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

GT-ETIC: Groupe de travail consommation d'énergie dans les TIC, 21 Juin 2011

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

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 $\Rightarrow$ 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

Why bother with energy?

- Minimizing total energy consumed by processors: very important objective (economic and environmental reasons)
- M. P. Mills, The internet begins with coal, Environment and Climate News (1999)
- 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)
- M. P. Mills, The internet begins with coal, Environment and Climate News (1999)
- 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)
- M. P. Mills, The internet begins with coal, Environment and Climate News (1999)
- 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

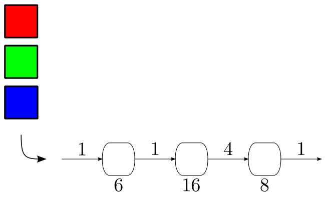
Why bother with energy?

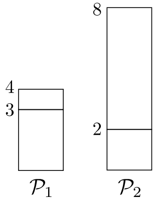
- Minimizing total energy consumed by processors: very important objective (economic and environmental reasons)
- M. P. Mills, The internet begins with coal, Environment and Climate News (1999)
- Algorithmic techniques:
  - Shut down idle processors
  - Dynamic speed scaling: processors can run at variable 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

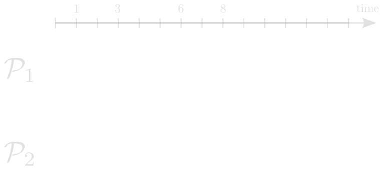
Why bother with energy?

- Minimizing total energy consumed by processors: very important objective (economic and environmental reasons)
- M. P. Mills, The internet begins with coal, Environment and Climate News (1999)
- 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

