First model: $T = L/p^\alpha$ for all $p$
Second model: $T = L/p$ for $p < 1$
Memory-constrained model

M2 Internship – ENS Lyon
Scheduling series parallel graphs to minimize the makespan

Bertrand Simon

*ENS Lyon – supervised by Loris Marchal and Frederic Vivien*

Feb – Jul 2014
Outline

1 Introduction

2 First model: \( T = L/p^\alpha \) for all \( p \)
   - Definition of the model
   - Computation of the optimal schedule

3 Second model: \( T = L/p \) for \( p < 1 \)
   - Introduction and motivation
   - Restriction to a subset of schedules
   - Heuristic to compute the optimal PFC schedule

4 Memory-constrained model
   - Description of the model
   - Results

5 Conclusion
First model: $T = L/p^\alpha$ for all $p$
Second model: $T = L/p$ for $p < 1$
Memory-constrained model

Outline

1. Introduction
2. First model: $T = L/p^\alpha$ for all $p$
3. Second model: $T = L/p$ for $p < 1$
4. Memory-constrained model
5. Conclusion
### Introduction

**Motivation**
- factorisation of sparse matrices creates task trees to be scheduled
- need to schedule such trees using parallelisation between the nodes and inside each node

**Related work**
- Prasanna, Musicus (1996)
- Marchal, Sinnen, Vivien (2013)

**Goals**
- reprove the results of Prasanna & Musicus using scheduling arguments
- take in account the memory constraints
Outline

1 Introduction

2 First model: $T = L/p^\alpha$ for all $p$
   - Definition of the model
   - Computation of the optimal schedule

3 Second model: $T = L/p$ for $p < 1$

4 Memory-constrained model

5 Conclusion
Model

Goal
Schedule a series-parallel graph (generalisation of trees) $G$ under $p$ processors minimizing the makespan

Hypotheses
- Task of length $L$ is completed with $p$ processors in time $L/p^\alpha$ with $0 < \alpha < 1$
- A non-integer share of processors can be allocated to a task
- We can allocate less than 1 processor to a task
- The tasks are malleable: the share of processors can be modified

Example

![Diagram of a series-parallel graph](image-url)
First model: \( T = \frac{L}{p^\alpha} \) for all \( p \)
Second model: \( T = \frac{L}{p} \) for \( p < 1 \)

Memory-constrained model

Definition of the model

Computation of the optimal schedule

Theorem to prove

Theorem (Prasanna & Musicus)

For any series-parallel graph \( G \), any optimal schedule under a constant number of processors \( p \) respects:

- **each task is assigned a constant processing power**
- **Processor Flow Conservation property**: every 'siblings' terminate at the same time and all the processing power is then assigned to the parent(s)
- **the allocation for different values of \( p \) are proportionals**

Corollary

Under a constant number of processors, each graph \( G \) has the same optimal makespan than the task \( T_G \) of length \( L_G \).

For series: \( L_G = L_1 + L_2 \)

For parallel: \( L_G = \left( L_1^{1/\alpha} + L_2^{1/\alpha} \right)^\alpha \)
Theorem to prove

**Theorem (Prasanna & Musicus)**

For any series-parallel graph $G$, any optimal schedule under a constant number of processors $p$ respects:

- each task is assigned a constant processing power
- Processor Flow Conservation property: every 'siblings' terminate at the same time and all the processing power is then assigned to the parent(s)
- the allocation for different values of $p$ are proportionals

**Corollary**

Under a constant number of processors, each graph $G$ has the same optimal makespan than the task $T_G$ of length $L_G$.

For series: $L_G = L_1 + L_2$  
For parallel: $L_G = \left( L_1^{1/\alpha} + L_2^{1/\alpha} \right)^\alpha$
First lemmas used in the proof

Lemma

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

Definition (Clean interval)

An interval during which no task terminates in the considered schedule.

Lemma

We have to process n independent 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.
First lemmas used in the proof

**Lemma**

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

**Definition (Clean interval)**

An interval during which no task terminates in the considered schedule.

