On scheduling malleable jobs to minimise the total weighted completion time

Ruslan Sadykov

INRIA Bordeaux — Sud-Ouest

Institut Mathématique de Bordeaux

INCOM'09
June 5, 2009

Contents

Scheduling parallel jobs

Classic scheduling

A **classic job** can be executed on at most one processor (machine) at the same time.

Parallel scheduling

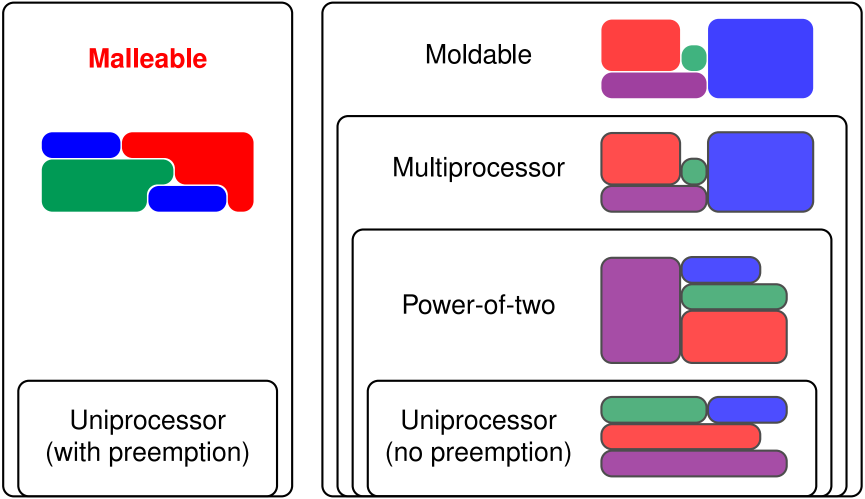
A **parallel job** can be executed on more than one processor at the same time.

δ_j — upper bound on the number of processors that may be used by job j.

Applications

- **Parallel computer applications**
- Reliable computing
- Bandwidth allocation
- Manufacturing
  - Printed Circuit Boards
  - Textile
  - ...

Types of parallel jobs



Cost of parallelism

The processing time p_j of job j depends on the number of machines assigned to it:

- $p_j(q) = p_j(1)/q$ (j is work preserving, no parallelism cost)
- $p_j(q) > p_j(1)/q$ (parallelism costs)
  - $p_j(q) = f(q)$ (particular continuous function)
  - $p_j(q)$ is an arbitrary discrete function of q.

Contents

The problem

- m identical machines
- n malleable jobs
- $\forall j$, $p_j(q)$ — processing time function
- $\forall j$, δ_j — parallelization limit
- $\forall j$, w_j — weight
- Objective function: $\min \sum_j w_j C_j$

$\alpha|\beta|\gamma$ notation

$$P \mid var,\ \delta_j \mid \sum w_j C_j$$

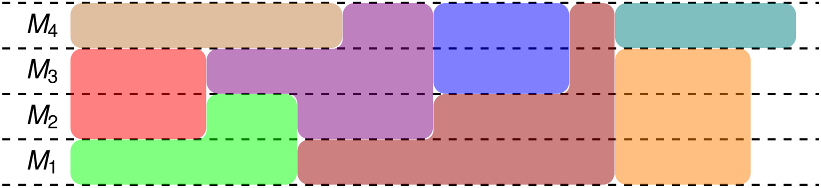
Complexity status

NP-hard (generalisation of $P \mid pmtn \mid \sum w_j C_j$)

Ascending property

A schedule satisfies the ascending property if, for every job j, the number of processors j is executed on do not decrease over time (until j is fully completed).

Example of schedule satisfying the ascending property



The result

Dominance

- For the work preserving case
- and some other (more practical **!**) processing time functions,

the class of schedules satisfying the ascending property is dominant, i.e. there always exists an optimal schedule which satisfies the ascending property.

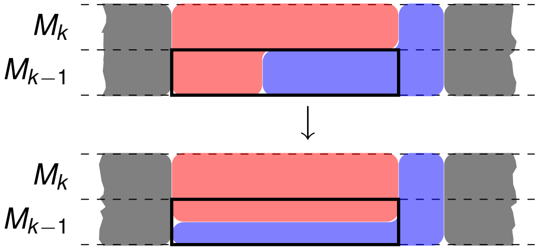
Impact

- Search space reduction for enumeration algorithms.
- Complexity reduction for approximation algorithms.

Contents

“Fractional” case

Easy to prove if jobs can use a fractional number of machines:



Not so easy to prove for the natural (“integer”) case.

Scheme of the proof

Definitions

A piece of job is a non-preemptive part of this job processed on some machine.
A piece of job j is early if it completed strictly before C_j.

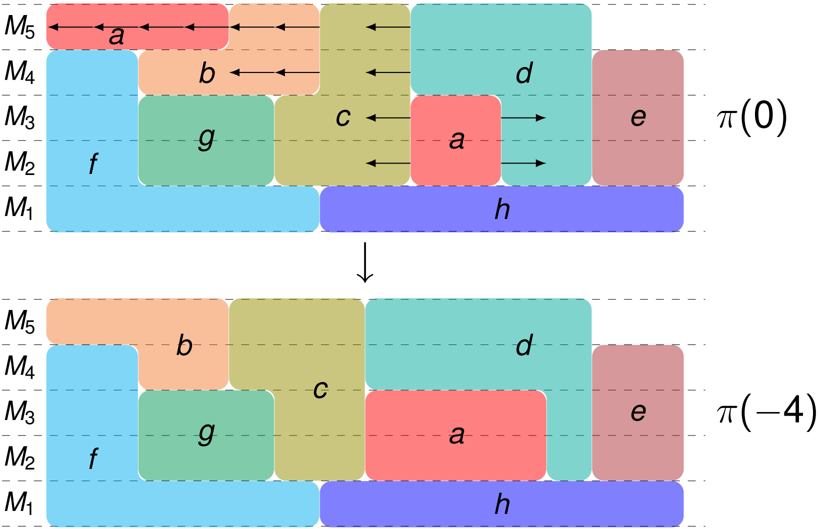
Proof scheme

- Consider an optimal schedule.
- If it does not contain early pieces, it satisfies the ascending property (we are done).
- Otherwise, it is possible to transform it without increasing its cost to another schedule in which the number of early pieces or the total number of pieces is strictly decreased.

