Performance and energy optimization of concurrent pipelined applications

Anne Benoit, Paul Renaud-Goud and Yves Robert

Institut Universitaire de France

ROMA team, LIP
École Normale Supérieure de Lyon, France

New Challenges in Scheduling Theory, Frejus
September 12-17, 2010
Motivations

- Mapping **concurrent pipelined applications** onto **distributed platforms**: practical applications, but difficult problems
- Assess problem hardness ⇒ different mapping rules and platform characteristics
- Energy saving is becoming a crucial problem
- Several **concurrent objective functions**: period, latency, power
  ⇒ **Multi-criteria approach**: minimize power consumption while guaranteeing some performance
- Exhaustive complexity study
- Heuristics on most general (NP-complete) case
Motivations

- Mapping **concurrent pipelined applications** onto **distributed platforms**: practical applications, but difficult problems
- Assess problem hardness ⇒ different mapping rules and platform characteristics

- **Energy saving** is becoming a crucial problem
  - Several **concurrent objective functions**: period, latency, power
  - ⇒ Multi-criteria approach: minimize power consumption while guaranteeing some performance

- Exhaustive complexity study
- Heuristics on most general (NP-complete) case
Motivations

- Mapping **concurrent pipelined applications** onto **distributed platforms**: practical applications, but difficult problems
- Assess problem hardness \( \Rightarrow \) different mapping rules and platform characteristics

- **Energy saving** is becoming a crucial problem

- Several **concurrent objective functions**: period, latency, power
  \( \Rightarrow \) Multi-criteria approach: minimize power consumption while guaranteeing some performance

- Exhaustive complexity study
- Heuristics on most general (NP-complete) case
Motivations

- Mapping concurrent pipelined applications onto distributed platforms: practical applications, but difficult problems
- Assess problem hardness ⇒ different mapping rules and platform characteristics
- Energy saving is becoming a crucial problem
- Several concurrent objective functions: period, latency, power
  ⇒ Multi-criteria approach: minimize power consumption while guaranteeing some performance
- Exhaustive complexity study
- Heuristics on most general (NP-complete) case
Minimizing total energy consumed by processors: very important objective (economic and environmental reasons)


Algorithmic techniques:
- Shut down idle processors
- Dynamic speed scaling

The higher the speed, the higher the power consumption

\[ \text{Power} = f \times V^2, \text{ and } V \text{ (voltage) increases with } f \text{ (frequency)} \]

Speed \( s \):
\[ P(s) = s^\alpha + P_{\text{static}}, \text{ with } 2 \leq \alpha \leq 3 \]

Problem: decide which processors to enroll, and at which speed to run them
Why bother with energy?

- Minimizing total energy consumed by processors: very important objective (economic and environmental reasons)

Algorithmic techniques:
- Shut down idle processors
- Dynamic speed scaling

- The higher the speed, the higher the power consumption
- $Power = f \times V^2$, and $V$ (voltage) increases with $f$ (frequency)
- Speed $s$: $P(s) = s^\alpha + P_{static}$, with $2 \leq \alpha \leq 3$

Problem: decide which processors to enroll, and at which speed to run them
Why bother with energy?

- Minimizing total energy consumed by processors: very important objective (economic and environmental reasons)

Algorithmic techniques:
- Shut down idle processors
- **Dynamic speed scaling**: processors can run at variable speed, e.g., Intel XScale, Intel Speed Step, AMD PowerNow
  - The higher the speed, the higher the power consumption
  - $Power = f \times V^2$, and $V$ (voltage) increases with $f$ (frequency)
  - Speed $s$: $P(s) = s^\alpha + P_{static}$, with $2 \leq \alpha \leq 3$

- Problem: decide which processors to enroll, and at which speed to run them
Why bother with energy?

- Minimizing total energy consumed by processors: very important objective (economic and environmental reasons)

**Algorithmic techniques:**
- Shut down idle processors
- **Dynamic speed scaling:** processors can run at variable speed, e.g., Intel XScale, Intel Speed Step, AMD PowerNow
- The higher the speed, the higher the power consumption
- \( \text{Power} = f \times V^2 \), and \( V \) (voltage) increases with \( f \) (frequency)
- Speed \( s \): \( P(s) = s^\alpha + P_{\text{static}} \), with \( 2 \leq \alpha \leq 3 \)

- Problem: decide which processors to enroll, and at which speed to run them
Why bother with energy?

