Scheduling Malleable Task Trees

Bertrand Simon  Loris Marchal  Frédéric Vivien

ENS Lyon

9th Scheduling for Large Scale Systems Workshop, Lyon 2014
Outline

1 Introduction and notations

2 Minimizing the makespan
   - Characterization of the optimal schedule
   - Scheme of the proof of the theorem

3 Minimizing the makespan with a modified speedup function
   - The refinement and its consequences
   - Computing the best PFC allocation

4 Minimizing the makespan and memory peak
   - Description of the model
   - Complexity results

5 Conclusion
Outline

1 Introduction and notations
2 Minimizing the makespan
3 Minimizing the makespan with a modified speedup function
4 Minimizing the makespan and memory peak
5 Conclusion
Introduction

Motivation

- Solving sparse linear systems $\rightarrow$ sparse matrix factorizations $\rightarrow$ task trees to be scheduled
- Processing power available: homogeneous parallel platform
- Need to schedule task trees using tree and task parallelism

Definitions

- **task tree**: structure defining precedence order, a node cannot begin before its children are completed
- **tree parallelism**: possibility to execute simultaneously several tasks
- **task parallelism**: possibility to allocate several processors to a task
Introduction

Motivation

- Solving sparse linear systems → sparse matrix factorizations → task trees to be scheduled
- Processing power available: homogeneous parallel platform
- Need to schedule task trees using tree and task parallelism

Definitions

- task tree: structure defining precedence order, a node cannot begin before its children are completed
- tree parallelism: possibility to execute simultaneously several tasks
- task parallelism: possibility to allocate several processors to a task
Minimizing the makespan

Minimizing the makespan with a modified speedup function

Minimizing the makespan and memory peak

Introduction

Motivation

- Solving sparse linear systems $\rightarrow$ sparse matrix factorizations $\rightarrow$ task trees to be scheduled
- Processing power available: homogeneous parallel platform
- Need to schedule task trees using tree and task parallelism

![Task Tree Diagram]

Definitions

- **task tree**: structure defining precedence order, a node cannot begin before its children are completed
- **tree parallelism**: possibility to execute simultaneously several tasks
- **task parallelism**: possibility to allocate several processors to a task
Introduction

Motivation
- Solving sparse linear systems $\rightarrow$ sparse matrix factorizations $\rightarrow$ task trees to be scheduled

- Processing power available: homogeneous parallel platform

- Need to schedule task trees using tree and task parallelism

Definitions
- **task tree**: structure defining precedence order, a node cannot begin before its children are completed

- **tree parallelism**: possibility to execute simultaneously several tasks

- **task parallelism**: possibility to allocate several processors to a task
Introduction

Motivation

- Solving sparse linear systems $\rightarrow$ sparse matrix factorizations $\rightarrow$ task trees to be scheduled
- Processing power available: homogeneous parallel platform
- Need to schedule task trees using tree and task parallelism

Definitions

- **task tree**: structure defining precedence order, a node cannot begin before its children are completed
- **tree parallelism**: possibility to execute simultaneously several tasks
- **task parallelism**: possibility to allocate several processors to a task
Model and notations

Parameters of the problem

- Need for a model of realist (imperfect) task parallelism: Malleable tasks [Le04]
- Tree graph $G$ (previous slide)
- Processor profile: step function $p(t)$, available number of processors at time $t$

Speedup $f$ (=sequential time / parallel time)

- $f(p) = p^\alpha$ for $0 < \alpha < 1$, $p \in \mathbb{R}^+$ (non-integer processor shares: time-sharing techniques)
  - Advocated for matrix computations [PM96,BG07]
- Processing time of task $T_i$ on $p$ processors: $L_i/p^\alpha$
Parameters of the problem

- Need for a model of realist (imperfect) task parallelism: **Malleable tasks** [Le04]
- **Tree graph** $G$ (previous slide)
- **Processor profile**: step function $p(t)$, available number of processors at time $t$

Speedup $f$ (= sequential time / parallel time)

