3ma_{i_1}$ in time 0 and using $3mB$ units of memory.
- step 2: process the nodes $N_{i_1}, N_{j_1}, N_{k_1}$ in time $(3/p)^\alpha$ and using memory $3mB+3$
- step 2n+1 with $1 \le n \le m-1$: process the nodes $T^{i_n}_x, T^{j_n}_y, T^{k_n}_z$ in time 0 and using $3mB+3n$ units of memory.
- step 2n+2 with $1 \le n \le m-1$: process the nodes $N_{i_1}, N_{j_1}, N_{k_1}$ in time $(3/p)^\alpha$ and using memory $3mB+3(n+1) \le B_{Mem}$
- step 2m+1: process the root node in time 0 and using memory $3m+1$.

Thus, the memory peak is B_{Mem} and the makespan is B_C so I_2 is solvable.

Now, assume there exists a solution to I_2. We will show that I_1 is solvable. Consider a makespan-optimal schedule of I_2 with a memory peak not larger than B_{Mem}.

First, there exists a schedule with the same makespan and a not larger memory peak that processes for any $i \in [1;3m]$ all T_x^i leaves with $x \in [1;3m \times a_i]$ right before N_i. Indeed, the execution file and the execution time of each leaf T_x^i is null so postponing the execution of such a node until the execution of its parent does not modify the makespan and cannot increase the memory peak. We now consider this schedule $\mathscr{S}$.

As for any i, $a_i > B/4$, we can never process in parallel 4 nodes N_i at the same time, because the sum of the corresponding a_i's will not be smaller than $B+1$ and the memory peak would be at least $3m \times (B+1) + 4 > B_{Mem}$. So, at any time, at most 3 of the N_i nodes are processed. We will refer to this memory constraint as CST_{3N_i}.

By Lemma 5, a schedule $Opt(\{S_n\})$ with an arbitrary partition of the a_i's into m triples $S_1, \ldots, S_m$ is makespan-optimal among the schedules that respect CST_{3N_i} (regardless of other memory constraints). Indeed, by Lemma 5, a makespan-optimal schedule of the nodes $N_1 \ldots N_{3m}$ that respects CST_{3N_i} processes the N_i nodes in groups of 3, and has then a makespan equal to B_C, which is the makespan of $Opt(\{S_n\})$. So the makespan of $\mathscr{S}$ is not smaller than B_C and it is furthermore equal to B_C as $\mathscr{S}$ is a solution of I_2. Then, $\mathscr{S}$ is makespan-optimal among the schedules that respect CST_{3N_i} so, by Lemma 5, it processes at any time 3 nodes N_i in parallel using equal shares of processors.

Because of CST_{3N_i}, there cannot be any preemption (i.e., begin the processing of a task, idle it and continue it later) because this would mean at least 4 of the N_i nodes running in parallel (with one being allocated a processor share of 0 during a time interval) as the memory footprint is conserved during the preemption. Therefore, $\mathscr{S}$ executes consecutively groups of three tasks, allocating $p/3$ processors to each one. So there exists a partition $\{S_n\}_{1 \le n \le m}$ such that $\mathscr{S} = Opt(\{S_n\})$. Then, because of memory constraints, for all n, $\sum_{a_i \in S_n} a_i \le B$ and so I_1 is solvable. □

5.3 Inapproximation results

As the bi-objective problem is NP-complete in this restrained model, the next step is to study the existence of approximation algorithms in the same model. Theorem 3 states that no approximation exists with constant factors for both makespan and memory peak, that is, with ratios that are independent of the number p of available processors.

Theorem 3. *There is no algorithm $\mathscr{A}$ that is both a β-approximation for makespan minimization and a γ-approximation for memory peak minimiza-*

tion when scheduling in-tree task graphs in the model: for all i, $f_i = 1, n_i = 0, L_i \in \{0,1\}$.

Proof. Suppose that $\mathscr{A}$ is such an algorithm. Without loss of generality, we suppose that β and γ are integers.

