Mapping pipelined applications with replication to increase throughput and reliability

Anne Benoit\textsuperscript{1,2}, Loris Marchal\textsuperscript{2}, Yves Robert\textsuperscript{1,2}, Oliver Sinnen\textsuperscript{3}

1. Institut Universitaire de France  
2. LIP, École Normale Supérieure de Lyon, France  
3. University of Auckland, New Zealand

SBAC-PAD, Petropolis, Rio de Janeiro, Brazil  
October 27-30, 2010
Motivations

- Mapping **pipelined applications** onto **parallel platforms**: practical applications, but **difficult challenge**

- Both **performance** (throughput) and **reliability** objectives: even more difficult!

- Use of **replication**: mapping an application stage onto more than one processor
  - **redundant computations**: increase reliability
  - **round-robin computations** (over consecutive data sets): increase throughput
  - **bi-criteria problem**: need to trade-off between two kinds of replication
Motivations

- Mapping **pipelined applications** onto parallel platforms: practical applications, but **difficult challenge**

- Both **performance** (throughput) and **reliability** objectives: even more difficult!

- Use of replication: mapping an application stage onto more than one processor
  - **redundant computations**: increase reliability
  - **round-robin computations** (over consecutive data sets): increase throughput
  - **bi-criteria problem**: need to trade-off between two kinds of replication
Motivations

- Mapping **pipelined applications** onto **parallel platforms**: practical applications, but **difficult challenge**

- Both **performance** (throughput) and **reliability** objectives: even more difficult!

- Use of **replication**: mapping an application stage onto more than one processor
  - **redundant computations**: increase reliability
  - **round-robin computations** (over consecutive data sets): increase throughput
  - **bi-criteria problem**: need to trade-off between two kinds of replication
Motivations

- Mapping **pipelined applications** onto **parallel platforms**: practical applications, but **difficult challenge**

- Both **performance** (throughput) and **reliability** objectives: even more difficult!

- Use of **replication**: mapping an application stage onto more than one processor
  - **redundant** computations: increase reliability
  - **round-robin** computations (over consecutive data sets): increase throughput
  - **bi-criteria problem**: need to trade-off between two kinds of replication
Motivations

- Mapping **pipelined applications** onto parallel platforms: practical applications, but **difficult challenge**

- Both **performance** (throughput) and **reliability** objectives: even more difficult!

- Use of **replication**: mapping an application stage onto more than one processor
  - **redundant** computations: increase reliability
  - **round-robin** computations (over consecutive data sets): increase throughput
  - **bi-criteria problem**: need to trade-off between two kinds of replication
Motivations

- Mapping **pipelined applications** onto parallel platforms: practical applications, but **difficult challenge**

- Both **performance** (throughput) and **reliability** objectives: even more difficult!

- Use of **replication**: mapping an application stage onto more than one processor
  - **redundant** computations: increase reliability
  - **round-robin** computations (over consecutive data sets): increase throughput
  - **bi-criteria problem**: need to trade-off between two kinds of replication
Main contributions

- **Theoretical side:**
  assess problem hardness with different mapping rules and platform characteristics

- **Practical side:**
  heuristics on most general (NP-complete) case,
  exact algorithm based on A*,
  experiments to assess heuristics performance
Main contributions

• **Theoretical side:**
  assess problem hardness with different mapping rules and platform characteristics

• **Practical side:**
  heuristics on most general (NP-complete) case,
  exact algorithm based on A*,
  experiments to assess heuristics performance
Outline of the talk

1 Framework
   - Application
   - Platform
   - Mapping
   - Objective

2 Complexity results
   - Mono-criterion
   - Bi-criteria
   - Approximation results

3 Practical side
   - Heuristics
   - Optimal algorithm using A*
   - Evaluation results

4 Conclusion
Applicative framework

- Pipeline of $n$ stages $S_1, \ldots, S_n$
- Stage $S_i$ performs a number $w_i$ of computations
- Communication costs are negligible in comparison with computation costs
Platform with $p$ processors $P_1, \ldots, P_p$, fully interconnected as a (virtual) clique

For $1 \leq u \leq p$, processor $P_u$ has speed $s_u$ and failure probability $0 < f_u < 1$

Failure probability: independent of the duration of the application, meant to run for a long time (cycle-stealing scenario)

*SpeedHom* platform: identical speeds $s_u = s$ for $1 \leq u \leq p$ (as opposed to *SpeedHet*)

*FailureHom* platform: identical failure probabilities (as opposed to *FailureHet*)
Platform with $p$ processors $P_1, \ldots, P_p$, fully interconnected as a (virtual) clique