- $f(p) = p^\alpha$ for $0 < \alpha < 1$, $p \in \mathbb{R}^+$ (non-integer processor shares: time-sharing techniques)
  - Advocated for matrix computations [PM96,BG07]
- Processing time of task $T_i$ on $p$ processors: $L_i / p^\alpha$
Definition of schedules

**Structure of schedules**

- Schedule $\mathcal{S}$: piecewise continuous functions $\{t \mapsto p_i(t)\}$ defined on $[0, \tau]$
- $\tau$: makespan of $\mathcal{S}$ (supposed tight: not all $p_i(\tau - \epsilon)$ are null)
- Ratio of work up to time $t$: $w_i(t) = \int_{0}^{t} p_i(x)\alpha \, dx / L_i$

**Validity conditions of a schedule**

- Does not use more than $p(t)$ processors at any time $t$: $\sum_i p_i(t) \leq p(t)$
- Completes all the tasks: $\forall i$, $w_i(\tau) = 1$
- Respects the precedence order: $\forall i$, $\forall t \in [0, \tau]$, $w_i(t) > 0 \implies \forall j \in \text{Children}(T_i)$, $w_j(t) = 1$
Generalization of trees

Our objective: study trees
Next two sections: study a more general structure

Series Parallel graphs

Recursively defined by being either:
- a single task
- a parallel composition of two SP graphs
- a series composition of two SP graphs

A tree can be extended to a SP graph.
Generalization of trees

Our objective: study trees
Next two sections: study a more general structure

Series Parallel graphs

Recursively defined by being either:
- a single task
- a parallel composition of two SP graphs
- a series composition of two SP graphs

A tree can be extended to a SP graph.
Generalization of trees

Our objective: study trees
Next two sections: study a more general structure

Series Parallel graphs

Recursively defined by being either:
- a single task
- a parallel composition of two SP graphs
- a series composition of two SP graphs

A tree can be extended to a SP graph.
Outline

1. Introduction and notations

2. Minimizing the makespan
   - Characterization of the optimal schedule
   - Scheme of the proof of the theorem

3. Minimizing the makespan with a modified speedup function

4. Minimizing the makespan and memory peak

5. Conclusion
Statement of the problem

Context
- Objects of interest: minimum-makespan schedules of a SP graph $G$
- [PM96] proved the theorem below using heavy optimal control theory
- Our objective: reprove it using pure-scheduling arguments

Theorem (Prasanna & Musicus)

Optimal schedules respect the **Processor Flow Conservation property**: the ratio of processors given to each branch of any parallel node is constant.
Consequences of the theorem

**Corollary**

- **Each task:** allotted a constant ratio, independent of \( p(t) \)
  - its children terminate simultaneously

- **Each graph** \( G \) is equivalent to the task of length \( L_G \) recursively defined by:
  - \( L_{T_i} = L_i \)
  - \( L_{G_1 \parallel G_2} = L_{G_1} + L_{G_2} \)
  - \( L_{G_1 \parallel G_2} = \left( L_{G_1}^{1/\alpha} + L_{G_2}^{1/\alpha} \right)^\alpha \)

- **The (unique) optimal schedule** \( S_{PM} \) can be computed in polynomial time.

A tree \( G \) (particular SP graph) and the shape of its optimal schedule under any \( p(t) \).
First step of the proof: \( p_i(t)'s \) are step functions

A **clean interval** of a schedule \( \mathcal{S} \): a time interval during which no task terminates.

**Lemma**

*If \( p(t) = p \), optimal schedules have constant \( p_i(t)'s \) on its clean intervals.*

**Proof.**

- Consider \( \mathcal{S} \) with \( p_j(t) \) not constant on a clean \( \Delta \rightarrow \mathcal{S}' \) with smaller makespan
- Uses strict concavity of \( f \): replace \( p_i(t)'s \) by their mean
- Get the inequality: 
  \[
  W_j^\Delta(\mathcal{S}) = \int_\Delta p_j(t)^\alpha \, dt < \int_\Delta \left( \frac{1}{\Delta} \int_\Delta p_j(t) \, dt \right)^\alpha \, dx
  \]
First step of the proof: \( p_i(t) \)'s are step functions

A **clean interval** of a schedule \( \mathcal{S} \): a time interval during which no task terminates.