- Minimizing total energy consumed by processors: very important objective (economic and environmental reasons)

Algorithmic techniques:
- Shut down idle processors
- **Dynamic speed scaling**: processors can run at variable speed, e.g., Intel XScale, Intel Speed Step, AMD PowerNow
- The higher the speed, the higher the power consumption
- \( \text{Power} = f \times V^2 \), and \( V \) (voltage) increases with \( f \) (frequency)
- Speed \( s \): \( P(s) = s^\alpha + P_{\text{static}} \), with \( 2 \leq \alpha \leq 3 \)

- Problem: decide which processors to enroll, and at which speed to run them
Motivating example

- Period: $T = 3$
- Latency: $L = 8$
Motivating example

- Period: $T = 3$
- Latency: $L = 8$
Motivating example

\[ P = 3^3 + 8^3 = 539 \]

- Period: \( T = 3 \)
- Latency: \( L = 8 \)
Motivating example

\[ P = 3^3 + 8^3 = 539 \]

- **Period:** \( T = 3 \)
- **Latency:** \( L = 8 \)
Motivating example

\[ P = 3^3 + 8^3 = 539 \]

- Period: \( T = 3 \)
- Latency: \( L = 8 \)
Motivating example

\[ P = 3^3 + 8^3 = 539 \]

- **Period:** \( T = 3 \)
- **Latency:** \( L = 8 \)
Motivating example

\[ P = 3^3 + 8^3 = 539 \]

- Period: \( T = 3 \)
- Latency: \( L = 8 \)
Motivating example

\[ P = 3^3 + 8^3 = 539 \]

- Period: \( T = 3 \)
- Latency: \( L = 8 \)
Motivating example

\[ P = 3^3 + 8^3 = 539 \]

- Period: \( T = 3 \)
- Latency: \( L = 8 \)
Motivating example

\[ P = 3^3 + 8^3 = 539 \]

- Period: \( T = 3 \)
- Latency: \( L = 8 \)
Motivating example

\[ P = 3^3 + 8^3 = 539 \]
Motivating example

\[ P = 3^3 + 8^3 = 539 \]
Motivating example

\[ P = 3^3 + 8^3 = 539 \]

Period: \( T = 3 \)

Latency: \( L = 8 \)
Motivating example

\[ P = 3^3 + 8^3 = 539 \]

- Period: \( T = 3 \)
- Latency: \( L = 8 \)
Motivating example

\[ P = 3^3 + 8^3 = 539 \]

\( \mathcal{P}_1 \)

\( \mathcal{P}_2 \)

- Period: \( T = 3 \)
- Latency: \( L = 8 \)
Motivating example

\[ P = 3^3 + 8^3 = 539 \]

Period: \( T = 3 \)
Latency: \( L = 8 \)
Motivating example

$P = 3^3 + 8^3 = 539$

- **Period**: $T = 3$
- **Latency**: $L = 8$
Motivating example

\[ P = 3^3 + 8^3 = 539 \]

- Period: \( T = 3 \)
- Latency: \( L = 8 \)
Motivating example

\[ P = 539 \]
\[ P = 8 \]

- **Period:** \( T = 3 \)
- **Latency:** \( L = 8 \)
Motivating example

\[ P = 539 \quad P = 8 \]

- Period: \( T = 3 \quad T = 15 \)
- Latency: \( L = 8 \)
Motivating example

\[ P = 539 \]
\[ P = 8 \]

\( \mathcal{P}_1 \)

\( \mathcal{P}_2 \)

- Period: \( T = 3 \), \( T = 15 \)
- Latency: \( L = 8 \), \( L = 17 \)
Outline of the talk

1 **Framework**
   - Application and platform
   - Mapping rules
   - Metrics

2 **Complexity results**
   - Mono-criterion problems
   - Bi-criteria problems
   - Tri-criteria problems
   - With resource sharing

3 **Experiments**
   - Heuristics
   - Experiments
   - Summary

4 **Conclusion**
Outline of the talk

1 Framework
   - Application and platform
   - Mapping rules
   - Metrics

2 Complexity results
   - Mono-criterion problems
   - Bi-criteria problems
   - Tri-criteria problems
   - With resource sharing

3 Experiments
   - Heuristics
   - Experiments
   - Summary

4 Conclusion
Concurrent pipelined applications

- $w^i_a$: weight of stage $S^i_a$ ($i^{th}$ stage of application $a$)
- $\delta^i_a$: size of outcoming data of $S^i_a$

Processors with multiple speeds (or modes): $\{s_{u,1}, \ldots, s_{u,m_u}\}$

Constant speed during the execution

Platform fully interconnected;

- $b_{u,v}$: bandwidth between processors $\mathcal{P}_u$ and $\mathcal{P}_v$;
- overlap or non-overlap of communications and computations

Three platform types:

- Fully homogeneous, or speed homogeneous
- Communication homogeneous, or speed heterogeneous
- Fully heterogeneous
Application model and execution platform

- **Concurrent pipelined applications**
  - $w^i_a$: weight of stage $S^i_a$ ($i^{th}$ stage of application $a$)
  - $\delta^i_a$: size of outcoming data of $S^i_a$

- Processors with **multiple speeds** (or modes): $\{s_{u,1}, \ldots, s_{u,m_u}\}$
  Constant speed during the execution

- **Platform** fully interconnected;
  $b_{u,v}$: bandwidth between processors $P_u$ and $P_v$;
  overlap or non-overlap of communications and computations

- **Three platform types:**
  - Fully homogeneous, or speed homogeneous
  - Communication homogeneous, or speed heterogeneous
  - Fully heterogeneous
Application model and execution platform

- **Concurrent pipelined applications**
  - $w^i_a$: weight of stage $S^i_a$ ($i^{th}$ stage of application $a$)
  - $\delta^i_a$: size of outcoming data of $S^i_a$