For $1 \leq u \leq p$, processor $P_u$ has speed $s_u$ and failure probability $0 < f_u < 1$

Failure probability: independent of the duration of the application, meant to run for a long time (cycle-stealing scenario)

*SpeedHom* platform: identical speeds $s_u = s$ for $1 \leq u \leq p$ (as opposed to *SpeedHet*)

*FailureHom* platform: identical failure probabilities (as opposed to *FailureHet*)
Interval mapping: consecutive stages mapped together: partition of [1..n] into $m \leq p$ intervals $I_j$

- $I_j$ mapped onto set of processors $A_j$, organized into $\ell_j$ teams
  - processors within a team perform redundant computations (replication for reliability)
  - different teams assigned to same interval execute distinct data sets in a round-robin fashion (replication for performance)

- A processor cannot participate in two different teams

- $\ell = \sum_{j=1}^{m} \ell_j$ is the total number of teams
Mapping problem

- **Interval mapping**: consecutive stages mapped together: partition of $[1..n]$ into $m \leq p$ intervals $I_j$

- $I_j$ mapped onto set of processors $A_j$, organized into $\ell_j$ teams
  - processors within a team perform redundant computations (replication for reliability)
  - different teams assigned to same interval execute distinct data sets in a round-robin fashion (replication for performance)

- A processor cannot participate in two different teams

- $\ell = \sum_{j=1}^{m} \ell_j$ is the total number of teams
Interval mapping: consecutive stages mapped together: partition of \([1..n]\) into \(m \leq p\) intervals \(I_j\)

- \(I_j\) mapped onto set of processors \(A_j\), organized into \(\ell_j\) teams
  - processors within a team perform redundant computations (replication for reliability)
  - different teams assigned to same interval execute distinct data sets in a round-robin fashion (replication for performance)

- A processor cannot participate in two different teams

- \(\ell = \sum_{j=1}^{m} \ell_j\) is the total number of teams
Mapping problem

- **Interval mapping**: consecutive stages mapped together: partition of \([1..n]\) into \(m \leq p\) intervals \(I_j\)

- \(I_j\) mapped onto set of processors \(A_j\), organized into \(\ell_j\) teams
  - processors within a team perform **redundant computations** (replication for reliability)
  - different teams assigned to same interval execute **distinct data sets** in a round-robin fashion (replication for performance)

- A processor cannot participate in two different teams

- \(\ell = \sum_{j=1}^{m} \ell_j\) is the total number of teams
Interval mapping: consecutive stages mapped together: partition of \([1..n]\) into \(m \leq p\) intervals \(I_j\)

- \(I_j\) mapped onto set of processors \(A_j\), organized into \(\ell_j\) teams
  - processors within a team perform redundant computations (replication for reliability)
  - different teams assigned to same interval execute distinct data sets in a round-robin fashion (replication for performance)

- A processor cannot participate in two different teams

\[ \ell = \sum_{j=1}^{m} \ell_j \] is the total number of teams
Mapping problem

- **Interval mapping**: consecutive stages mapped together: partition of $[1..n]$ into $m \leq p$ intervals $I_j$

- $I_j$ mapped onto set of processors $A_j$, organized into $\ell_j$ teams
  - processors within a team perform redundant computations (replication for reliability)
  - different teams assigned to same interval execute distinct data sets in a round-robin fashion (replication for performance)

- A processor cannot participate in two different teams

- $\ell = \sum_{j=1}^{m} \ell_j$ is the total number of teams
Example of mapping

$n = 5$ stages divided into $m = 3$ intervals

$p = 11$ processors organized in $\ell = 6$ teams

$\ell_1 = 3, \ell_2 = 1, \ell_3 = 2$
Example of mapping

\[ n = 5 \text{ stages divided into } m = 3 \text{ intervals} \]
\[ p = 11 \text{ processors organized in } \ell = 6 \text{ teams} \]
\[ \ell_1 = 3, \ell_2 = 1, \ell_3 = 2 \]
Objective functions

- **Period** of the application:

\[ P = \max_{1 \leq j \leq m} \left\{ \sum_{i \in I_j} w_i \frac{\ell_j \times \min_{P_u \in A_j} S_u}{\ell_j} \right\} \]

Round-robin distribution: each team compute one data set every other \( \ell_j \) ones, computation slowed down by slowest processor for interval

- **Failure probability**:

\[ F = 1 - \prod_{1 \leq k \leq \ell} (1 - \prod_{P_u \in T_k} f_u) \]

Computation successful if at least one surviving processor per team
Objective functions

- **Period** of the application:

$$\mathcal{P} = \max_{1 \leq j \leq m} \left\{ \frac{\sum_{i \in I_j} w_i}{\ell_j \times \min_{P_u \in A_j} s_u} \right\}$$