Motivating example

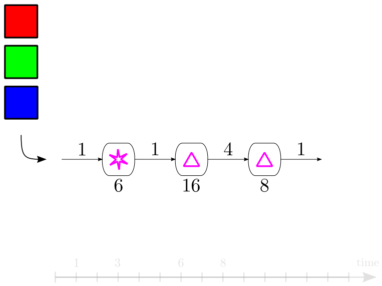






- Period: $T = 3$
- Latency: $L = 8$

Motivating example

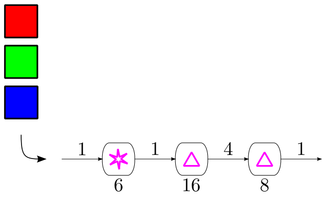


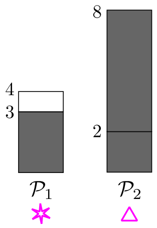
$\mathcal{P}_1$

$\mathcal{P}_2$

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

Motivating example





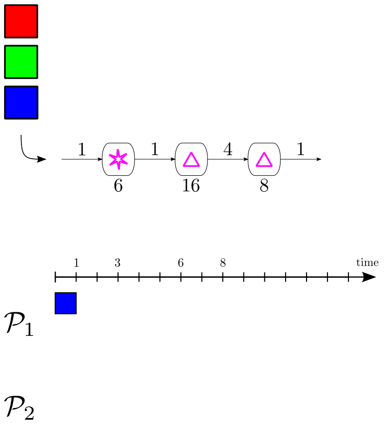
$$P = 3^3 + 8^3$$
$$= 539$$

$\mathcal{P}_1$

$\mathcal{P}_2$

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

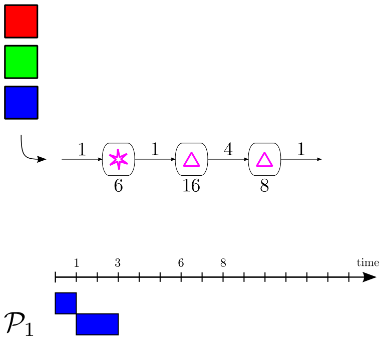
Motivating example



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

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

Motivating example

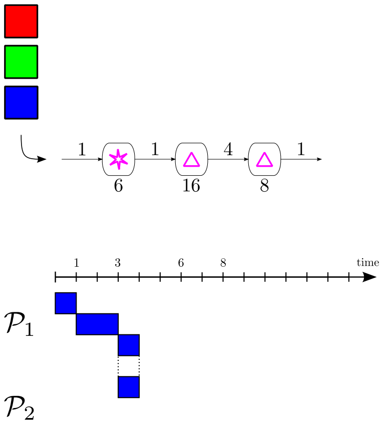


$$P = 3^3 + 8^3$$
$$= 539$$

$\mathcal{P}_2$

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

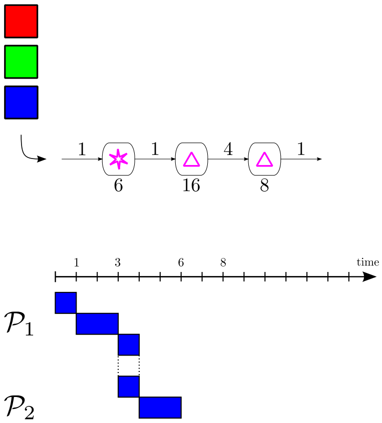
Motivating example



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

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

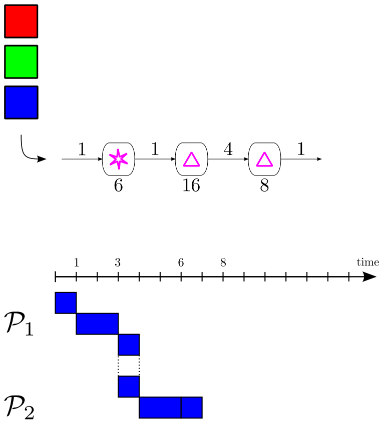
Motivating example



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

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

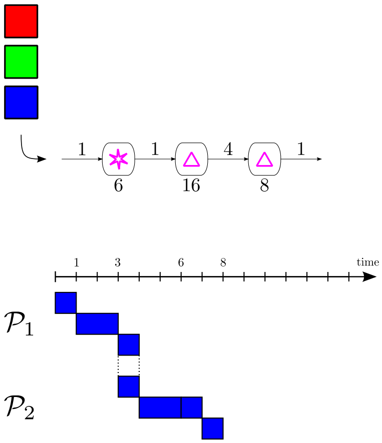
Motivating example



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

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

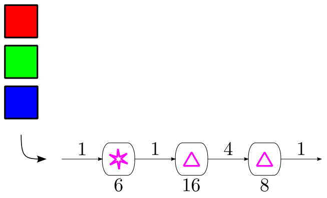
Motivating example

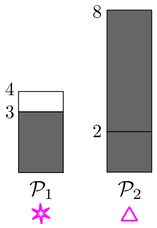


$$P = 3^3 + 8^3$$
$$= 539$$

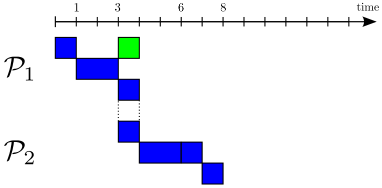
- Period: $T = 3$
- Latency: $L = 8$

Motivating example



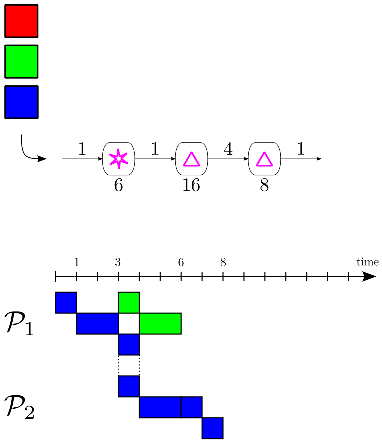


$$P = 3^3 + 8^3$$
$$= 539$$