We consider the tree of Figure 6 to be scheduled with $p = \delta n$ processors. The root has n children $N_1 \dots N_n$ which each have δ children $T_1^i \dots T_\delta^i$. Only the T_x^i leaves have a non null execution time, which is equal to 1. The values δ and n will be fixed later.

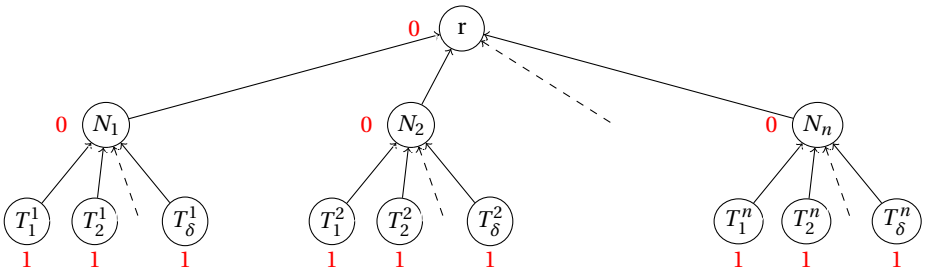


Figure 6: Tree used for establishing Theorem 3, with execution times written in red.

Optimal makespan. We have only δn tasks of non null execution time, so by Lemma 5, the optimal makespan is equal to $Opt_C = (n\delta/p)^\alpha = 1$.

Optimal memory peak. The optimal memory peak is $\delta + n$, by completing the subtrees one after the other.

Contradiction with $\mathscr{A}$. We set $\delta = \gamma n^2$ and we let n be a multiple of $(\gamma + 1)$.

By definition of $\mathscr{A}$, its memory peak is not larger than

$$Mem = \gamma(\delta + n) = \delta(\gamma + n\gamma/\delta) = \delta(\gamma + 1/n) < \delta(\gamma + 1)$$

Then, we cannot process more than $\Delta = \delta(\gamma + 1)$ leaves at any time. We will refer to this constraint as CST_Δ.

The makespan of $\mathscr{A}$ is not smaller than the makespan of a makespan-optimal schedule $\mathscr{B}$ that processes the δn leaves under constraint CST_Δ.

By Lemma 5, as Δ divides δn, $\mathscr{B}$ processes Δ tasks at any time, allocating $p/\Delta = \delta n/\Delta$ processors per task. There are δn tasks in total, so $\delta n/\Delta$ groups of Δ tasks. The makespan of $\mathscr{B}$ is then equal to:

$$M_{\mathscr{B}} = \frac{\delta n}{\Delta}\left(\frac{\Delta}{\delta n}\right)^\alpha = \left(\frac{n}{\gamma+1}\right)^{1-\alpha}$$

Therefore, there exists n, multiple of $(\gamma + 1)$, such that $M_{\mathscr{B}} > \beta$ and so the makespan of $\mathscr{A}$ is larger than βOpt_C which is a contradiction with the definition of $\mathscr{A}$. □

In this theorem, we restrained ourselves to constant ratio approximations. We now give a lower bound in Theorem 4 on approximation ratios that can depend on p.

Theorem 4. *When scheduling in-tree task graphs with p processors, there is no algorithm $\mathscr{A}$ that is both a $\beta(p)$-approximation for makespan minimization and a $\gamma(p)$-approximation for memory peak minimization in the model: for all i, $f_i = 1, n_i = 0, L_i \in \{0,1\}$ with $\beta(p)$ and $\gamma(p)$ verifying:*

$$\gamma(p)\beta(p)^{1-\alpha} \leq \left(\frac{p}{\log p + 1}\right)^{1-\alpha}$$

Proof. Suppose that $\mathscr{A}$ is such an algorithm.

We consider the tree of Figure 7 to be scheduled with $p = 2^{p'}$ processors. The tree is binary and complete over a depth of $n = p'$. There are then $N = 2^n = p$ leaves. Only the leaves have a non null execution time, which is equal to 1.