Round-robin distribution: each team compute one data set every other $\ell_j$ ones, computation slowed down by slowest processor for interval

- **Failure probability**:

$$\mathcal{F} = 1 - \prod_{1 \leq k \leq \ell} \left( 1 - \prod_{P_u \in T_k} f_u \right)$$

Computation successful if at least one surviving processor per team
Objective functions

- **Period** of the application:

\[
P = \max_{1 \leq j \leq m} \left\{ \frac{\sum_{i \in I_j} w_i}{\ell_j \times \min_{P_u \in A_j} s_u} \right\}
\]

Round-robin distribution: each team compute one data set every other \( \ell_j \) ones, computation slowed down by slowest processor for interval

- **Failure probability**:

\[
F = 1 - \prod_{1 \leq k \leq \ell} \left( 1 - \prod_{P_u \in T_k} f_u \right)
\]

Computation successful if at least one surviving processor per team
Objective functions

- **Period** of the application:

  \[
  \mathcal{P} = \max_{1 \leq j \leq m} \left\{ \frac{\sum_{i \in I_j} w_i}{\ell_j \times \min_{P_u \in A_j} s_u} \right\}
  \]

  **Round-robin distribution**: each team compute one data set every other \( \ell_j \) ones, computation slowed down by slowest processor for interval

- **Failure probability**:

  \[
  \mathcal{F} = 1 - \prod_{1 \leq k \leq \ell} \left( 1 - \prod_{P_u \in T_k} f_u \right)
  \]

  Computation successful if at least one surviving processor per team
The problem

- Determine the best interval mapping, over all possible partitions into intervals and processor assignments

- Mono-criterion: minimize period or failure probability

- Bi-criteria: (i) given a threshold period, minimize failure probability or (ii) given a threshold failure probability, minimize period
The problem

- Determine the **best interval mapping**, over all possible partitions into intervals and processor assignments.
  - **Mono-criterion**: minimize period or failure probability
  - **Bi-criteria**: (i) given a threshold period, minimize failure probability or (ii) given a threshold failure probability, minimize period
The problem

- Determine the **best interval mapping**, over all possible partitions into intervals and processor assignments

- **Mono-criterion**: minimize period or failure probability

- **Bi-criteria**: (i) given a **threshold period**, minimize failure probability or (ii) given a **threshold failure probability**, minimize period
Outline of the talk

1. Framework
   - Application
   - Platform
   - Mapping
   - Objective

2. Complexity results
   - Mono-criterion
   - Bi-criteria
   - Approximation results

3. Practical side
   - Heuristics
   - Optimal algorithm using A*
   - Evaluation results

4. Conclusion
Mono-criterion complexity results

- **Failure probability**: easy on any kind of platforms: group all stages as a single interval, processed by one single team with all $p$ processors.

- **Period**: one processor per team
  - *SpeedHom* platform: one interval processed by $p$ teams
  - *SpeedHet* platforms: NP-hard in the general case, polynomial if $w_i = w$ for $1 \leq i \leq n$ (see previous work [Algorithmica2010])
Mono-criterion complexity results

- **Failure probability**: easy on any kind of platforms: group all stages as a single interval, processed by one single team with all \( p \) processors.

- **Period**: one processor per team
  - *SpeedHom* platform: one interval processed by \( p \) teams
  - *SpeedHet* platforms: NP-hard in the general case, polynomial if \( w_i = w \) for \( 1 \leq i \leq n \) (see previous work [Algorithmica2010])
Bi-criteria complexity results

- **Preliminary result:** for *SpeedHom* platforms, there exists an optimal bi-criteria mapping with **one single interval**
  - **Proof:** starting from an optimal solution with several intervals, merge intervals, and the single interval is processed by all teams of optimal solution
  - Failure probability remains the same (same teams)
  - New period cannot be greater than optimal period (*SpeedHom* platform)

- Not true on *SpeedHet* platforms:
  - example with $w_1 = s_1 = 1$ and $w_2 = s_2 = 2$, $F^* = 1$
  - period 1 with two intervals
  - period 3/2 with one single interval
Preliminary result: for SpeedHom platforms, there exists an optimal bi-criteria mapping with one single interval

Proof: starting from an optimal solution with several intervals, merge intervals, and the single interval is processed by all teams of optimal solution

- Failure probability remains the same (same teams)
- New period cannot be greater than optimal period (SpeedHom platform)

Not true on SpeedHet platforms: example with \( w_1 = s_1 = 1 \) and \( w_2 = s_2 = 2 \), \( F^* = 1 \)