**Lemma**

If \( p(t) = p \), optimal schedules have constant \( p_i(t) \)'s on its clean intervals.

**Proof.**

- Consider \( \mathcal{S} \) with \( p_j(t) \) not constant on a clean \( \Delta \to \mathcal{S}' \) with smaller makespan
- Uses strict concavity of \( f \): replace \( p_i(t) \)'s by their mean
- Get the inequality: 
  \[
  W_j^\Delta(\mathcal{S}) = \int_\Delta p_j(t)^\alpha dt < \int_\Delta \left( \frac{1}{\Delta} \int_\Delta p_j(t) dt \right)^\alpha dx
  \]
First step of the proof: $p_i(t)$’s are step functions

A **clean interval** of a schedule $\mathcal{S}$: a time interval during which no task terminates.

**Lemma**

If $p(t) = p$, optimal schedules have constant $p_i(t)$’s on its clean intervals.

**Proof.**

- Consider $\mathcal{S}$ with $p_j(t)$ not constant on a clean $\Delta \rightarrow \mathcal{S}'$ with smaller makespan
- Uses strict concavity of $f$: replace $p_i(t)$’s by their mean
- Get the inequality: $W_j^\Delta(\mathcal{S}) = \int_{\Delta} p_j(t)^\alpha dt < \int_{\Delta} \left( \frac{1}{\Delta} \int_{\Delta} p_j(t) dt \right)^\alpha dx$
For any $G$: let $r_i(t) = p_i(t)/p(t)$ be the fraction of processors allocated to $T_i$.

**Lemma**

For $G$ being $T_1 \parallel T_2$, in optimal schedules: $r_1(t) = L_{1}^{1/\alpha} / L_{1\parallel 2}^{1/\alpha}$.

**Proof.** (Note that $p(t)$ is not necessarily constant)

- Suppose $S$ optimal with $r_1(t)$ not constant $\rightarrow S'$ with a smaller makespan
- Properties used: strict concavity of $f$ and $\forall xy, f(xy) = f(x)f(y)$

Details: with $Ap_A^\alpha = Bp_B^\alpha$ and $2r_1 = r_A^1 + r_B^1$,

$$\frac{(r_B^1)^\alpha - (r_1^1)^\alpha}{r_B^1 - r_1^1} < \frac{(r_1^1)^\alpha - (r_A^1)^\alpha}{r_1^1 - r_A^1} \Rightarrow Ap_A^\alpha (r_A^1)^\alpha + Bq_B^\alpha (r_B^1)^\alpha < r_1^1 (Ap_A^\alpha + Bq_B^\alpha)$$
For any $G$: let $r_i(t) = p_i(t)/p(t)$ be the fraction of processors allocated to $T_i$.

**Lemma**

For $G$ being $T_1 \parallel T_2$, in optimal schedules: $r_1(t) = L_1^{1/\alpha} / L_1^{1/\alpha}$.

**Proof.** (Note that $p(t)$ is not necessarily constant)

- Suppose $S$ optimal with $r_1(t)$ not constant $\rightarrow S'$ with a smaller makespan
- Properties used: strict concavity of $f$ and $\forall xy$, $f(xy) = f(x)f(y)$

Details: with $Ap_A^\alpha = Bp_B^\alpha$ and $2r_1 = r_A^A + r_B^B$,

$$\frac{(r_B^B)^\alpha - (r_1)^\alpha}{r_B^B - r_1} < \frac{(r_1)^\alpha - (r_A^A)^\alpha}{r_1 - r_A^A} \Rightarrow Ap_A^\alpha \left(r_A^A\right)^\alpha + Bq_B^\alpha \left(r_B^B\right)^\alpha < r_1^\alpha \left(Ap_A^\alpha + Bq_B^\alpha\right)$$
For any $G$: let $r_i(t) = p_i(t)/\lambda(t)$ be the fraction of processors allocated to $T_i$. 