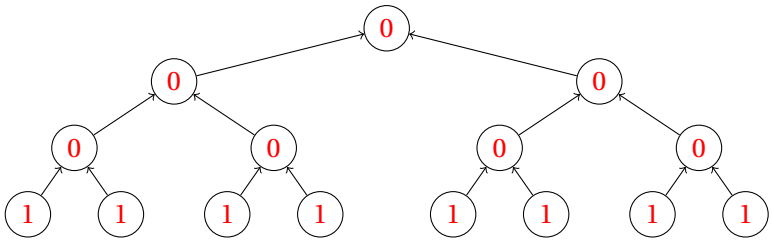


Figure 7: Tree used for establishing Theorem 4 for $n = 3$, with execution times written in red.

Optimal makespan. By Lemma 5, a makespan-optimal schedule executes the N leaves in parallel, with 1 processor per task, and takes a time $Opt_C = 1$.

Optimal memory peak. By completing the tasks following a postorder search, we get a memory peak of $Opt_M = n+1$, which is optimal as any schedule reaches at least this peak during the execution of the last leaf. For indication, an example of a postorder is the order the tasks of a tree G are visited by the recursive algorithm `postorder(G)`: 1) call `postorder` on the left subtree – 2) call `postorder` on the right subtree – 3) `visit` the root.

Contradiction with $\mathscr{A}$. By definition, the memory of $\mathscr{A}$ is not larger than $M = Opt_M\beta(p)$. We can assume that $\beta(p) \leq p/Opt_M$ because as $\gamma(p) \geq 1$, the aimed bound would be verified otherwise.

By Lemma 5, under this memory bound, a makespan-optimal algorithm handling only the leaves executes the leaves by parallelizing groups of $\lfloor Opt_M\beta(p) \rfloor$ tasks. The makespan of the execution of one group is $\left(\frac{\lfloor Opt_M\beta(p) \rfloor}{p}\right)^{\alpha}$ and there are at least $\frac{N}{\lfloor Opt_M\beta(p) \rfloor}$ groups. The overall makespan is then

$$C \geq \frac{N}{\lfloor Opt_M \beta(p) \rfloor} \left(\frac{\lfloor Opt_M \beta(p) \rfloor}{p} \right)^{\alpha} \geq \left(\frac{p}{\lfloor Opt_M \beta(p) \rfloor} \right)^{1-\alpha} \geq \left(\frac{p}{Opt_M \beta(p)} \right)^{1-\alpha}$$

Then, the makespan of $\mathscr{A}$ is larger than C as it must obey stricter memory constraints, because of the memory consumed by non-leaf tasks. Thus,

$$\gamma(p)\beta(p)^{1-\alpha} > \frac{C}{Opt_C} \beta(p)^{1-\alpha} \geq \left(\frac{p}{Opt_M} \right)^{1-\alpha} = \left(\frac{p}{\log p + 1} \right)^{1-\alpha}$$

Hence the bound. □

When disabling task parallelization, in [7], which is in this proof equivalent to state $\alpha = 0$, the bound proved on the factors of approximation was:

$$\gamma(p)\beta(p) > \frac{2p}{\lceil \log p \rceil + 2}$$

With $\alpha = 0$, the bound proved in this report is:

$$\gamma(p)\beta(p) > \frac{p}{\log p + 1}$$

Therefore, this bound is approximately two times smaller than the one of [7] on similar cases, so it is not tight. The gap is explained by the difficulty to find a more precise characterization of makespan-optimal schedules than the one of Lemma 5, whereas without task parallelisation, their structure is easier to study.

6 Conclusion

In this report, we have re-established the results of [8] for minimizing the makespan with a speedup of p^{α} with pure scheduling techniques. This work is relevant as optimal control theory was used in that paper to achieve the same results, which requires too much formalism and artefacts such as approximations of step functions into continuous functions which are actually not necessary to establish in this theorem. Its proof is now decomposed in lemmas which can be proved step by step by pure scheduling techniques and gives an overview on the foundations of the theorem. We have also proposed a refinement of the model, which is more realistic for applications allocating less than one processor to some tasks, but complicates the computation of the optimal schedule. An algorithm has been proposed to find an approximation of this schedule when $\alpha > 1/2$, but if it behaves satisfactorily on every example tested, no formal proof has been obtained.