- period 1 with two intervals
- period 3/2 with one single interval
Bi-criteria complexity results

- **Preliminary result:** for SpeedHom platforms, there exists an optimal bi-criteria mapping with **one single interval**
  - **Proof:** starting from an optimal solution with several intervals, merge intervals, and the single interval is processed by all teams of optimal solution
  - **Failure probability** remains the same (same teams)
  - **New period** cannot be greater than optimal period (SpeedHom platform)

- **Not true on SpeedHet platforms:**
  example with $w_1 = s_1 = 1$ and $w_2 = s_2 = 2$, $F^* = 1$
  - period 1 with two intervals
  - period 3/2 with one single interval
Bi-criteria complexity results

- **Preliminary result**: for *SpeedHom* platforms, there exists an optimal bi-criteria mapping with one single interval
  - *Proof*: starting from an optimal solution with several intervals, merge intervals, and the single interval is processed by all teams of optimal solution
  - Failure probability remains the same (same teams)
  - New period cannot be greater than optimal period (*SpeedHom* platform)

- **Not true on *SpeedHet* platforms**: example with $w_1 = s_1 = 1$ and $w_2 = s_2 = 2$, $F^* = 1$
  - period 1 with two intervals
  - period 3/2 with one single interval
Bi-criteria complexity results

- **Preliminary result:** for *SpeedHom* platforms, there exists an optimal bi-criteria mapping with one single interval
  
  - **Proof:** starting from an optimal solution with several intervals, merge intervals, and the single interval is processed by all teams of optimal solution
  - **Failure probability** remains the same (same teams)
  - **New period** cannot be greater than optimal period (*SpeedHom* platform)

- **Not true on *SpeedHet* platforms:**
  example with $w_1 = s_1 = 1$ and $w_2 = s_2 = 2$, $F^* = 1$
  
  - period 1 with two intervals
  - period 3/2 with one single interval
$\textbf{SpeedHom-FailureHom platforms}$

- $\textbf{SpeedHom-FailureHom}$: Polynomial time algorithm

- **Fixed period $\mathcal{P}^*$**
  - one single interval with minimum number of teams
  
  \[
  \ell_{\text{min}} = \left\lceil \frac{\sum_{i=1}^{n} w_i}{\mathcal{P}^* \times s} \right\rceil
  \]
  
  - greedily assign processors to teams to have balanced teams
  - algorithm in $O(p)$

- **Converse problem: fixed $\mathcal{F}^*$**
  - one single interval...
  - ...but must try all possible number of teams $1 \leq \ell \leq p$
  - algorithm in $O(p \log p)$
**SpeedHom-FailureHom platforms**

- **SpeedHom-FailureHom**: Polynomial time algorithm

- **Fixed period** $\mathcal{P}^*$
  - one single interval with minimum number of teams
    \[
    \ell_{\text{min}} = \left\lceil \frac{\sum_{i=1}^{n} w_i}{\mathcal{P}^* \times s} \right\rceil
    \]
  - greedily assign processors to teams to have balanced teams
  - algorithm in $O(p)$

- **Converse problem**: fixed $\mathcal{F}^*$
  - one single interval...
  - ...but must try all possible number of teams $1 \leq \ell \leq p$
  - algorithm in $O(p \log p)$
**SpeedHom-FailureHom** platforms

- **SpeedHom-FailureHom**: Polynomial time algorithm
- **Fixed period** $\mathcal{P}^*$
  - one single interval with minimum number of teams
  
  \[
  \ell_{\text{min}} = \left\lceil \frac{\sum_{i=1}^{n} W_i}{\mathcal{P}^* \times s} \right\rceil
  \]
  - greedily assign processors to teams to have balanced teams
  - algorithm in $O(p)$

- Converse problem: fixed $\mathcal{F}^*$
  - one single interval...
  - ...but must try all possible number of teams $1 \leq \ell \leq p$
  - algorithm in $O(p \log p)$
- **SpeedHom-FailureHom**: Polynomial time algorithm

- **Fixed period** $\mathcal{P}^*$
  - one single interval with minimum number of teams
  
  \[
  \ell_{\text{min}} = \left\lceil \frac{\sum_{i=1}^{n} W_i}{\mathcal{P}^* \times s} \right\rceil
  \]
  - greedily assign processors to teams to have balanced teams
  - algorithm in $O(p)$

- **Converse problem**: fixed $\mathcal{F}^*$
  - one single interval...
  - ...but must try all possible number of teams $1 \leq \ell \leq p$
  - algorithm in $O(p \log p)$
**SpeedHom-FailureHom** platforms