**Lemma**

*For $G$ being $T_1 \parallel T_2$, in optimal schedules: $r_1(t) = L_1^{1/\alpha} / L_1^{1/\alpha}$.*

**Proof.** *(Note that $\lambda(t)$ is not necessarily constant)*

- Suppose $\mathcal{S}$ optimal with $r_1(t)$ not constant $\rightarrow \mathcal{S}'$ with a smaller makespan
- Properties used: strict concavity of $f$ and $\forall xy, f(xy) = f(x)f(y)$

Details: with $Ap_A^\alpha = Bp_B^\alpha$ and $2r_1 = r_1^A + r_1^B$,

\[
\frac{(r_1^B)^\alpha - (r_1)^\alpha}{r_1^B - r_1} < \frac{(r_1)^\alpha - (r_1^A)^\alpha}{r_1 - r_1^A} \implies Ap^\alpha (r_1^A)^\alpha + Bq^\alpha (r_1^B)^\alpha < r_1^\alpha (Ap^\alpha + Bq^\alpha)
\]
End of the proof of the theorem

Few steps remaining to prove the theorem:

- $T_1 \parallel T_2$ under any $p(t)$ $\iff$ $T_1 \parallel 2$ of length $L_1 \parallel 2$ under any $p(t)$
- $T_1; T_2$ under any $p(t)$ $\iff$ $T_1; 2$ of length $L_1; 2$ under any $p(t)$
- Proof by induction on the structure of $G$

- $p(t) = 6$

Example of computed schedule
End of the proof of the theorem

Few steps remaining to prove the theorem:

- $T_1 \parallel T_2$ under any $p(t) \iff T_1 \parallel 2$ of length $L_1 \parallel 2$ under any $p(t)$
- $T_1 ; T_2$ under any $p(t) \iff T_1 ; 2$ of length $L_1 ; 2$ under any $p(t)$
- Proof by induction on the structure of $G$

- $p(t) = 6$
- $M = \left( \frac{2}{3} \right)^{\alpha} + \left( \frac{4}{3} \right)^{\alpha}$

Example of computed schedule
Outline

1. Introduction and notations
2. Minimizing the makespan
3. Minimizing the makespan with a modified speedup function
   - The refinement and its consequences
   - Computing the best PFC allocation
4. Minimizing the makespan and memory peak
5. Conclusion
Motivation: the previous model overestimates the speedup for \( p < 1 \)

Modification of the speedup function

- \( p \geq 1: \ f(p) = p^\alpha \)
- \( p \leq 1: \ f(p) = p \)

Consequences

The previous theorem does not hold.
We cannot compute the optimal schedule.
Restriction: assume \( p(t) = p \) in the following.
Definition (PM allocation)

The allocation $\mathcal{A}_{PM}$ computed by the formulas of previous section.

Theorem

*The PM allocation is not a constant ratio approximation at $\alpha$ fixed.*

- $p(t) = 6$

Example of graph where the PM allocation is not optimal
Consequence of the refinement

Definition (PM allocation)

The allocation $\mathcal{A}_{PM}$ computed by the formulas of previous section.

Theorem

*The PM allocation is not a constant ratio approximation at $\alpha$ fixed.*

- $p(t) = 6$
- PM schedule, optimal with previous model
- $M_1 = \left(\frac{2}{3}\right)^\alpha + \frac{4}{3}$

Example of graph where the PM allocation is not optimal
Consequence of the refinement

Definition (PM allocation)
The allocation $\mathcal{A}_{PM}$ computed by the formulas of previous section.

Theorem
The PM allocation is not a constant ratio approximation at $\alpha$ fixed.