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

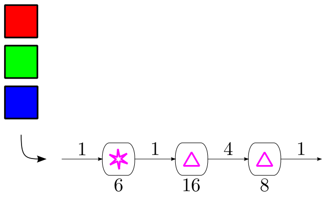
Motivating example

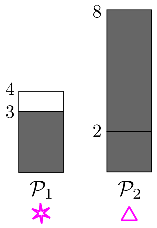


$$P = 3^3 + 8^3$$
$$= 539$$

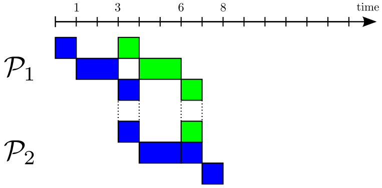
- Period: $T = 3$
- Latency: $L = 8$

Motivating example



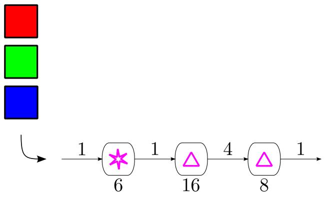


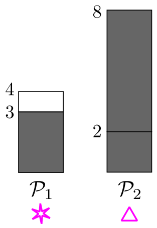
$$P = 3^3 + 8^3$$
$$= 539$$



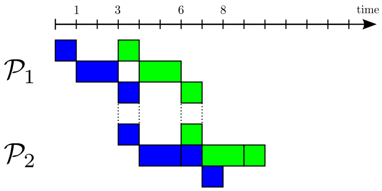
- Period: $T = 3$
- Latency: $L = 8$

Motivating example



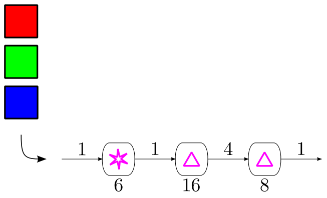


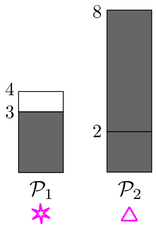
$$P = 3^3 + 8^3$$
$$= 539$$



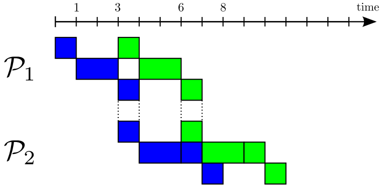
- Period: $T = 3$
- Latency: $L = 8$

Motivating example



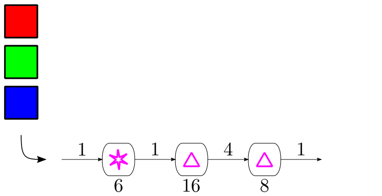


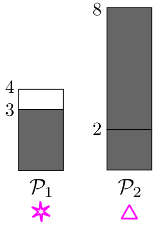
$$P = 3^3 + 8^3$$
$$= 539$$



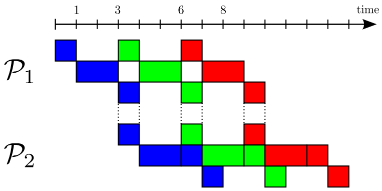
- Period: $T = 3$
- Latency: $L = 8$

Motivating example



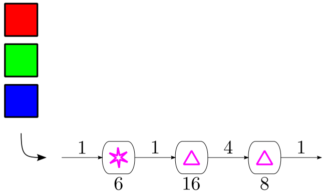


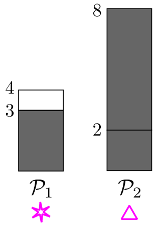
$$P = 3^3 + 8^3$$
$$= 539$$



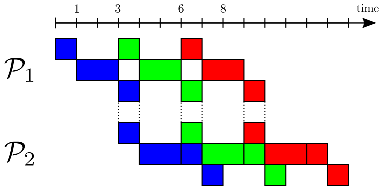
- Period: $T = 3$
- Latency: $L = 8$

Motivating example



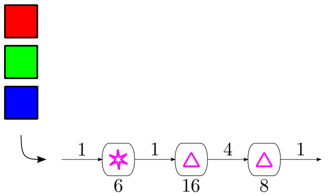


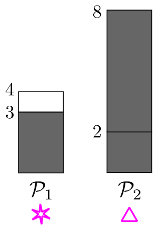
$$P = 3^3 + 8^3$$
$$= 539$$



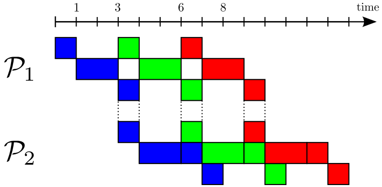
- Period: $T = 3$
- Latency: $L = 8$

Motivating example



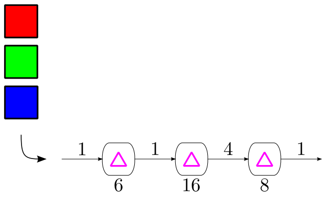


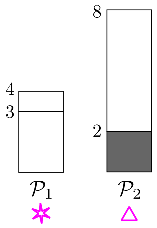
$$P = 3^3 + 8^3$$
$$= 539$$



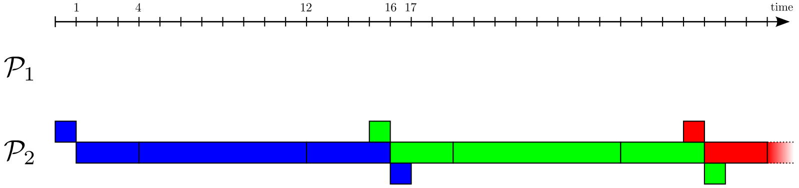
- Period: $T = 3$
- Latency: $L = 8$

Motivating example



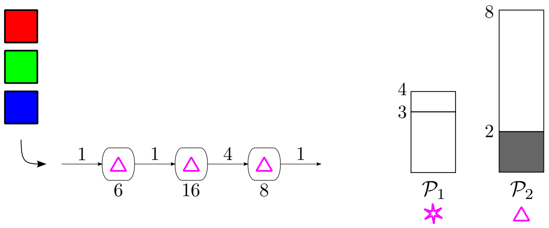


$$P = 3^3 + 8^3$$