Then, we presented the first study that combines makespan and memory constraints together with parallelizable tasks. We achieved to reproduce theorems analogous to state-of-the-art results without task parallelization, which state the complexity of the problem.

In short term future work, it remains to prove the convergence and the approximate ratio of the algorithm of Section 4. In the long term, it would be interesting to develop a heuristic allowing a compromise between minimizing the makespan and the memory peak, for example with a guaranteed bound on the memory peak. Finally, an experimental validation of the proposed heuristics would be very desirable, and we have reasonable hope that at least some of them will be tested in the context of the SOLHAR project.

References

[1] O. Beaumont and A. Guermouche. Task scheduling for parallel multifrontal methods. In A.-M. Kermarrec, L. Bouge, and T. Priol, editors, *Euro-Par 2007 Parallel Processing*, volume 4641 of *Lecture Notes in Computer Science*, pages 758–766. Springer Berlin Heidelberg, 2007.

[2] M. Drozdowski. Scheduling parallel tasks – algorithms and complexity. In J. Leung, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, Inc., Boca Raton, FL, USA, 2004.

[3] P.-F. Dutot. Hierarchical Scheduling for Moldable Tasks. In *Euro-Par*, volume 3648 of *Lecture Notes in Computer Science*, pages 302–311, Lisbonne, Portugal, Aug. 2005. Springer-Verlag.

[4] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[5] M. Jacquelin, L. Marchal, Y. Robert, and B. Uçar. On optimal tree traversals for sparse matrix factorization. In *IPDPS'2011, the 25th IEEE International Parallel and Distributed Processing Symposium*. IEEE Computer Society Press, 2011.

[6] R. Lepère, D. Trystram, and G. J. Woeginger. Approximation algorithms for scheduling malleable tasks under precedence constraints. *Int. J. Found. Comput. Sci.*, 13(4):613–627, 2002.

[7] L. Marchal, O. Sinnen, and F. Vivien. Scheduling tree-shaped task graphs to minimize memory and makespan. *Parallel and Distributed Processing Symposium, International*, 0:839–850, 2013.

[8] G. N. S. Prasanna and B. R. Musicus. Generalized multiprocessor scheduling and applications to matrix computations. *IEEE Trans. Parallel Distrib. Syst.*, 7(6):650–664, 1996.

[9] G. N. S. Prasanna and B. R. Musicus. The optimal control approach to generalized multiprocessor scheduling. *Algorithmica*, 15(1):17–49, 1996.

[10] S. Tomov, J. Dongarra, and M. Baboulin. Towards dense linear algebra for hybrid {GPU} accelerated manycore systems. *Parallel Computing*, 36(5–6):232 – 240, 2010. Parallel Matrix Algorithms and Applications.

Appendix A: Proof of Properties 1 and 4

We prove here that there is a unique PFC-optimal allocation, which is the unique PFC allocation that respects $\Delta_{\mathscr{P}} = 0$.

Proof. In this proof, we need a partial order on the parallel nodes of the graph G. We define the order $\succ$: $b \succ a$ if the node b belongs to the subtree rooted in a.

First, we show that for any PFC-optimal schedule $\mathscr{P}$, we necessarily have $\Delta_{\mathscr{P}} = 0$. Suppose the contrary. Let $\mathscr{P}$ be a PFC-optimal schedule that has a difference of makespan at one parallel node, such that no PFC-optimal schedule has a difference of makespan at a strictly smaller parallel node (according to the order $\succ$).