- Processors with **multiple speeds** (or modes): \{s_u,1, \ldots, s_u,m_u\}
  - Constant speed during the execution

- **Platform** fully interconnected;
  - $b_{u,v}$: bandwidth between processors $\mathcal{P}_u$ and $\mathcal{P}_v$;

- **Three platform types**:
  - Fully homogeneous, or speed homogeneous
  - Communication homogeneous, or speed heterogeneous
  - Fully heterogeneous
Mapping rules

- Mapping with no processor sharing: relevant in practice (security rules)
  - One-to-one mapping

![Diagram of one-to-one mapping]

- Interval mapping

![Diagram of interval mapping]

- General mapping with resource sharing: better resource utilization

![Diagram of general mapping]
Mapping rules

- Mapping with **no processor sharing**: relevant in practice (security rules)
  - One-to-one mapping

- Interval mapping

- General mapping **with resource sharing**: better resource utilization
Mapping rules

- Mapping with **no processor sharing**: relevant in practice (security rules)
  - One-to-one mapping

  \[\text{App}_1 \rightarrow \text{App}_2\]

- Interval mapping

  \[\text{App}_1 \rightarrow \text{App}_2\]

- General mapping **with resource sharing**: better resource utilization

  \[\text{App}_1 \rightarrow \text{App}_2\]
Interval mapping on a single application with no resource sharing; $k$ intervals $I_j$ of stages from $S^{d_j}$ to $S^{e_j}$

- **Period $T$** of an application: minimum delay between the processing of two consecutive data sets
  \[
  T^{(overlap)} = \max_{j \in \{1, \ldots, k\}} \left( \max \left( \frac{\delta^{d_j} - 1}{b_{\text{alloc}(d_j - 1), \text{alloc}(d_j)}}, \frac{\sum_{i=d_j}^{e_j} w^i}{s_{\text{alloc}(d_j)}}, \frac{\delta^{e_j}}{b_{\text{alloc}(d_j), \text{alloc}(e_j + 1)}} \right) \right)
  \]

- **Latency $L$** of an application: time, for a data set, to go through the whole pipeline
  \[
  L = \frac{\delta^0}{b_{\text{alloc}(0), \text{alloc}(1)}} + \sum_{j=1}^{m} \left( \sum_{i=d_j}^{e_j} \frac{w^i}{s_{\text{alloc}(d_j)}} + \frac{\delta^{e_j}}{b_{\text{alloc}(d_j), \text{alloc}(e_j + 1)}} \right)
  \]

- **Power $P$** of the platform: sum of power of processors
  \[
  P = \sum_{\mathcal{P}_u} P(u), \quad P(u) = P_{\text{dyn}}(s_u) + P_{\text{stat}}(u), \quad P_{\text{dyn}}(s_u) = s_u^\alpha, \quad 2 \leq \alpha \leq 3
  \]
Metrics without resource sharing

Interval mapping on a single application with no resource sharing; $k$ intervals $I_j$ of stages from $S_{dj}$ to $S_{ej}$

- **Period** $T$ of an application: minimum delay between the processing of two consecutive data sets

$$T^{(overlap)} = \max_{j \in \{1, \ldots, k\}} \left( \max \left( \frac{\delta_{dj} - 1}{b_{\text{alloc}(dj - 1), \text{alloc}(dj)}}, \frac{\sum_{i = dj}^{ej} w_i}{s_{\text{alloc}(dj)}}, \frac{\delta_{ej}}{b_{\text{alloc}(dj), \text{alloc}(ej + 1)}} \right) \right)$$

- **Latency** $L$ of an application: time, for a data set, to go through the whole pipeline

$$L = \frac{\delta_0}{b_{\text{alloc}(0), \text{alloc}(1)}} + \sum_{j=1}^{m} \left( \sum_{i = dj}^{ej} \frac{w_i}{s_{\text{alloc}(dj)}} + \frac{\delta_{ej}}{b_{\text{alloc}(dj), \text{alloc}(ej + 1)}} \right)$$

- **Power** $P$ of the platform: sum of power of processors

$$P = \sum_{P_u} P(u), \quad P(u) = P_{\text{dyn}}(s_u) + P_{\text{stat}}(u), \quad P_{\text{dyn}}(s_u) = s_u^\alpha, \quad 2 \leq \alpha \leq 3$$
Interval mapping on a single application with no resource sharing; $k$ intervals $I_j$ of stages from $S_{d_j}$ to $S_{e_j}$

- **Period** $T$ of an application: minimum delay between the processing of two consecutive data sets

$$T^{(overlap)} = \max_{j \in \{1, \ldots, k\}} \left( \max \left( \frac{\delta_{d_j} - 1}{b_{ alloc(d_j-1), alloc(d_j) }}, \frac{\sum_{i=d_j}^{e_j} w_i}{s_{ alloc(d_j) }}, \frac{\delta_{e_j}}{b_{ alloc(d_j), alloc(e_j+1) }} \right) \right)$$