- $p(t) = 6$
- Better schedule
- $M_2 = 2 < M_1$

Example of graph where the PM allocation is not optimal
Consequence of the refinement

**Definition (PM allocation)**

The allocation $\mathcal{A}_{PM}$ computed by the formulas of previous section.

**Theorem**

The PM allocation is not a constant ratio approximation at $\alpha$ fixed.

- $p(t) = 6$

![Diagram](image)

Example of graph where the PM allocation is not optimal
Consequence of the refinement

Definition (PM allocation)
The allocation $\mathcal{A}_{PM}$ computed by the formulas of previous section.

Theorem
*The PM allocation is not a constant ratio approximation at $\alpha$ fixed.*

- $p(t) = 6$

Example of graph where the PM allocation is not optimal

Need to extend the study to more general allocations...
PFC allocations

Need for a more general structure, close to an optimal solution, and simple to study

Definition (PFC allocation)
An allocation that allocates a constant ratio to each subgraph at every parallel node.

Theorem
The (unique) best PFC allocation is not always the optimal schedule, even in the restriction to pseudo-trees.

\[ p(t) = 4 \]

Example of pseudo-tree graph illustrating the theorem
Need for a more general structure, close to an optimal solution, and simple to study

**Definition (PFC allocation)**

An allocation that allocates a constant ratio to each subgraph at every parallel node.

**Theorem**

*The (unique) best PFC allocation is not always the optimal schedule, even in the restriction to pseudo-trees.*

- $p(t) = 4$
- PFC schedules
- $M_1(x, \alpha) > 2$

Example of pseudo-tree graph illustrating the theorem
PFC allocations

Need for a more general structure, close to an optimal solution, and simple to study

**Definition (PFC allocation)**
An allocation that allocates a constant ratio to each subgraph at every parallel node.

**Theorem**
*The (unique) best PFC allocation is not always the optimal schedule, even in the restriction to pseudo-trees.*

- $p(t) = 4$
- Better schedule
- $M_2 = 2 < M_1(x, \alpha)$

Example of pseudo-tree graph illustrating the theorem
PFC allocations

Need for a more general structure, close to an optimal solution, and simple to study

Definition (PFC allocation)

An allocation that allocates a constant ratio to each subgraph at every parallel node.

Theorem

*The (unique) best PFC allocation is not always the optimal schedule, even in the restriction to pseudo-trees.*

Remark (best PFC allocation seen as an approximation)

*Approximation ratio* \(< p^{1-\alpha}.\)

*For* \(\alpha = 1/2: \) *approximation ratio* > 1.09 \(\rightarrow\) *the exact ratio is unknown.*

Remark

*Possibility to check if a PFC allocation is the best one (existence of idle times)* . . .

* . . . but not to compute it.*
Heuristic towards the computation of the best PFC allocation

**Principle of the heuristic**
- In the PM schedule: makespan of tasks with $p_i < 1$ is underestimated
- Artificially increase their processor need
- Goal: find $L_i$ from $L_i$ such that $L_i/p_i = \bar{L}_i/p_i^\alpha \rightarrow \bar{L}_i := L_i \cdot p_i^{\alpha - 1} > L_i$

**Iterative algorithm**
1. Initialisation: $G_0 \leftarrow G$
2. Repeat step $k$ until (hoped) convergence:
   - Compute the PM schedule $\mathcal{S}_k$ of $G_k$
   - Modify the $L_i$’s with $p_i < 1$ to create $G_{k+1}$

**Elements towards its correctness for $\alpha > 1/2$**
- Convergence is proved on $T_1 \parallel T_2$
- Observations on random/selected graphs:
  - For any graph $G$ the heuristic converges
  - Both $\Delta_{2k}$ and $\Delta_{2k+1}$ decrease and converge to 0
  $\Delta_k$: largest idle time of $\mathcal{S}_k$
Minimizing the makespan