- **SpeedHom-FailureHom**: Polynomial time algorithm
- **Fixed period \( P^* \)**
  - one single interval with minimum number of teams

\[
\ell_{\text{min}} = \left\lceil \frac{\sum_{i=1}^{n} W_i}{P^* \times s} \right\rceil
\]

- greedily assign processors to teams to have balanced teams
- algorithm in \( O(p) \)

- Converse problem: fixed \( F^* \)
  - one single interval...
  - ...but must try all possible number of teams \( 1 \leq \ell \leq p \)
  - algorithm in \( O(p \log p) \)
With heterogeneous platforms

- \textit{SpeedHet-FailureHom} is \textsc{NP}-hard because \textit{SpeedHet} is \textsc{NP}-hard for period minimization

- \textit{SpeedHom-FailureHet} becomes \textsc{NP}-hard as well: balancing processors within teams is combinatorial; reduction from 3-PARTITION

- \textbf{Intermediate result:} best reliability always obtained by balancing failure probabilities of each team
With heterogeneous platforms

- \textit{SpeedHet-FailureHom} is \textbf{NP-hard}
  because \textit{SpeedHet} is \textbf{NP-hard} for period minimization

- \textit{SpeedHom-FailureHet} becomes \textbf{NP-hard} as well:
  balancing processors within teams is combinatorial;
  reduction from 3-PARTITION

- Intermediate result: best reliability always obtained by
  balancing failure probabilities of each team
With heterogeneous platforms

- \textit{SpeedHet-FailureHom} is NP-hard because \textit{SpeedHet} is NP-hard for period minimization.

- \textit{SpeedHom-FailureHet} becomes NP-hard as well: balancing processors within teams is combinatorial; reduction from 3-PARTITION.

- Intermediate result: best reliability always obtained by balancing failure probabilities of each team.
**Approximation results**

- **SpeedHom**: always optimal with single interval
- **SpeedHet**: period minimization problem (NP-hard)

The optimal single-interval mapping can be found:
- sort processors by non-increasing speeds
- for $1 \leq i \leq p$, compute period using $i$ fastest processors
- time $O(p \log p)$

**Theorem**: single-interval mapping is a $n$-approximation algorithm for period minimization; this factor cannot be improved

**Proof sketch**: start from an optimal solution, with $m \leq n$ intervals, and build a single interval solution, with period $\mathcal{P}_1 \leq m \times \mathcal{P}_m$
Approximation results

- **SpeedHom**: always optimal with single interval
- **SpeedHet**: period minimization problem (NP-hard)

The optimal single-interval mapping can be found:
- sort processors by non-increasing speeds
- for \(1 \leq i \leq p\), compute period using \(i\) fastest processors
- time \(O(p \log p)\)

Theorem: single-interval mapping is a \(n\)-approximation algorithm for period minimization; this factor cannot be improved

Proof sketch: start from an optimal solution, with \(m \leq n\) intervals, and build a single interval solution, with period \(P_1 \leq m \times P_m\)
Approximation results

- **SpeedHom**: always optimal with single interval
- **SpeedHet**: period minimization problem (NP-hard)

- The optimal single-interval mapping can be found:
  - sort processors by non-increasing speeds
  - for \(1 \leq i \leq p\), compute period using \(i\) fastest processors
  - time \(O(p \log p)\)

- **Theorem**: single-interval mapping is a \(n\)-approximation algorithm for period minimization; this factor cannot be improved

  - **Proof sketch**: start from an optimal solution, with \(m \leq n\) intervals, and build a single interval solution, with period
    \(P_1 \leq m \times P_m\)
Approximation results

- **SpeedHom**: always optimal with single interval
- **SpeedHet**: period minimization problem (NP-hard)

The optimal single-interval mapping can be found:
- sort processors by non-increasing speeds
- for $1 \leq i \leq p$, compute period using $i$ fastest processors
- time $O(p \log p)$

**Theorem**: single-interval mapping is a $n$-approximation algorithm for period minimization; this factor cannot be improved

**Proof sketch**: start from an optimal solution, with $m \leq n$ intervals, and build a single interval solution, with period $\mathcal{P}_1 \leq m \times \mathcal{P}_m$
Outline of the talk

1. Framework
   - Application
   - Platform
   - Mapping
   - Objective

2. Complexity results
   - Mono-criterion
   - Bi-criteria
   - Approximation results

3. Practical side
   - Heuristics
   - Optimal algorithm using A*
   - Evaluation results

4. Conclusion
Heuristics

- \textit{SpeedHet-FailureHet} platforms
- Minimize $\mathcal{F}$ under a fixed upper period $\mathcal{P}^*$
- Counterpart problem: binary search over $\mathcal{P}^*$