- **Latency** $L$ of an application: time, for a data set, to go through the whole pipeline

$$L = \frac{\delta_0}{b_{ alloc(0), alloc(1) }} + \sum_{j=1}^{m} \left( \sum_{i=d_j}^{e_j} \frac{w_i}{s_{ alloc(d_j) }} + \frac{\delta_{e_j}}{b_{ alloc(d_j), alloc(e_j+1) }} \right)$$

- **Power** $P$ of the platform: sum of power of processors

$$P = \sum_{P_u} P(u), \quad P(u) = P_{dyn}(s_u) + P_{stat}(u), \quad P_{dyn}(s_u) = s_u^\alpha, \quad 2 \leq \alpha \leq 3$$
Metrics without resource sharing

Interval mapping on a single application with no resource sharing; \( k \) intervals \( I_j \) of stages from \( S_{d_j} \) to \( S_{e_j} \)

- **Period** \( T \) of an application: minimum delay between the processing of two consecutive data sets

\[
T^{(overlap)} = \max_{j \in \{1, \ldots, k\}} \left( \max \left( \frac{\delta_{d_j} - 1}{b_{\text{alloc}(d_j), \text{alloc}(d_j)}}, \frac{\sum_{i=d_j}^{e_j} w_i}{s_{\text{alloc}(d_j)}}, \frac{\delta_{e_j}}{b_{\text{alloc}(d_j), \text{alloc}(e_j+1)}} \right) \right)
\]

- **Latency** \( L \) of an application: time, for a data set, to go through the whole pipeline

\[
L = \frac{\delta_0}{b_{\text{alloc}(0), \text{alloc}(1)}} + \sum_{j=1}^{m} \left( \frac{\sum_{i=d_j}^{e_j} w_i}{s_{\text{alloc}(d_j)}} + \frac{\delta_{e_j}}{b_{\text{alloc}(d_j), \text{alloc}(e_j+1)}} \right)
\]

- **Power** \( P \) of the platform: sum of power of processors

\[
P = \sum_{P_u} P(u), \quad P(u) = P_{\text{dyn}}(s_u) + P_{\text{stat}}(u), \quad P_{\text{dyn}}(s_u) = s_u^\alpha, \quad 2 \leq \alpha \leq 3
\]
With classical latency definition, **NP-completeness of the execution scheduling**, given a mapping with a period/latency objective

⇒ for general mappings, **latency model of Özgüner**:

\[ L = (2m - 1)T \]

where \( m - 1 \) is the number of processor changes, and \( T \) the period of the application.

Period given ⇒ bound on number of processor changes

Given an application, we can check if the mapping is valid, given a bound on period and latency per application:

- For period, check that each processor can handle its load computation and meet some communication constraints
- For latency, check the number of processor changes
With classical latency definition, **NP-completeness of the execution scheduling**, given a mapping with a period/latency objective

⇒ for general mappings, **latency model of Özuguner**:  
\[ L = (2m - 1)T \], where \( m - 1 \) is the number of processor changes, and \( T \) the period of the application

Period given ⇒ bound on number of processor changes

Given an application, we can check if the mapping is valid, given a bound on period and latency per application:

- For period, check that each processor can handle its load computation and meet some communication constraints
- For latency, check the number of processor changes
With classical latency definition, **NP-completeness of the execution scheduling**, given a mapping with a period/latency objective

⇒ for general mappings, **latency model of Özgüner**: 
\[ L = (2m - 1)T \]
where \( m - 1 \) is the number of processor changes, and \( T \) the period of the application

Period given ⇒ bound on number of processor changes

Given an application, we can **check if the mapping is valid**, given a bound on period and latency per application:

- For **period**, check that each processor can handle its load computation and meet some communication constraints
- For **latency**, check the number of processor changes
With classical latency definition, **NP-completeness of the execution scheduling**, given a mapping with a period/latency objective

⇒ for general mappings, **latency model of Özgüner**: 
\[ L = (2m - 1)T, \]  
where \( m - 1 \) is the number of processor changes, and \( T \) the period of the application

\[ L = 7 \times T \]

Period given ⇒ bound on number of processor changes

Given an application, we can **check if the mapping is valid**, given a bound on period and latency per application:

- For **period**, check that each processor can handle its load computation and meet some communication constraints
- For **latency**, check the number of processor changes
With classical latency definition, **NP-completeness of the execution scheduling**, given a mapping with a period/latency objective

⇒ for general mappings, **latency model of Özgüner**: 

\[ L = (2m - 1)T \]

where \( m - 1 \) is the number of processor changes, and \( T \) the period of the application.

\[ L = 7 \times T \]

Period given ⇒ bound on number of processor changes

Given an application, we can check if the mapping is valid, given a bound on period and latency per application:

- For **period**, check that each processor can handle its load, computation and meet some communication constraints
- For **latency**, check the number of processor changes
Metrics with resource sharing

With classical latency definition, **NP-completeness of the execution scheduling**, given a mapping with a period/latency objective

⇒ for general mappings, latency model of Önzüner:

\[ L = (2m - 1)T \]

where \( m - 1 \) is the number of processor changes, and \( T \) the period of the application

![Diagram showing processor changes](image)

\[ L = 7 \times T \]

Period given ⇒ bound on number of processor changes

Given an application, we can **check if the mapping is valid**, given a bound on period and latency per application:

- For **period**, check that each processor can handle its load computation and meet some communication constraints
- For **latency**, check the number of processor changes
Optimization problems

- **Minimizing one criterion:**
  - Period or latency: minimize $\max_a W_a \times T_a$ or $\max_a W_a \times L_a$
  - Power: minimize $P = \sum_u P(u)$

- **Fixing one criterion:**
  - Fix the period or latency of each application
    $\rightarrow$ fix an array of periods or latencies
  - Fix a bound on total power consumption $P$

- **Multi-criteria approach:** minimizing one criterion, fixing the other ones

- Energy criterion = power consumption, i.e., energy per time unit $\Rightarrow$ combination power/period
Optimization problems

- **Minimizing one criterion:**
  - Period or latency: minimize $\max_a W_a \times T_a$ or $\max_a W_a \times L_a$
  - Power: minimize $P = \sum_u P(u)$

- **Fixing one criterion:**
  - Fix the period or latency of each application
    $\rightarrow$ fix an array of periods or latencies
  - Fix a bound on total power consumption $P$

- **Multi-criteria approach:** minimizing one criterion, fixing the other ones

- Energy criterion $=$ power consumption, i.e., energy per time unit $\Rightarrow$ combination power/period
Optimization problems

- Minimizing **one criterion**:
  - Period or latency: minimize $\max_a W_a \times T_a$ or $\max_a W_a \times L_a$
  - Power: minimize $P = \sum_u P(u)$

- **Fixing one criterion**:
  - Fix the period or latency of each application
    $\rightarrow$ fix an array of periods or latencies
  - Fix a bound on total power consumption $P$

- **Multi-criteria approach**: minimizing one criterion, fixing the other ones

- Energy criterion $=$ power consumption, i.e., energy per time unit $\Rightarrow$ combination power/period
Outline of the talk

1 Framework
   - Application and platform
   - Mapping rules
   - Metrics

2 Complexity results
   - Mono-criterion problems
   - Bi-criteria problems
   - Tri-criteria problems
   - With resource sharing

3 Experiments
   - Heuristics
   - Experiments
   - Summary

4 Conclusion
## Mono-criterion complexity results

### Period minimization:

<table>
<thead>
<tr>
<th></th>
<th>proc-hom</th>
<th>special-app(^1)</th>
<th>proc-het</th>
<th>com-hom</th>
<th>com-het</th>
</tr>
</thead>
<tbody>
<tr>
<td>one-to-one</td>
<td></td>
<td>polynomial (binary search)</td>
<td></td>
<td></td>
<td>NP-complete</td>
</tr>
<tr>
<td>interval</td>
<td></td>
<td>polynomial</td>
<td></td>
<td>NP-complete</td>
<td>NP-complete</td>
</tr>
</tbody>
</table>

\(^1\)special-app: com-hom & pipe-hom

### Latency minimization:

<table>
<thead>
<tr>
<th></th>
<th>proc-hom</th>
<th>special-app(^1)</th>
<th>proc-het</th>
<th>com-hom</th>
<th>com-het</th>
</tr>
</thead>
<tbody>
<tr>
<td>one-to-one</td>
<td></td>
<td>polynomial</td>
<td></td>
<td>NP-complete</td>
<td>NP-complete</td>
</tr>
<tr>
<td>interval</td>
<td></td>
<td>polynomial (binary search)</td>
<td></td>
<td>NP-complete</td>
<td></td>
</tr>
</tbody>
</table>

Anne.Benoit@ens-lyon.fr  September 14, 2010
### Mono-criterion complexity results

#### Period minimization:

<table>
<thead>
<tr>
<th></th>
<th>proc-hom</th>
<th>special-app&lt;sup&gt;1&lt;/sup&gt;</th>
<th>proc-het</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>com-hom</td>
<td>com-het</td>
<td>com-het</td>
</tr>
<tr>
<td>one-to-one</td>
<td>polynomial</td>
<td>[binary search]</td>
<td>NP-complete</td>
</tr>
<tr>
<td>interval</td>
<td>polynomial</td>
<td>NP-complete</td>
<td>NP-complete</td>
</tr>
</tbody>
</table>

#### Latency minimization:

<table>
<thead>
<tr>
<th></th>
<th>proc-hom</th>
<th>special-app&lt;sup&gt;1&lt;/sup&gt;</th>
<th>proc-het</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>com-hom</td>
<td>com-het</td>
<td>com-het</td>
</tr>
<tr>
<td>one-to-one</td>
<td>polynomial</td>
<td>NP-complete</td>
<td>NP-complete</td>
</tr>
<tr>
<td>interval</td>
<td>polynomial</td>
<td>[binary search]</td>
<td>NP-complete</td>
</tr>
</tbody>
</table>

<sup>1</sup>special-app: com-hom & pipe-hom
Latency minimization (1)

- Problem: one-to-one mapping - many applications - heterogeneous platform - no communication - homogeneous pipelines - minimize $\max_a L_a$