Minimizing the makespan with a modified speedup function

Minimizing the makespan and memory peak

The refinement and its consequences

Computing the best PFC allocation

Heuristic towards the computation of the best PFC allocation

Principle of the heuristic

- In the PM schedule: makespan of tasks with $p_i < 1$ is underestimated
- Artificially increase their processor need
- Goal: find $L_i$ from $L_i$ such that $L_i/p_i = L_i/p_i^\alpha \rightarrow L_i := L_i \cdot p_i^\alpha - 1 > L_i$

Iterative algorithm

1. Initialisation: $G_0 \leftarrow G$
2. Repeat step $k$ until (hoped) convergence:
   - compute the PM schedule $S_k$ of $G_k$
   - modify the $L_i$’s with $p_i < 1$ to create $G_{k+1}$

Elements towards its correctness for $\alpha > 1/2$

- Convergence is proved on $T_1 \parallel T_2$
- Observations on random/selected graphs:
  - For any graph $G$ the heuristic converges
  - Both $\Delta_{2k}$ and $\Delta_{2k+1}$ decrease and converge to 0

$\Delta_k$: largest idle time of $S_k$
Outline

1. Introduction and notations
2. Minimizing the makespan
3. Minimizing the makespan with a modified speedup function
4. Minimizing the makespan and memory peak
   - Description of the model
   - Complexity results
5. Conclusion
Memory: constraint on parallel platforms for direct sparse matrix factorization methods

Objective
Complexity results on schedules trying to minimize both makespan and memory peak

Assumptions on the instance of the problem
- $G$ is a tree, $f(p) = p^\alpha$ and $p(t)$ is constant
- Tasks have output files
- While executing a task, input and output files must be allocated
- In our proofs: file sizes are equal to 1 and lengths to 0 or 1

Lemma (backbone of the following theorems)
Regardless general memory constraints, under the hypotheses:
- $G$: $k \times n$ independent tasks of length 1
- $p(t) = k \times p$
- processing more than $k$ tasks simultaneously is forbidden
Minimum makespan is reached iff successive batches of $k$ tasks are scheduled.
Describing the model

Illustration of the optimal schedule, for $k = 3$ and $n = 5$

**Lemma (backbone of the following theorems)**

Regardless general memory constraints, under the hypotheses:

- $G$: $k \times n$ independent tasks of length 1
- $p(t) = k \times p$
- *processing more than $k$ tasks simultaneously is forbidden*

Minimum makespan is reached iff successive batches of $k$ tasks are scheduled.
NP-completeness of the bi-objective problem

The BiObjectiveParallelTreeScheduling problem

Given a valid instance: is there a schedule respecting \( \{\text{makespan} < B_{C_{\text{max}}}\} \) and \( \{\text{memory peak} < B_{\text{mem}}\} \)?

Theorem

The BiObjectiveParallelTreeScheduling problem is NP-Complete.

Proof.

Reduction from 3-PARTITION

Bertrand Simon, Loris Marchal, Frédéric Vivien
Scheduling Malleable Task Trees
Inapproximation results

**Theorem (unbounded number of processors)**

There is no algorithm that is both a $\beta$-approximation for the makespan and a $\gamma$-approximation for the memory peak.

**Theorem (fixed number of processors)**

There is no algorithm with $\beta(p)$ and $\gamma(p)$ verifying:

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

**Remark (Comparison with previous bounds)**

Without task parallelism [MSV13]:

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

Here, assuming $\alpha = 0$:

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

1. Introduction and notations
2. Minimizing the makespan
3. Minimizing the makespan with a modified speedup function
4. Minimizing the makespan and memory peak
5. Conclusion
Conclusion

Model $f(p) = p^\alpha$ for all $p$
- Results of [PM96] are proved using pure-scheduling arguments

Model $f(p) = p$ for $p < 1$
- PM schedules are not $\lambda$-approximations, PFC schedules are not optimal
- A heuristic probably converges towards the PFC optimal schedule for $\alpha > 1/2$

Memory-aware model
- Deciding if there exists a schedule that respects a makespan and a memory constraint is NP-complete
- There is a lower bound over the approximation ratios, coherent with the state-of-the-art bound without task parallelism