Two heuristics:
- \textsc{OneInterval}: stages grouped as a single interval (motivated by complexity results)
- \textsc{MultiInterval}: solution with multiple intervals (recall that single interval may be far from optimal)
Heuristics

- **SpeedHet-FailureHet** platforms
- Minimize $\mathcal{F}$ under a fixed upper period $P^*$
- Counterpart problem: binary search over $P^*$

Two heuristics:
- **OneInterval**: stages grouped as a single interval (motivated by complexity results)
- **MultiInterval**: solution with multiple intervals (recall that single interval may be far from optimal)
One single interval

Determine number of teams: try all values $\ell$ between 1 and $p$

For a given $\ell$, discard processors too slow for period

Assign processors to teams to minimize failure probability

- From complexity results: balance reliability across teams
- NP-hard problem but efficient greedy heuristic: sort processors by non-decreasing failure probability and assign next processor to team with highest failure probability

Time complexity: $O(p^2 \log p)$
OneInterval

- One single interval
- Determine number of teams: try all values \( \ell \) between 1 and \( p \)
- For a given \( \ell \), discard processors too slow for period

Assign processors to teams to minimize failure probability

- From complexity results: balance reliability across teams
- NP-hard problem but efficient greedy heuristic: sort processors by non-decreasing failure probability and assign next processor to team with highest failure probability

- Time complexity: \( O(p^2 \log p) \)
One single interval

Determine number of teams: try all values $\ell$ between 1 and $p$

For a given $\ell$, discard processors too slow for period

Assign processors to teams to minimize failure probability

- From complexity results: balance reliability across teams
- NP-hard problem but efficient greedy heuristic: sort processors by non-decreasing failure probability and assign next processor to team with highest failure probability

Time complexity: $O(p^2 \log p)$
MultiInterval

• **Step 1**: create \( \min(n, p) \) intervals (one stage per processor, or balance computational load across intervals)

• **Step 2**: greedily add processors to stages, to minimize maximum ratio of interval computation load to accumulated processor speed

• **Step 3**: for each interval, use **OneInterval** to form teams; use previously unallocated processors (too slow for period); increase bound on period for the interval until valid allocation returned

• **Step 4**: if period bound not achieved for at least one interval, merge interval with largest period with previous or next interval, until bound is achieved

• **Step 5**: merge intervals with highest failure probability as long as it is beneficial

• Note that **OneInterval** is called each time we tentatively merge two intervals (steps 4 and 5)

• Time complexity: \( O(p^3 \log p) \)
MultiInterval

- **Step 1**: create $\min(n, p)$ intervals (one stage per processor, or balance computational load across intervals)

- **Step 2**: greedily add processors to stages, to minimize maximum ratio of interval computation load to accumulated processor speed

- **Step 3**: for each interval, use OneInterval to form teams; use previously unallocated processors (too slow for period); increase bound on period for the interval until valid allocation returned

- **Step 4**: if period bound not achieved for at least one interval, merge interval with largest period with previous or next interval, until bound is achieved

- **Step 5**: merge intervals with highest failure probability as long as it is beneficial

Note that OneInterval is called each time we tentatively merge two intervals (steps 4 and 5)

- Time complexity: $O(p^3 \log p)$
MultiInterval

- **Step 1**: create $\min(n, p)$ intervals (one stage per processor, or balance computational load across intervals)
- **Step 2**: greedily add processors to stages, to minimize maximum ratio of interval computation load to accumulated processor speed
- **Step 3**: for each interval, use \texttt{OneInterval} to form teams; use previously unallocated processors (too slow for period); increase bound on period for the interval until valid allocation returned
- **Step 4**: if period bound not achieved for at least one interval, merge interval with largest period with previous or next interval, until bound is achieved
- **Step 5**: merge intervals with highest failure probability as long as it is beneficial

Note that \texttt{OneInterval} is called each time we tentatively merge two intervals (steps 4 and 5)

- Time complexity: $O(p^3 \log p)$
MultiInterval

- **Step 1**: create $\min(n, p)$ intervals (one stage per processor, or balance computational load across intervals)

- **Step 2**: greedily add processors to stages, to minimize maximum ratio of interval computation load to accumulated processor speed

- **Step 3**: for each interval, use OneInterval to form teams; use previously unallocated processors (too slow for period); increase bound on period for the interval until valid allocation returned

- **Step 4**: if period bound not achieved for at least one interval, merge interval with largest period with previous or next interval, until bound is achieved

- **Step 5**: merge intervals with highest failure probability as long as it is beneficial

  - Note that OneInterval is called each time we tentatively merge two intervals (steps 4 and 5)