Transformation of the schedule with early pieces

- Let piece q of job a be the early piece with the maximum completion time (all pieces completed after C_a^q are not early).
- Let schedule $\pi(\varepsilon)$ be the schedule in which the starting time of every piece u of job j is changed by $\Delta_\varepsilon(S_j^u)$ according to the change of the completion time of the preceding piece, and its completion time — by $\Delta_\varepsilon(C_j^u)$, where
  - $\Delta_\varepsilon(C_q^a) = \varepsilon$,
  - if $C_j^u > C_q^a$, then $\Delta_\varepsilon(C_j^u) = \dfrac{\sum_{k \in K(j)} \Delta_\varepsilon(S_j^k)[-\varepsilon]_{\text{if } j=a}}{|K(j)|}$, where $K(j)$ — set of non-early pieces of job j (the change of starting times of pieces in $K(j)$ is distributed equally among the changes of completion times of these pieces),
  - otherwise $\Delta_\varepsilon(C_j^u) = 0$.

Example of the transformation



Transformation analysis

- $\Delta_\varepsilon(C_j)$ is a linear function of ε as long as non-early pieces remain non-early ($0 > \varepsilon_2 \leq \varepsilon \leq \varepsilon_1 > 0$).
- Schedule $\pi(\varepsilon)$ remains feasible as long as the lengths of all pieces remain non-negative ($0 > \varepsilon_4 \leq \varepsilon \leq \varepsilon_3 > 0$).
- Schedule $\pi(\varepsilon)$ remains feasible as long as, for all pieces u, v of a same job j, if u precedes v in $\pi(0)$, u still precedes v, meaning that the number of simultaneously processed pieces of j does not exceed δ_j ($0 > \varepsilon_6 \leq \varepsilon \leq \varepsilon_5 > 0$).
- As schedule $\pi(0)$ is optimal, the function $\sum_j w_j \Delta_\varepsilon(C_j) = 0$ for ε such that

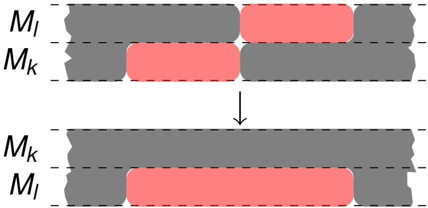
$$0 > \underline{\varepsilon} = \max\{\varepsilon_2, \varepsilon_4, \varepsilon_6\} \leq \varepsilon \leq \min\{\varepsilon_1, \varepsilon_3, \varepsilon_5\} = \overline{\varepsilon} > 0,$$

and all schedules $\pi(\varepsilon)$, $\underline{\varepsilon} \leq \varepsilon \leq \overline{\varepsilon}$, are optimal.

Transformation analysis (2)

Consider schedule $\pi(\underline{\varepsilon})$, $\underline{\varepsilon} = \max\{\varepsilon_2, \varepsilon_4, \varepsilon_6\}$.

- $\underline{\varepsilon} = \varepsilon_2 \Rightarrow$ some early piece becomes non-early.
- $\underline{\varepsilon} = \varepsilon_4 \Rightarrow$ the length of some piece becomes zero (a piece disappears).
- $\underline{\varepsilon} = \varepsilon_6 \Rightarrow C_j^u = S_j^v$ and we can combine two pieces into one.



- End of proof.

Contents

The case in which parallelism costs

The results also holds for the case when, for each job j,

$$\frac{1}{p(q)} - \frac{1}{p(q-1)} \geq \frac{1}{p(q+1)} - \frac{1}{p(q)} \geq 0, \qquad (1)$$

meaning that the processing speed of job j

1. increases when j is passed from q machines to $q+1$,
2. and does not increase more when j is passed from q machines to $q+1$ than when it is passed from $q-1$ machines to q.

Idea of the proof

- The same idea of the proof as in the work preserving case.
- The difference is that $\Delta_\varepsilon(C_j)$ is not a linear function of ε any more, but a concave function.

The case in which parallelism costs (2)

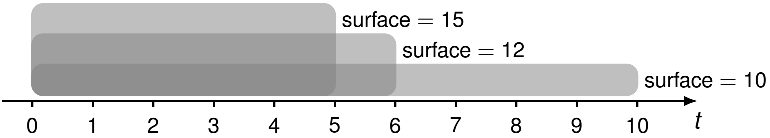
Note that the case (1) “covers” the case in which

$$0 \leq q \cdot p(q) - (q-1) \cdot p(q-1) \leq (q+1) \cdot p(q+1) - q \cdot p(q),$$

meaning that the surface of job j

1. increases when j uses an additional machine,
2. and does not increases less when j is passed from q machines to $q+1$ than when it is passed from $q-1$ machines to q.

Example



Contents

Open problem

Hendel and Kubiak (2008) proposed a polynomial algorithm for the problem

$$P2 \mid var, p_j(q) = p_j/q, \delta_j \mid \sum C_j.$$

The problem

$$P \mid var, p_j(q) = p_j/q, \delta_j \mid \sum C_j$$

remains open, even for the case with 3 machines.