Then, by rebalancing the processors among the two branches of this node, we can achieve a smaller makespan in this subgraph with the same share of processors. We now have two cases. First, there is no parallel composition above this node, and the new schedule has then a smaller makespan. Otherwise, there is a parallel composition above this node, and the new schedule has the optimal PFC makespan, and has a difference of makespan in a smaller parallel composition than $\mathscr{P}$, which violates the hypothesis. We have a contradiction, and so $\Delta_{\mathscr{P}} = 0$.

Now, we prove that there is exactly one schedule $\mathscr{P}$ using all the processors with $\Delta_{\mathscr{P}} = 0$. This implies by Lemma 1 that the PFC-optimal makespan is unique.

This statement is proven by induction on the structure of G, with the hypothesis: for all G, there is a unique schedule using all the processors with $\Delta_{\mathscr{P}} = 0$ for any value of p, and the makespan decreases in function of p. Then, at a parallel composition, there is only one point of equilibrium between the makespans of the two branches and so the induction is validated. □

Appendix B: Convergence of the heuristic in the case of two parallel tasks

We prove here the convergence of the heuristic presented in Algorithm 1 towards the optimal PFC schedule, for the parallel composition of two tasks T_A and T_B of respective lengths A and B (with $A < B$) in the most difficult case, when $A/B < (p-1)^{\alpha} < B/A$. Indeed, in other cases, either both tasks need more than one processor in the optimal schedule, or they both need less than one. In the first case, the heuristic returns the optimal PM schedule. In the second case, its behavior can be proved by arguments similar to the following ones.

We study two parallel tasks T_A and T_B, to be scheduled with $p > 1$ processors such that:

$$A/B < (p-1) \qquad \text{and} \qquad (p-1)^{\alpha} < B/A$$

This way, in the optimal schedule we have $\bar{p_A} < 1 < \bar{p_B}$, with $\bar{p_A}$ (resp. $\bar{p_B}$) being the share of processors allocated to A (resp. B). Indeed, suppose $\bar{p_A} = 1$. Then, as $A < B$, we have $A/1 = B/(p-1)^\alpha$, so $B/A = (p-1)^\alpha$. Therefore, $\bar{p_A} < 1$ is implied by $B/A > (p-1)^\alpha$. Then, similarly, to ensure $\bar{p_B} > 1$, we need $A/B < (p-1)$.

During the execution of the algorithm, for all $\alpha \geq 0.5$ and for all j, we will consecutively prove that we have:

1. if $p_A^0 \leq p_A^j < 1$, then $p_A^{j+1} < 1$
2. $p_A^{2j} < p_A^{2j+2} < p_A^{2j+3} < p_A^{2j+1}$
3. p_A^j converges towards $\bar{p_A}$

Claim 1. *if* $p_A^0 \leq p_A^j < 1$, *then* $p_A^{j+1} < 1$.

Proof. First, we compute $\bar{q}$ such that if we replace A by $A \cdot \bar{q}^{\alpha-1}$ and compute the PM schedule, we will obtain $p_A = 1$. Then, if $p_A^j > \bar{q}$, we will obtain $p_A^{j+1} < 1$.

We have:

$$\frac{A\bar{q}^{\alpha-1}}{1} = \frac{B}{(p-1)^\alpha}$$

$$\bar{q} = \left((p-1)^\alpha \frac{A}{B}\right)^{1/(1-\alpha)} < 1$$

We have $p_A^0 = \frac{p}{1+\left(\frac{B}{A}\right)^{1/\alpha}}$

Then, we have:

$$
\begin{aligned}
(p-1)^\alpha &< \frac{B}{A} \\
p &< 1+\left(\frac{B}{A}\right)^{1/\alpha} \\
\frac{1}{p} &> \frac{1}{1+\left(\frac{B}{A}\right)^{1/\alpha}} \\
1-\frac{1}{p} &< \frac{1}{1+\left(\frac{A}{B}\right)^{1/\alpha}} \\
\frac{p-1}{p} &< \frac{1}{1+\left(\frac{A}{B}\right)^{1/\alpha}} \\
(p-1)\left(\frac{A}{B}\right)^{1/\alpha} &< \frac{p}{1+\left(\frac{B}{A}\right)^{1/\alpha}} \\
\left((p-1)^\alpha\left(\frac{A}{B}\right)\right)^{1/\alpha} &< p_A^0 \\
\left((p-1)^\alpha\left(\frac{A}{B}\right)\right)^{1/(1-\alpha)} &< p_A^0 \\
\bar{q} &< p_A^0
\end{aligned}
$$

because $\bar{q} < 1$ and $\alpha \geq 1/2$.