**Lemma**

We have to process $n$ independent 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.
Lemma

Let $G$ be the graph given by the parallel composition of $G_1$ and $G_2$. If a constant number of processors is given to schedule $G$, then any optimal schedule associates a constant number of processors to $G_1$ and to $G_2$.

Proof.

Property used: $(ab)^\alpha = a^\alpha b^\alpha$

Remark

Consequently, $G_1$ and $G_2$ have the same makespan.
First model: \( T = \frac{L}{p^\alpha} \) for all \( p \)

Second model: \( T = \frac{L}{p} \) for \( p < 1 \)

Introduction and motivation
Restriction to a subset of schedules
Heuristic to compute the optimal PFC schedule

Outline

1. Introduction
2. First model: \( T = \frac{L}{p^\alpha} \) for all \( p \)
3. Second model: \( T = \frac{L}{p} \) for \( p < 1 \)
   - Introduction and motivation
   - Restriction to a subset of schedules
   - Heuristic to compute the optimal PFC schedule
4. Memory-constrained model
5. Conclusion
Introduction to the model

Modification

- \( p \geq 1 \): the time to complete a task \( L \) is \( L/p^\alpha \)
- \( p \leq 1 \): the time to complete a task \( L \) is \( L/p \)

Time to complete a task in function of the processing power in the different models

Consequences

The last lemma does not hold so the previous theorem does not hold. We cannot compute the optimal schedule.
Introduction to the model

Modification

- $p \geq 1$: the time to complete a task $L$ is $L/p^\alpha$
- $p \leq 1$: the time to complete a task $L$ is $L/p$

Time to complete a task in function of the processing power in the different models

Consequences

The last lemma does not hold so the previous theorem does not hold. We cannot compute the optimal schedule.
First model: \[ T = \frac{L}{p^\alpha} \] for all \( p \)

Second model: \[ T = \frac{L}{p} \] for \( p < 1 \)

Memory-constrained model

Introduction and motivation

Restriction to a subset of schedules

Heuristic to compute the optimal PFC schedule

Motivation

Definition (PM allocation)

The allocation that associates to each task the share computed by the formulas of previous section.

Theorem

\textit{No PM algorithm is a constant ratio approximation.}
PFC schedules

Definition (PFC schedule)

An allocation that associates a constant share to each subgraph at every parallel node, under a constant number of processors.

Theorem

The optimal PFC allocation is not always the optimal schedule. This statement is still valid on tree-shaped graphs and for moldable tasks.

Example of tree-shaped graph illustrating the theorem, with $p = 4$
**PFC schedules**

**Definition (PFC schedule)**
An allocation that associates a constant share to each subgraph at every parallel node, under a constant number of processors.

**Theorem**
*The optimal PFC allocation is not always the optimal schedule.*
*This statement is still valid on tree-shaped graphs and for moldable tasks.*

**Remark (Optimal PFC allocation seen as an approximation)**
*The approximation ratio is smaller than \( p^{1-\alpha} \).*
*The worst ratio observed is 1.1: the exact ratio is unknown.*

**Remark**
*We can easily check that a PFC allocation is optimal among PFC’s: at every parallel nodes, the two branches must terminate at the same time.*
*But we cannot compute the optimal PFC.*
New goal

Compute the makespan-optimal PFC schedule.
No exact algorithm is known, so use of a heuristic.

Idea

- In PM schedules: the makespan of tasks with less than 1 processors is underestimated
- Artificially increase the need of processors: increase the length $L$ seen by the algorithm
- Goal: $L/p = \bar{L}/p^\alpha \Rightarrow \bar{L} := L \cdot p^{\alpha-1} > L$
- Algorithm: compute the PM schedule, modify the $L_i$, and iterate

Remark

- For any graph $G$ scheduled, define $A_G$ the largest difference of makespan between two parallel branches
- $A_G$ is a way to check the schedule returned by the heuristic
Results of the heuristic

Results proved

- on any two-tasks graph $G$, there exists $\alpha_G < 0.5$ such that for any $\alpha > \alpha_G$, the heuristic converges
- for $\alpha > \alpha_G$, the odd and even subsequences of $L_i$ are monotone and adjacent
- for $\alpha < \alpha_G$, the heuristic does not converge