$$= 539$$



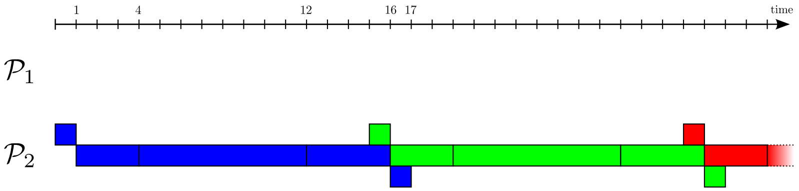
- Period: $T = 3$
- Latency: $L = 8$

Motivating example



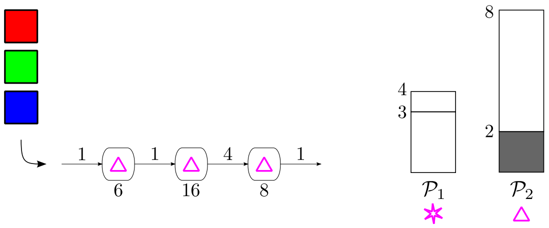
~~$P = 539$~~

$P = 8$

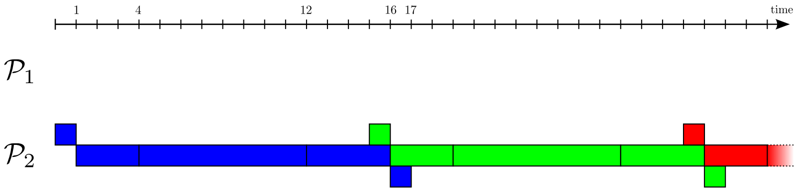


- Period: $T = 3$
- Latency: $L = 8$

Motivating example

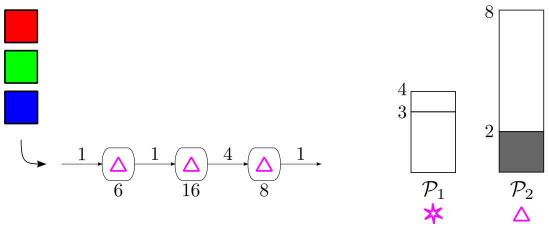


~~$P = 539$~~
$P = 8$

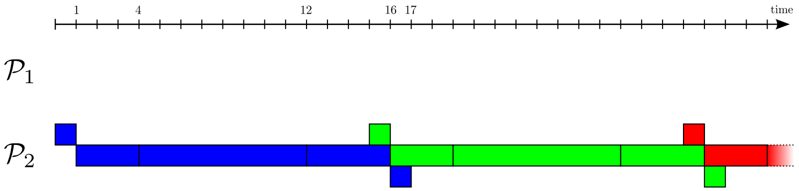


- Period: ~~$T = 3$~~ $T = 15$
- Latency: $L = 8$

Motivating example



~~$P = 539$~~
$P = 8$



- Period: ~~$T = 3$~~ $T = 15$
- Latency: ~~$L = 8$~~ $L = 17$

Outline of the talk

Outline of the talk

Application model and execution platform

- Concurrent pipelined applications
  - w_a^i: weight of stage $\mathcal{S}_a^i$ (i^{th} stage of application a)
  - δ_a^i: size of outcoming data of $\mathcal{S}_a^i$
- 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

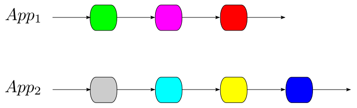
- Concurrent pipelined applications
  - w_a^i: weight of stage $\mathcal{S}_a^i$ (i^{th} stage of application a)
  - δ_a^i: size of outcoming data of $\mathcal{S}_a^i$
- Processors with multiple speeds (or modes): $\{s_{u,1}, \dots, s_{u,m_u}\}$
  Constant speed during the execution
- Platform fully 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

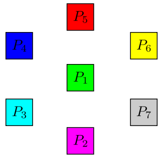
Application model and execution platform

- Concurrent pipelined applications
  - w_a^i: weight of stage $\mathcal{S}_a^i$ (i^{th} stage of application a)
  - δ_a^i: size of outcoming data of $\mathcal{S}_a^i$
- Processors with multiple speeds (or modes): $\{s_{u,1}, \dots, s_{u,m_u}\}$
  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