- Single application: greedy polynomial algorithm

- Many applications: reduction from 3-PARTITION

- 3-PARTITION:
  - Input: $3m + 1$ integers $a_1, a_2, \ldots, a_{3m}$ and $B$ such that $\sum_i a_i = mB$
  - Does there exist a partition $I_1, \ldots, I_m$ of $\{1, \ldots, 3m\}$ such that for all $j \in \{1, \ldots, m\}$, $|I_j| = 3$ and $\sum_{i \in I_j} a_i = B$?
Latency minimization (2)

- **3-PARTITION**: renumbering of the $a_i$ such that:

  \[
  \begin{align*}
  a_{1,1} + a_{1,2} + a_{1,3} &= B \\
  a_{2,1} + a_{2,2} + a_{2,3} &= B \\
  \vdots \\
  a_{m,1} + a_{m,2} + a_{m,3} &= B
  \end{align*}
  \]

- **Reduction**: Can we obtain a latency $L^0 \leq B$?

- **Equivalence of problems**

$3m$ heterogeneous unimodal processors
## Bi-criteria complexity results

### Period/latency minimization:

<table>
<thead>
<tr>
<th></th>
<th>proc-hom</th>
<th>special-app</th>
<th>proc-het</th>
</tr>
</thead>
<tbody>
<tr>
<td>com-hom</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>one-to-one or interval</td>
<td>polynomial</td>
<td></td>
<td>NP-complete</td>
</tr>
</tbody>
</table>

### Power/period minimization:

<table>
<thead>
<tr>
<th></th>
<th>proc-hom</th>
<th>special-app</th>
<th>proc-het</th>
<th>com-het</th>
</tr>
</thead>
<tbody>
<tr>
<td>com-hom</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>one-to-one</td>
<td>polynomial (minimum matching)</td>
<td>NP-complete</td>
<td></td>
<td></td>
</tr>
<tr>
<td>interval</td>
<td>polynomial</td>
<td></td>
<td>NP-complete</td>
<td></td>
</tr>
</tbody>
</table>
## Bi-criteria complexity results

### Period/latency minimization:

<table>
<thead>
<tr>
<th></th>
<th>proc-hom</th>
<th>special-app</th>
<th>proc-het</th>
<th>com-hom</th>
<th>com-het</th>
</tr>
</thead>
<tbody>
<tr>
<td>one-to-one or interval</td>
<td>polynomial</td>
<td></td>
<td></td>
<td></td>
<td>NP-complete</td>
</tr>
</tbody>
</table>

### Power/period minimization:

<table>
<thead>
<tr>
<th></th>
<th>proc-hom</th>
<th>special-app</th>
<th>proc-het</th>
<th>com-hom</th>
<th>com-het</th>
</tr>
</thead>
<tbody>
<tr>
<td>one-to-one interval</td>
<td>polynomial</td>
<td>(minimum matching)</td>
<td>NP-complete</td>
<td></td>
<td></td>
</tr>
<tr>
<td>interval</td>
<td>polynomial</td>
<td></td>
<td>NP-complete</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Problem: one-to-one mapping - many applications - communication homogeneous platform - power minimization for a given array of periods

Minimum weighted matching of a bipartite graph

weight: power of the minimum mode of $\mathcal{P}_u$ which runs $S_i$ within the period

$p \geq N$
### Bi-criteria complexity results

#### Period/latency minimization:

<table>
<thead>
<tr>
<th></th>
<th>proc-hom</th>
<th>special-app</th>
<th>proc-het</th>
</tr>
</thead>
<tbody>
<tr>
<td>com-hom</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>one-to-one or interval</td>
<td>polynomial</td>
<td></td>
<td>NP-complete</td>
</tr>
</tbody>
</table>

#### Power/period minimization:

|                     | proc-hom | special-app | proc-het | com-het |
|---------------------|----------|-------------|----------|
| com-hom             |          |             |          |
| one-to-one          | polynomial (minimum matching) |           | NP-complete |
| interval            | polynomial |             | NP-complete |
Single application (1)

- Problem: interval mapping - single application - fully homogeneous platform - power minimization for a given period
- $P(i, j, k)$: minimum power to run stages $S^i$ to $S^j$ using exactly $k$ processors $\rightarrow$ looking for $\min_{1 \leq k \leq p} P(1, n, k)$
- Recurrence relation:

\[
P(i, j, k) = \min_{1 \leq \ell \leq j-1} (P(i, \ell, k - 1) + P(\ell + 1, j, 1))
\]
Single application (2)

- $P(i, i, q) = +\infty$ if $q > 1$

- $\mathcal{F}^j_i$: possible powers of a processor running the stages $S^i$ to $S^j$, fulfilling the period constraint

  $$
  \mathcal{F}^j_i = \left\{ P_{\text{dyn}}(s_\ell) + P_{\text{stat}}, \max \left( \frac{\delta^{i-1}}{b}, \frac{\sum_{k=i}^j w^k}{s_\ell}, \frac{\delta^j}{b} \right) \leq T, \ell \in \{1, \ldots, m\} \right\}
  $$

- $P(i, j, 1) = \left\{ \begin{array}{ll}
  \min \mathcal{F}^j_i & \text{if } \mathcal{F}^j_i \neq \emptyset \\
  +\infty & \text{otherwise}
  \end{array} \right.$
Many applications (1)