Observations on random/selected graphs

- for any graph $G$, for any $\alpha > 0.5$, the heuristic converges
- $A_{G^{2j}}$ and $A_{G^{2j+1}}$ decrease and converge to 0
Outline

1. Introduction

2. First model: \( T = L/p^\alpha \) for all \( p \)

3. Second model: \( T = L/p \) for \( p < 1 \)

4. Memory-constrained model
   - Description of the model
   - Results

5. Conclusion
Introduction

Assumptions
- each output file is of size $f_i = 1$
- each task has an internal memory need of $n_i = 0$
- to execute a task, the output files of the children and of the task must be allocated
- the lengths of the tasks are $w_i \in \{0, 1\}$ (unique difference with Pebble Game)
- the time to complete a task of size $w_i$ with $p$ processors is always $w_i/p^\alpha$

Lemma (backbone of the following theorems)

Under the hypotheses:
- $tn$ independent tasks of length 1
- $tp$ processors
- memory constraints forbid to parallelize more than $t$ tasks

Optimality $\iff$ batches of $t$ tasks
First model: $T = L/p\alpha$ for all $p$
Second model: $T = L/p$ for $p < 1$

Memory-constrained model

Introduction

Assumptions
- each output file is of size $f_i = 1$
- each task has an internal memory need of $n_i = 0$
- to execute a task, the output files of the children and of the task must be allocated
- the lengths of the tasks are $w_i \in \{0, 1\}$ (unique difference with Pebble Game)
- the time to complete a task of size $w_i$ with $p$ processors is always $w_i/p^\alpha$

Lemma (backbone of the following theorems)

Under the hypotheses:
- $tn$ independent tasks of length 1
- $tp$ processors
- memory constraints forbid to parallelize more than $t$ tasks

Optimality $\Leftrightarrow$ batches of $t$ tasks
NP-completeness of the BiObjective problem

The BiObjectiveParallelTreeScheduling problem

Given a tree-shaped task graph $T$ provided with memory weights and task durations, $p$ processors, and two bounds $B_{C_{\text{max}}}$ and $B_{\text{mem}}$, is there a schedule of the task graph on the processors whose makespan is not larger than $B_{C_{\text{max}}}$ and whose peak memory is not larger than $B_{\text{mem}}$?

Theorem

The BiObjectiveParallelTreeScheduling problem is NP-Complete.

Proof.

Reduction from 3-partition.
Inapproximation results

**Theorem**

There is no algorithm $A$ that is both a $\beta$-approximation for makespan minimization and a $\gamma$-approximation for memory peak minimization when scheduling in-tree task graphs.

**Theorem**

When scheduling in-tree task graphs with $p$ processors, there is no algorithm $A$ that is both a $\beta(p)$-approximation for makespan minimization and a $\gamma(p)$-approximation for memory peak minimization with $\beta(p)$ and $\gamma(p)$ verifying:

$$\gamma(p)\beta(p)^{1-\alpha} > \left(\frac{p}{\log p + 1}\right)^{1-\alpha}$$
First model: $T = L/p^\alpha$ for all $p$

Second model: $T = L/p$ for $p < 1$

Memory-constrained model
Conclusion

Model \( T = \frac{L}{p^\alpha} \) for all \( p \)
- Every graph is equivalent to a task, which length is computable efficiently

Model \( T = \frac{L}{p} \) for \( p < 1 \)
- PM schedules are not \( \lambda \)-approximations
- PFC schedules are not optimal
- A heuristic probably converges towards the PFC optimal schedule
- The approximation ratio is not known

Memory-aware model
- Deciding if there exists a schedules that respects a makespan and a memory constraint is NP-complete
- There is a bound over the approximate ratios
Future work

Model $T = L/p$ for $p < 1$
- Compute the approximation ratio of the optimal PFC allocation
- (Dis)prove the convergence of the heuristic

Memory-aware model
- Design a heuristic with guarantees on makespan and memory peak (in progress)