Then, by hypothesis, $p_A^0 < p_A^j$ so $p_A^{j+1} < 1$ □

Claim 2. *For all j, $p_A^{2j} < p_A^{2j+2} < p_A^{2j+3} < p_A^{2j+1}$ and $\frac{A}{p_A^{2j+2}} > \frac{B}{(p-p_A^{2j+2})^\alpha}$ and $\frac{A}{p_A^{2j+1}} < \frac{B}{(p-p_A^{2j+1})^\alpha}$.*

Proof. We prove the result by induction on j. We have $\frac{A}{p_A^{2j}} > \frac{B}{(p-p_A^{2j})^\alpha}$ and $p_A^{2j} < p_A^{2j+1}$ for $j = 0$. Suppose the result true for all $k < j$. Results involving negative indices are supposed true by convention, so the result is true for all $k < 0$.

We will show that $p_A^{2j} < p_A^{2j+2} < p_A^{2j+3} < p_A^{2j+1}$ and $\frac{A}{p_A^{2j+2}} > \frac{B}{(p-p_A^{2j+2})^\alpha}$ and $\frac{A}{p_A^{2j+1}} < \frac{B}{(p-p_A^{2j+1})^\alpha}$. In order to simplify the formulas, we denote $q_j \doteq p_A^j$ and we replace $2j$ by j. Because of Claim 1, as we know that $q_j > q_0$, we know that $q_{j+1} < 1$. By definition of A_{j+1} and q_{j+1}, we have

$$
\begin{aligned}
A_{j+1} &= A_0 q_j^{\alpha-1} > A_0 \\
\frac{A_{j+1}}{q_{j+1}^{\alpha}} &= \frac{B}{(p-q_{j+1})^{\alpha}} \\
\frac{A_0}{q_j}\left(\frac{q_j}{q_{j+1}}\right)^{\alpha} &= \frac{B}{(p-q_{j+1})^{\alpha}} \\
\frac{A_0}{q_{j+1}} &= \frac{B}{(p-q_{j+1})^{\alpha}}\left(\frac{q_j}{q_{j+1}}\right)^{1-\alpha}
\end{aligned}
$$

Let's show that $q_{j+1} > q_j$. We have

$$
\begin{aligned}
\frac{A_0}{q_j} &< \frac{B}{(p-q_j)^{\alpha}} \\
\frac{A_0}{q_j} &= \frac{B}{(p-q_{j+1})^{\alpha}}\left(\frac{q_{j+1}}{q_j}\right)^{\alpha}
\end{aligned}
$$

then

$$
\frac{q_j^{\alpha}}{(p-q_j)^{\alpha}} > \frac{q_{j+1}^{\alpha}}{(p-q_{j+1})^{\alpha}}
$$

Therefore, $q_{j+1} > q_j$ and so $\frac{A_0}{q_{j+1}} < \frac{B}{(p-q_{j+1})^{\alpha}}$. Then, by Claim 1, $q_{j+2} < 1$.

At the next step, $A_{j+2} = A_0 q_{j+1}^{\alpha-1}$. So, $A_0 < A_{j+2} < A_{j+1}$. By similar formulas, we show that $\frac{A_0}{q_{j+2}} < \frac{B}{(p-q_{j+2})^{\alpha}}$. If $j = 0$, we have $A_j < A_{j+2}$. Otherwise, we know by induction that $q_{j+1} < q_{j-1}$. Therefore, we also have the result $q_j < q_{j+2} < q_{j+1}$. At the following step, $A_{j+3} = A_0 q_{j+2}^{\alpha-1}$. So $A_{j+1} > A_{j+3} > A_{j+2}$. □

Claim 3. *p_A^j converges towards $\bar{p_A}$.*

Proof. We know that A^{2j} and A^{2j+1} both converge, to A_e and A_o.