- Time complexity: $O(p^3 \log p)$
MultiInterval

- **Step 1**: create \( \min(n, p) \) intervals (one stage per processor, or balance computational load across intervals)

- **Step 2**: greedily add processors to stages, to minimize maximum ratio of interval computation load to accumulated processor speed

- **Step 3**: for each interval, use \texttt{OneInterval} to form teams; use previously unallocated processors (too slow for period); increase bound on period for the interval until valid allocation returned

- **Step 4**: if period bound not achieved for at least one interval, merge interval with largest period with previous or next interval, until bound is achieved

- **Step 5**: merge intervals with highest failure probability as long as it is beneficial

- Note that \texttt{OneInterval} is called each time we tentatively merge two intervals (steps 4 and 5)

- Time complexity: \( O(p^3 \log p) \)
Step 1: create $\min(n, p)$ intervals (one stage per processor, or balance computational load across intervals)

Step 2: greedily add processors to stages, to minimize maximum ratio of interval computation load to accumulated processor speed

Step 3: for each interval, use OneInterval to form teams; use previously unallocated processors (too slow for period); increase bound on period for the interval until valid allocation returned

Step 4: if period bound not achieved for at least one interval, merge interval with largest period with previous or next interval, until bound is achieved

Step 5: merge intervals with highest failure probability as long as it is beneficial

Note that OneInterval is called each time we tentatively merge two intervals (steps 4 and 5)

Time complexity: $O(p^3 \log p)$
**MultiInterval**

- **Step 1**: create $\min(n, p)$ intervals (one stage per processor, or balance computational load across intervals)
- **Step 2**: greedily add processors to stages, to minimize maximum ratio of interval computation load to accumulated processor speed
- **Step 3**: for each interval, use **OneInterval** to form teams; use previously unallocated processors (too slow for period); increase bound on period for the interval until valid allocation returned
- **Step 4**: if period bound not achieved for at least one interval, merge interval with largest period with previous or next interval, until bound is achieved
- **Step 5**: merge intervals with highest failure probability as long as it is beneficial

Note that **OneInterval** is called each time we tentatively merge two intervals (steps 4 and 5)

- Time complexity: $O(p^3 \log p)$
A* algorithm

- A* best-first state space search algorithm for small problem instances

- Non-linearity of failure probability: rules out the use of integer linear programming

- Search space: state $s$ is a partial solution (i.e., partial mapping), with underestimated cost value $c(s)$

- Expansion of a partial solution with lowest $c(s)$ value, with a stage or a processor

- Complete mapping obtained: optimal solution (best-first strategy)
A* algorithm

- **A* best-first state space search algorithm** for small problem instances
- **Non-linearity** of failure probability: rules out the use of integer linear programming

- **Search space**: state $s$ is a partial solution (i.e., partial mapping), with underestimated cost value $c(s)$
- Expansion of a partial solution with lowest $c(s)$ value, with a stage or a processor
- Complete mapping obtained: optimal solution (best-first strategy)
A* algorithm

- **A*** best-first state space search algorithm for small problem instances
- **Non-linearity** of failure probability: rules out the use of **integer linear programming**

- **Search space**: state $s$ is a partial solution (i.e., partial mapping), with underestimated cost value $c(s)$
- Expansion of a partial solution with lowest $c(s)$ value, with a stage or a processor
- Complete mapping obtained: optimal solution (best-first strategy)
A* algorithm

- **A* best-first state space search algorithm** for small problem instances

- **Non-linearity** of failure probability: rules out the use of integer linear programming

- **Search space**: state $s$ is a partial solution (i.e., partial mapping), with underestimated cost value $c(s)$

- Expansion of a partial solution with lowest $c(s)$ value, with a stage or a processor

- Complete mapping obtained: optimal solution (best-first strategy)
A* algorithm

- **A* best-first state space search algorithm** for small problem instances
- **Non-linearity** of failure probability: rules out the use of integer linear programming

**Search space**: state \( s \) is a partial solution (i.e., partial mapping), with underestimated cost value \( c(s) \)

- Expansion of a partial solution with lowest \( c(s) \) value, with a stage or a processor
- Complete mapping obtained: optimal solution (best-first strategy)
State tree for two stages on two processors

Legend

- \([S_a; S_b]\): one interval
- \([P_1, P_2]\): first team for this interval
- \([P_3, P_4]\): second team for this interval
- \(P_5, P_6\): processors not selected for the last interval
- \(\rightarrow\): expansion with a new stage
- \(\rightarrow\rightarrow\rightarrow\): expansion with a new processor
- \(\times\): invalid state
- \(\square\): goal state
Underestimate cost functions