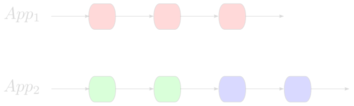
Mapping rules

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

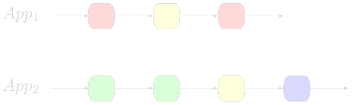




  - Interval mapping

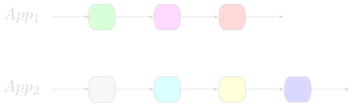


- General mapping with resource sharing: better resource utilization

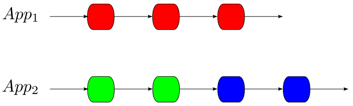


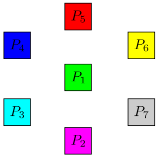
Mapping rules

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

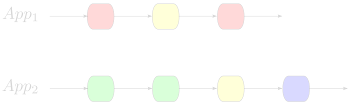


  - Interval mapping



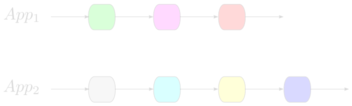


- General mapping with resource sharing: better resource utilization

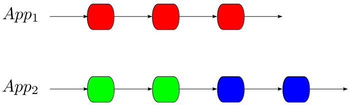


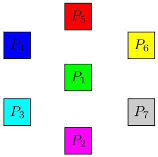
Mapping rules

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

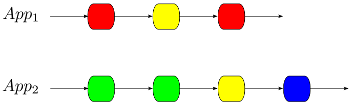


  - Interval mapping





- General mapping with resource sharing: better resource utilization



Metrics without resource sharing

Interval mapping on a single application with no resource sharing: k intervals I_j of stages from $\mathcal{S}^{d_j}$ to $\mathcal{S}^{e_j}$

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

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

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

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

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

$$P = \sum_{\mathcal{P}_u} P(u), \quad P(u) = P_{dyn}(s_u) + P_{stat}(u), \quad P_{dyn}(s_u) = s_u^{\alpha}, \quad 2 \le \alpha \le 3$$

Metrics without resource sharing

Interval mapping on a single application with no resource sharing: k intervals I_j of stages from $\mathcal{S}^{d_j}$ to $\mathcal{S}^{e_j}$

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

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

- 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

$$P = \sum_{\mathcal{P}_u} P(u), \quad P(u) = P_{dyn}(s_u) + P_{stat}(u), \quad P_{dyn}(s_u) = s_u^{\alpha}, \quad 2 \leq \alpha \leq 3$$

Metrics without resource sharing

Interval mapping on a single application with no resource sharing: k intervals I_j of stages from $\mathcal{S}^{d_j}$ to $\mathcal{S}^{e_j}$

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

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

- 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

$$P = \sum_{\mathcal{P}_u} P(u), \quad P(u) = P_{dyn}(s_u) + P_{stat}(u), \quad P_{dyn}(s_u) = s_u^{\alpha}, \quad 2 \le \alpha \le 3$$

Metrics without resource sharing

Interval mapping on a single application with no resource sharing: k intervals I_j of stages from $\mathcal{S}^{d_j}$ to $\mathcal{S}^{e_j}$

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

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

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

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

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

$$P = \sum_{\mathcal{P}_u} P(u), \quad P(u) = P_{dyn}(s_u) + P_{stat}(u), \quad P_{dyn}(s_u) = s_u^{\alpha}, \quad 2 \le \alpha \le 3$$

Metrics with resource sharing

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

$\Rightarrow$ 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 $\Rightarrow$ bound on number of processor changes

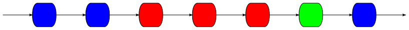
Given an application, we can check if the mapping is valid, given a bound on period and latency per application:

- 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

$\Rightarrow$ 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 $\Rightarrow$ bound on number of processor changes

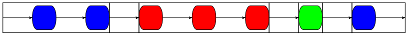
Given an application, we can check if the mapping is valid, given a bound on period and latency per application:

- 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

$\Rightarrow$ 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 $\Rightarrow$ bound on number of processor changes

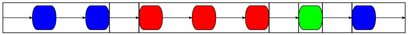
Given an application, we can check if the mapping is valid, given a bound on period and latency per application:

- 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

$\Rightarrow$ 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 $\Rightarrow$ bound on number of processor changes

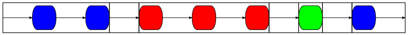
Given an application, we can check if the mapping is valid, given a bound on period and latency per application:

- 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

$\Rightarrow$ 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 $\Rightarrow$ bound on number of processor changes

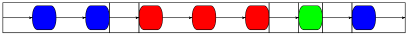
Given an application, we can check if the mapping is valid, given a bound on period and latency per application:

- 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

$\Rightarrow$ 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 $\Rightarrow$ 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

Mono-criterion complexity results

Period minimization:

	proc-hom	proc-het		
	com-hom	special-app[1]	com-hom	com-het
one-to-one	polynomial (binary search)			NP-complete
interval	polynomial	NP-complete	NP-complete	

Latency minimization:

	proc-hom	proc-het		
	com-hom	special-app[1]	com-hom	com-het
one-to-one	polynomial	NP-complete		NP-complete
interval	polynomial (binary search)			NP-complete

[1]special-app: com-hom & pipe-hom

Mono-criterion complexity results

Period minimization:

	proc-hom	proc-het		
	com-hom	special-app[1]	com-hom	com-het
one-to-one	polynomial (binary search)			NP-complete
interval	polynomial	NP-complete	NP-complete	