We have

$$
\begin{aligned}
A_e &= A q_o^{\alpha-1} \\
A_o &= A q_e^{\alpha-1}
\end{aligned}
$$

$$
\begin{aligned}
\frac{B}{(p-q_o)^{\alpha}} &= \frac{B}{(p-q_e)^{\alpha}}\left(\frac{q_e}{q_o}\right)^{\alpha}\left(\frac{q_o}{q_e}\right)^{(1-\alpha)} \\
\frac{q_o^{2\alpha-1}}{(p-q_o)^{\alpha}} &= \frac{q_e^{2\alpha-1}}{(p-q_e)^{\alpha}}
\end{aligned}
$$

As $\alpha \geq 1/2$, by monotonicity, we must have $q_o = q_e$ and then A^j converges towards a value A_x.

Therefore,

$$A_x = Aq_x^{\alpha-1}$$

$$\frac{A}{q_x} = \frac{A_x}{q_x^\alpha} = \frac{B}{(p-q_x)^\alpha}$$

so $q_x = \bar{p_A}$. □

Appendix C: Approximate ratios of PM and PFC schedules

The goal here is to prove that with $\alpha = 1/2$, the PM allocation is not a constant ratio approximation for the makespan and no PFC allocation is a 1.09-approximation for the makespan. To achieve this goal, we consider the graph G of Figure 8 to schedule with $3n$ processors. n and B are parameters of the problem, with $n > 0$ and $B > 1$. The lengths of c_1 and c_2, equal to L, will be defined later.

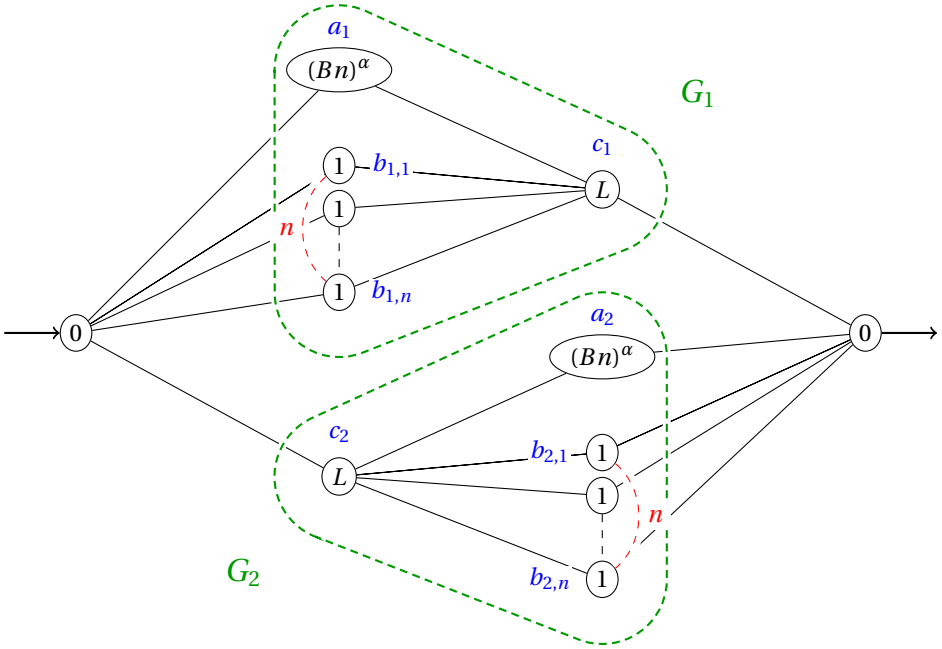


Figure 8: Series-parallel graph G

Consider the schedule $\mathscr{S}$, allocating $2n$ processors to each group of a_i, $b_{i,j}$'s and n to each c_i (the length L will be set such that all the nodes have the same makespan and so this exchange of processors between the branches is possible without loss of makespan). We want the a_i's and $b_{i,j}$'s to have the same makespan. Let $x < 1$ be the share of processors allocated to each $b_{i,j}$. Then, the makespan of each $b_{i,j}$ is $M_b = 1/x$ and the makespan of each a_i is