- **Failure probability $\mathcal{F}$**
  - Partial mapping: *adding team* increases failure probability
  - Underestimate: *add remaining processors to existing teams*
  - NP-hard problem: consider *amount of reliability* available and distribute it to the existing teams to *balance* their reliability

- **Period $\mathcal{P}$**
  - Need to check that *partial solution does not exceed the bound*: can be computed exactly
  - Second underestimate: *optimal period achieved by remaining processors on remaining stages*
  - NP-hard problem: consider *perfect load balance*: $\mathcal{P} \leq \frac{\sum w_i}{\sum s_u}$
Underestimate cost functions

- **Failure probability $\mathcal{F}$**
  - Partial mapping: *adding team* increases failure probability
  - Underestimate: *add remaining processors to existing teams*
  - NP-hard problem: consider *amount of reliability* available and distribute it to the existing teams to balance their reliability

- **Period $\mathcal{P}$**
  - Need to check that *partial solution does not exceed the bound*: can be computed exactly
  - Second underestimate: *optimal period achieved by remaining processors on remaining stages*
  - NP-hard problem: consider *perfect load balance*: $\mathcal{P} \leq \frac{\sum w_i}{\sum s_u}$
Heuristics vs A*

- Randomly generated workload scenarios
- Both heuristics close to optimal solution
- OneInterval is better than MultiInterval in a few cases
- A* much slower, but main limitation is memory
Performance of heuristics

- Distribution of ratio between failure probability obtained by a heuristic (OneInterval in red, MultiInterval in blue) and optimal failure probability (A*) (optimal: ratio 1)
- On average, heuristics 20% above optimal
- Ratio 10: cases in which heuristics find no solution ($\approx$ 10%)
Larger scenarios

- **OneInterval** better in 61% of the cases
- **MultiInterval** better in 20% of the cases

On average, failure probability of **OneInterval** 2% above **MultiInterval**

Comparison of **OneInterval** with optimal single-interval solution (easy to compute with A*): in average, 0.05% above optimal, and 5% in the worst case
Larger scenarios

- **OneInterval** better in 61% of the cases
- **MultiInterval** better in 20% of the cases

On average, failure probability of **OneInterval** 2% above **MultiInterval**

Comparison of **OneInterval** with optimal single-interval solution (easy to compute with A*): in average, 0.05% above optimal, and 5% in the worst case
Larger scenarios

- **OneInterval** better in 61% of the cases
- **MultiInterval** better in 20% of the cases
- On average, failure probability of **OneInterval** 2% above **MultiInterval**
- Comparison of **OneInterval** with optimal single-interval solution (easy to compute with A*): in average, 0.05% above optimal, and 5% in the worst case
Outline of the talk

1. Framework
   - Application
   - Platform
   - Mapping
   - Objective

2. Complexity results
   - Mono-criterion
   - Bi-criteria
   - Approximation results

3. Practical side
   - Heuristics
   - Optimal algorithm using A*
   - Evaluation results

4. Conclusion
Conclusion and future work

- **Exhaustive complexity study**
  - polynomial time algorithm for *SpeedHom-FailureHom* platforms
  - NP-completeness with one level of heterogeneity
  - approximation results to compare single interval solution with any other solution

- **Practical solution to the problem**
  - efficient heuristics (inspired by theoretical study) for *SpeedHet-FailureHet* platforms
  - A* algorithm with non-trivial underestimate functions
  - experimental results: very good behaviour of heuristics

- **Future work**
  - further approximation results
  - enhanced multiple interval heuristics
  - improved A* techniques
Conclusion and future work

- **Exhaustive complexity study**
  - polynomial time algorithm for $SpeedHom$-$FailureHom$ platforms
  - NP-completeness with one level of heterogeneity
  - approximation results to compare single interval solution with any other solution

- **Practical solution to the problem**
  - efficient heuristics (inspired by theoretical study) for $SpeedHet$-$FailureHet$ platforms
  - A* algorithm with non-trivial underestimate functions
  - experimental results: very good behaviour of heuristics

- **Future work**
  - further approximation results
  - enhanced multiple interval heuristics
  - improved A* techniques
Conclusion and future work

- Exhaustive complexity study
  - polynomial time algorithm for SpeedHom-FailureHom platforms
  - NP-completeness with one level of heterogeneity
  - approximation results to compare single interval solution with any other solution

- Practical solution to the problem
  - efficient heuristics (inspired by theoretical study) for SpeedHet-FailureHet platforms
  - A* algorithm with non-trivial underestimate functions
  - experimental results: very good behaviour of heuristics

- Future work
  - further approximation results
  - enhanced multiple interval heuristics
  - improved A* techniques