Latency minimization:

	proc-hom	proc-het		
	com-hom	special-app[1]	com-hom	com-het
one-to-one	polynomial	NP-complete		NP-complete
interval	polynomial (binary search)			NP-complete

[1] special-app: com-hom & pipe-hom

Latency minimization

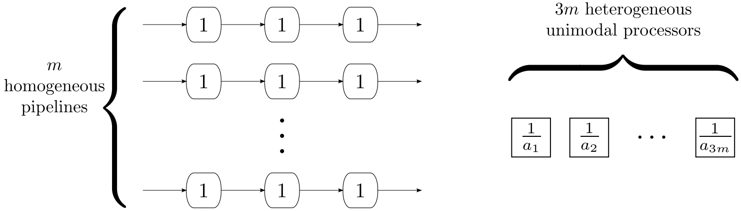
- Single application: greedy polynomial algorithm
- Many applications: NP-hard; reduction from 3-PARTITION
  - Input: $3m+1$ integers $a_1, a_2, \ldots, a_{3m}$ and B such that $\sum_i a_i = mB$
  - Does there exist a partition $I_1, \ldots, I_m$ of $\{1, \ldots, 3m\}$ such that for all $j \in \{1, \ldots, m\}$, $|I_j| = 3$ and $\sum_{i \in I_j} a_i = B$?

Latency minimization

- Single application: greedy polynomial algorithm
- Many applications: NP-hard; reduction from 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

- Single application: greedy polynomial algorithm
- Many applications: NP-hard; reduction from 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 $\sum_{i \in I_j} a_i = B$?
- Reduction:


m homogeneous pipelines
1
$3m$ heterogeneous unimodal processors
$\frac{1}{a_1}$
$\frac{1}{a_2}$
$\ldots$
$\frac{1}{a_{3m}}$

Can we obtain a latency $L^0 \leq B$?

- Equivalence of problems

Bi-criteria complexity results

Period/latency minimization:

	proc-hom	proc-het		
	com-hom	special-app	com-hom	com-het
one-to-one or interval	polynomial	NP-complete		

Power/period minimization:

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

Bi-criteria complexity results

Period/latency minimization:

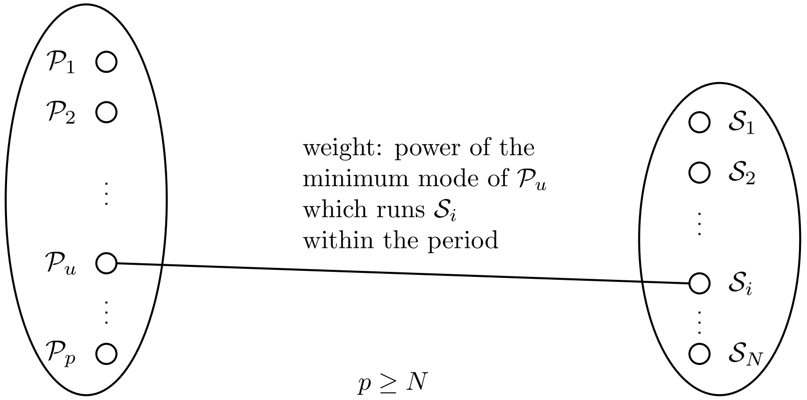
	proc-hom	proc-het		
	com-hom	special-app	com-hom	com-het
one-to-one or interval	polynomial	NP-complete		

Power/period minimization:

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

Power/period minimization

- Minimum weighted matching of a bipartite graph:



Bi-criteria complexity results

Period/latency minimization:

	proc-hom	proc-het		
	com-hom	special-app	com-hom	com-het
one-to-one or interval	polynomial	NP-complete		

Power/period minimization:

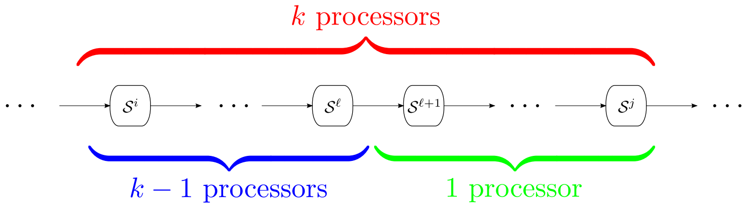
	proc-hom	proc-het		
	com-hom	special-app	com-hom	com-het
one-to-one	polynomial (minimum matching)			NP-complete
interval	polynomial	NP-complete		

Single application

- For one application, $P_1(i,j,k)$: minimum power to run stages $\mathcal{S}^i$ to $\mathcal{S}^j$ using exactly k processors
- Solution to the single application problem:

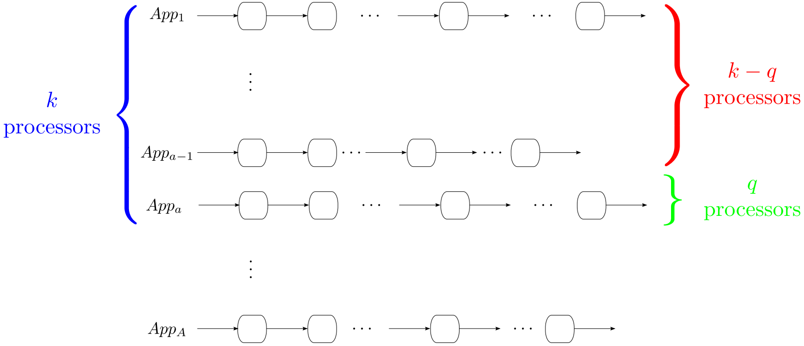
  $\min_{1 \leq k \leq p} P_1(1,n,k)$
- Recurrence relation:

$$P_1(i,j,k) = \min_{1 \leq \ell \leq j-1} \left(P_1(i,\ell,k-1) + P_1(\ell+1,j,1) \right)$$



Many applications

- Recurrence:
  $P(a,k) = \min_{1 \leq q < k} \left(P(a-1, k-q) + P_1(1, n, q) \right)$



Tri-criteria complexity results

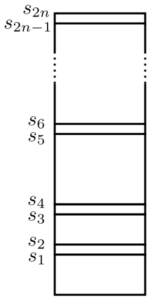
	proc-hom	proc-het		
	com-hom	special-app	com-hom	com-het
one-to-one or interval	NP-complete			

Reduction from 2-PARTITION

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

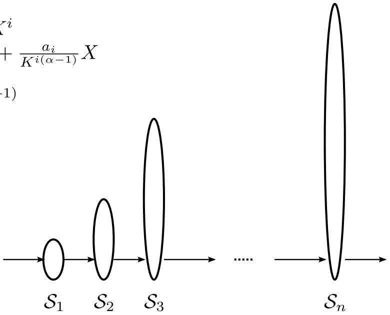
Problem instance

One-to-one mapping - fully homogeneous platform



$$\begin{cases} s_{2i-1} = K^i \\ s_{2i} = K^i + \frac{a_i}{K^{i(\alpha-1)}} X \end{cases}$$

$$w_i = K^{i(\alpha+1)}$$



- Stage $\mathcal{S}_i$ must be processed at speed s_{2i-1} or s_{2i}
- $P^0 = P^* + \alpha X(\sigma/2 + 1/2)$, $L^0 = L^* - X(\sigma/2 - 1/2)$, $T^0 = L^0$, where P^* and L^* are power and latency with speeds s_{2i-1}

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

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

Outline of the talk

Heuristics

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

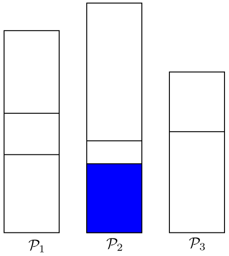
Each heuristic (except H2) exists in two variants: interval mapping without resource sharing and general mapping with resource sharing in order to evaluate the impact of processor reuse

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)