- Problem: interval mapping - fully homogeneous platform - power minimization for given periods by application

- $P_a^q$: minimum power consumed by $q$ processors so that the period constraint on the application $a$ is met, found by the previous dynamic programming

- $P(a, k)$: minimum power consumed by $k$ processors on the applications $1, \ldots, a$, unknown

- Initialization: $\forall k \in \{1, \ldots, p\}$ \hspace{1cm} $P(1, k) = P_1^k$
Many applications (2)

- Recurrence: $P(a, k) = \min_{1 \leq q < k} \left( P(a - 1, k - q) + P^q_a \right)$
## Tri-criteria complexity results

<table>
<thead>
<tr>
<th>proc-hom</th>
<th>special-app</th>
<th>proc-het</th>
</tr>
</thead>
<tbody>
<tr>
<td>com-hom</td>
<td>com-hom</td>
<td>com-het</td>
</tr>
</tbody>
</table>

**Reduction from 2-PARTITION**

(Instance of 2-PARTITION: \( a_1, a_2, \ldots, a_n \) with \( \sigma = \sum_{i=1}^{n} a_i \))
Problem instance

One-to-one mapping - fully homogeneous platform

\[
\begin{align*}
P^0 &= P^* + \alpha X(\sigma/2 + 1/2), \\
L^0 &= L^* - X(\sigma/2 - 1/2), \\
T^0 &= L^0
\end{align*}
\]

where \( P^* \) and \( L^* \) are power and latency when each \( S_i \) is run at speed \( s_{2i-1} \)
Main ideas

- K big enough and X small enough so that the stage $S_i$ must be processed at speed $s_{2i-1}$ or $s_{2i}$

- For a subset $\mathcal{I}$ of $\{1, \ldots, n\}$, if ($S_i$ is run at speed $s_{2i} \iff i \in \mathcal{I}$),

  $$P = P^* + \sum_{i \in \mathcal{I}} (\alpha a_i X + o(X)) \quad , \quad L = L^* - \sum_{i \in \mathcal{I}} (a_i X - o(X))$$

- Recall:

  $$P^0 = P^* + \alpha X (\sigma/2 + 1/2) \quad , \quad L^0 = L^* - X (\sigma/2 - 1/2)$$
And for general mappings with resource sharing?

- Exhaustive complexity study with no resource sharing: new polynomial algorithms for multiple applications and results of NP-completeness

- With the simplified latency model, tri-criteria polynomial dynamic programming algorithm with no resource sharing and speed-homogeneous platforms

- With resource sharing or speed-heterogeneous platforms, all problem instances are NP-hard, even for only period minimization
And for general mappings with resource sharing?

- Exhaustive complexity study with no resource sharing: new polynomial algorithms for multiple applications and results of NP-completeness

- With the simplified latency model, tri-criteria polynomial dynamic programming algorithm with no resource sharing and speed-homogeneous platforms

- With resource sharing or speed-heterogeneous platforms, all problem instances are NP-hard, even for only period minimization
Exhaustive complexity study with no resource sharing: new polynomial algorithms for multiple applications and results of NP-completeness

With the simplified latency model, tri-criteria polynomial dynamic programming algorithm with no resource sharing and speed-homogeneous platforms

With resource sharing or speed-heterogeneous platforms, all problem instances are NP-hard, even for only period minimization
Outline of the talk

1 Framework
   • Application and platform
   • Mapping rules
   • Metrics

2 Complexity results
   • Mono-criterion problems
   • Bi-criteria problems
   • Tri-criteria problems
   • With resource sharing

3 Experiments
   • Heuristics
   • Experiments
   • Summary

4 Conclusion
Heuristics

Tri-criteria problem: power consumption minimization given a bound on period and latency per application, on speed heterogeneous platform.

Each heuristic (except H2) exists in two variants: interval mapping without resource sharing and general mapping with resource sharing in order to evaluate the impact of processor reuse.

Latency model of Özgüner: \( L = (2m - 1)T \)

- H1: random cuts
- H2: one entire application per processor (assignment problem)
- H2-split: interval splitting
- H3: two-step heuristic: choose a speed distribution and find a valid mapping (variants on both steps)
Fix processor speeds

\[ \mathcal{P}_1 \quad \mathcal{P}_2 \quad \mathcal{P}_3 \]
Mapping heuristic: find a valid mapping
Mapping heuristic: find a valid mapping

\[ \mathcal{P}_1 \quad \mathcal{P}_2 \quad \mathcal{P}_3 \]

\[ \text{Check} \]

\[ \begin{align*}
\text{App}_1 & \rightarrow \quad \text{App}_2 \rightarrow \quad \text{App}_3 \\
\end{align*} \]
Mapping heuristic: find a valid mapping

\[ \mathcal{P}_1 \quad \mathcal{P}_2 \quad \mathcal{P}_3 \]

\[ \text{KO} \]

\[ \text{App}_1 \rightarrow \text{App}_2 \rightarrow \text{App}_3 \]
Iterate the process: increase processor speeds
Iterate the process: increase processor speeds
Iterate the process: increase processor speeds
Iterate the process: increase processor speeds
Iterate the process: increase processor speeds
Iterate the process: increase processor speeds

\[ \text{Check} \]