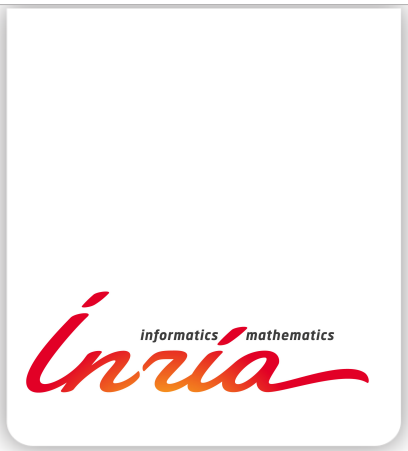
$M_a = (Bn)^\alpha/(2n-nx)^\alpha = (B/(2-x))^{1/2}$. Then, we have $1/x^2 = B/(2-x)$ and so $x = \frac{\sqrt{1+8B}-1}{2B}$. So we have $M_a = M_b = \frac{2B}{\sqrt{1+8B}-1}$.

We set $L = M_a n^\alpha$. Then, the makespan of $\mathscr{S}$ is $M_{\mathscr{S}} = 2M_a = \frac{4B}{\sqrt{1+8B}-1}$.

In the PM schedule $\mathscr{S}_{\text{PM}}$, $3n/2$ processors are dedicated to each subgraph. So the nodes a_i's and $b_{i,j}$'s have a makespan of $M_a^{PM} = 2(B+1)/3$. Indeed, the n nodes $b_{1,j}$'s are equivalent to a task of length n^α so they are allocate a fraction $1/(B+1)$ of the available processors, which is $3n/(2B+2)$. Therfore, the ratio $M_a^{PM}/M_{\mathscr{S}}$ grows to the infinity as B increases, and so $\mathscr{S}_{\text{PM}}$ is not a constant ratio approximation of $\mathscr{S}$.

In the optimal PFC schedule $\mathscr{S}'$, $3n/2$ processors are allocated to each branch as they are equivalent. Then, as for $\mathscr{S}$, replacing 2 by 3/2, we get that the share x allocate to each $b_{i,j}$ is $x = \frac{\sqrt{1+6B}-1}{2B}$. So the makespan for nodes a_i's and $b_{i,j}$'s is $M'_a = \frac{2B}{\sqrt{1+6B}-1}$. And the makespan of c_i's is $M'_c = \left(\frac{2}{3}\right)^\alpha \frac{2B}{\sqrt{1+8B}-1} = \sqrt{\frac{2}{3}}\frac{2B}{\sqrt{1+8B}-1}$.

For $B = 1$, we obtain $(M'_c + M_a)/M_{\mathscr{S}} = \frac{1}{\sqrt{6}} + \frac{1}{\sqrt{7}-1} \simeq 1.0159$. When looking for the worse approximation ratio between of a PFC allocation, we have designed a recursive structure $\mathscr{G}_x$ to increase the error made by PFC allocations. This structure is basically describe by the following process: $\mathscr{G}_0$ equals G where $B = 1$ and $n = 2$, and $\mathscr{G}_x$ is similar to $\mathscr{G}_0$ where nodes a_i and c_i are replaced by $\mathscr{G}_{x-1}$ and the lengths of $b_{i,j}$'s are changed. In simulation, the approximation ratio between the best PFC schedule (computed by the heuristic, and certified by the value of Δ) and a schedule similar to $\mathscr{S}$ has raised to approximately 1.09, which is the largest PFC approximation ratio observed.



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée

655 avenue de l'Europe Montbonnot

38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399