- \(\mathcal{P}_1\)
- \(\mathcal{P}_2\)
- \(\mathcal{P}_3\)

\[ App_1 \rightarrow \]
\[ App_2 \rightarrow \]
\[ App_3 \rightarrow \]
Iterate the process: increase processor speeds
Iterate the process: increase processor speeds
Iterate the process: increase processor speeds
Iterate the process: increase processor speeds
Iterate the process: increase processor speeds
Experimental plan

- **Integer linear program** to assess the absolute performance of the heuristics on small instances

- **Small instances**: two or three applications, around 15 stages per application, around 8 processors

- Execution time on 30 small instances: less than one second for all heuristics, one week for the ILP

- Each heuristic and the ILP: variant without sharing ("-n") and variant with sharing ("-r")
  - General behavior of heuristics
  - Impact of resource sharing
  - Scalability of heuristics
Experimental plan

- **Integer linear program** to assess the absolute performance of the heuristics on small instances

- **Small instances**: two or three applications, around 15 stages per application, around 8 processors

- Execution time on 30 small instances: less than one second for all heuristics, one week for the ILP

- Each heuristic and the ILP: variant without sharing ("-n") and variant with sharing ("-r")

  - General behavior of heuristics
  - Impact of resource sharing
  - Scalability of heuristics
Experimental plan

- **Integer linear program** to assess the absolute performance of the heuristics on small instances

- **Small instances**: two or three applications, around 15 stages per application, around 8 processors

- Execution time on 30 small instances: less than one second for all heuristics, one week for the ILP

- Each heuristic and the ILP: variant without sharing ("-n") and variant with sharing ("-r")
  - General behavior of heuristics
  - Impact of resource sharing
  - Scalability of heuristics
Increasing latency

![Graph showing energy optimization under increasing latency]

- **cplex-r**
- **H1-r**
- **H2**
- **H2-split-r**
- **H3-upDown-r**
- **H3-speed-r**
- **H3-energy-r**
- **best**
Impact of static power

![Graph showing the impact of static power on different algorithms and heuristics.](image-url)
Impact of mode distribution

![Impact of mode distribution graph]

- $1/Energy$
- $s_{u,l+1} - s_{u,l}$
- cplex-n
- H1-n
- H1-r
- H2
- H2-split-n
- H2-split-r
- H3-upDown-n
- H3-upDown-r

Anne.Benoit@ens-lyon.fr  September 14, 2010
Scalability

The diagram illustrates the energy consumption (y-axis) against the number of applications (nbApp, x-axis) for different heuristics. The heuristics include H1-r, H2, H2-split-r, H3-upDown-r, H3-speed-r, and H3-energy-r. The 'best' line represents the optimal energy consumption across all heuristics. As the number of applications increases, the energy consumption for all heuristics increases, with H3-energy-r showing the highest energy consumption compared to the others.
Summary of experiments

- **Efficient heuristics**: best heuristic always at 90% of the optimal solution on small instances
- Supremacy of H2-split-r, better in average, and gets even better when problem instances get larger
- H3 has smaller execution time (one second versus three minutes for 20 applications), ILP not usable in practice
- **Resource sharing** becomes crucial with important static power (use fewer processors) or with distant modes (better use of all available speed)
Summary of experiments

- **Efficient heuristics**: best heuristic always at 90% of the optimal solution on small instances
- Supremacy of H2-split-r, better in average, and gets even better when problem instances get larger
- H3 has smaller execution time (one second versus three minutes for 20 applications), ILP not usable in practice
- **Resource sharing** becomes crucial with important static power (use fewer processors) or with distant modes (better use of all available speed)
Outline of the talk

1 Framework
   - Application and platform
   - Mapping rules
   - Metrics

2 Complexity results
   - Mono-criterion problems
   - Bi-criteria problems
   - Tri-criteria problems
   - With resource sharing

3 Experiments
   - Heuristics
   - Experiments
   - Summary

4 Conclusion
Conclusion and future work

- **Exhaustive complexity study**
  - new polynomial algorithms
  - new NP-completeness proofs
  - impact of model on complexity (tri-criteria homogeneous)

- **Experimental study**
  - efficient heuristics
  - impact of resource reuse

- **Current/future work**
  - continuous speeds
  - approximation algorithms
Conclusion and future work

- **Exhaustive complexity study**
  - new polynomial algorithms
  - new NP-completeness proofs
  - impact of model on complexity (tri-criteria homogeneous)

- **Experimental study**
  - efficient heuristics
  - impact of resource reuse

- **Current/future work**
  - continuous speeds
  - approximation algorithms
Conclusion and future work

- **Exhaustive complexity study**
  - new polynomial algorithms
  - new NP-completeness proofs
  - impact of model on complexity (tri-criteria homogeneous)

- **Experimental study**
  - efficient heuristics
  - impact of resource reuse

- **Current/future work**
  - continuous speeds
  - approximation algorithms