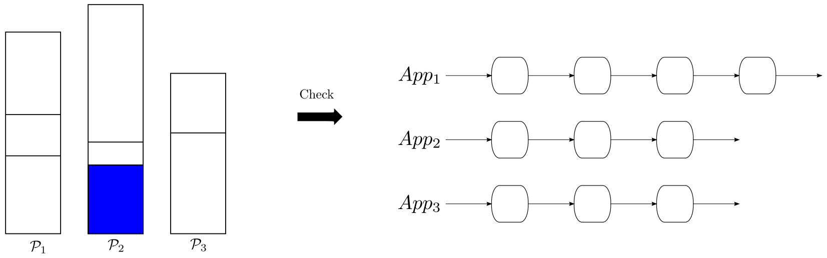
H3-energy

Fix processor speeds



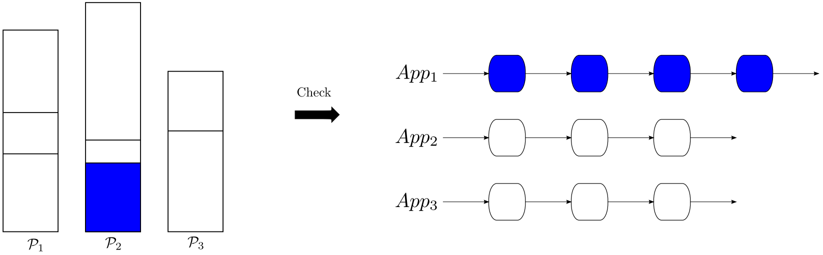
H3-energy

Mapping heuristic: find a valid maping



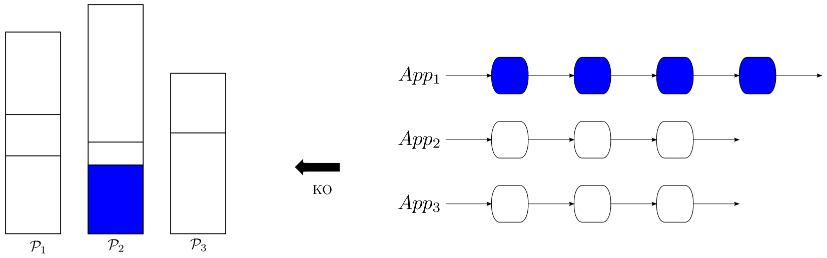
H3-energy

Mapping heuristic: find a valid maping



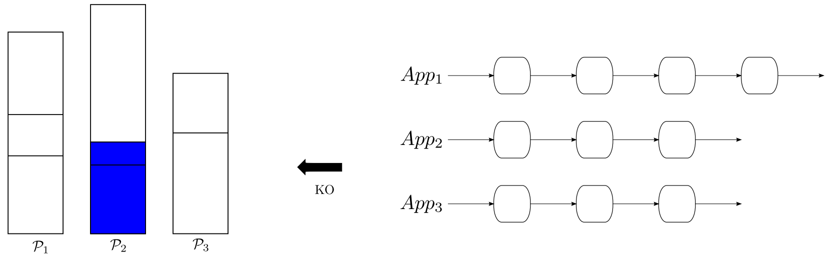
H3-energy

Mapping heuristic: find a valid maping



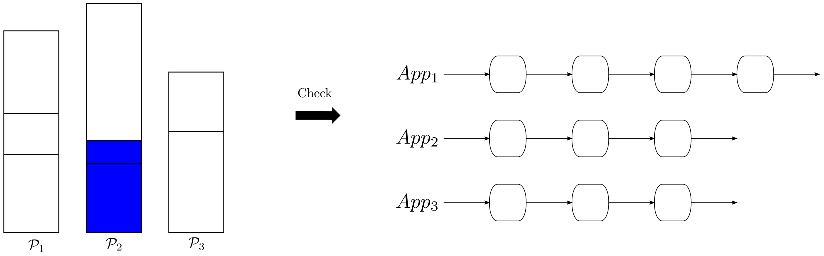
H3-energy

Iterate the process: increase processor speeds



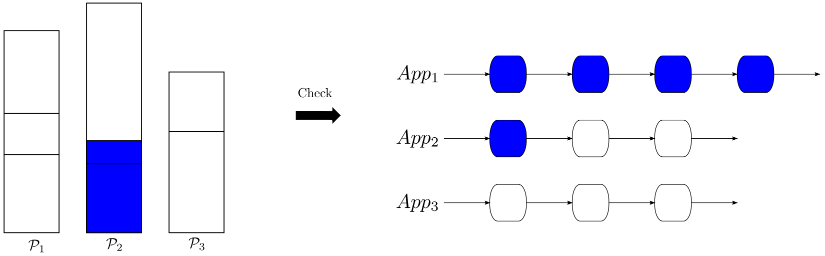
H3-energy

Iterate the process: increase processor speeds



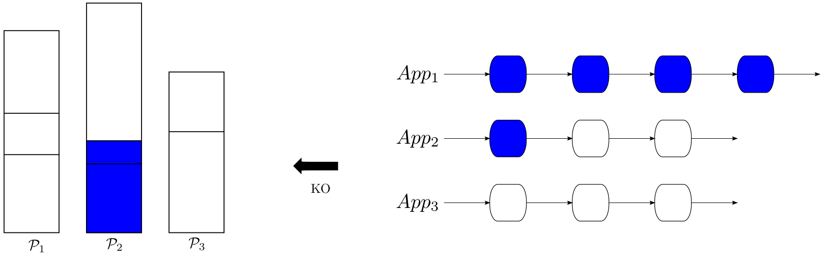
H3-energy

Iterate the process: increase processor speeds



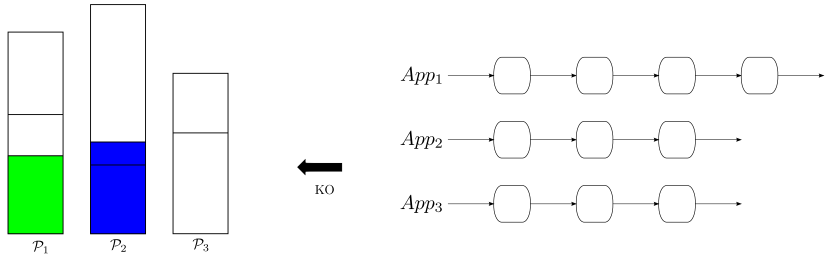
H3-energy

Iterate the process: increase processor speeds



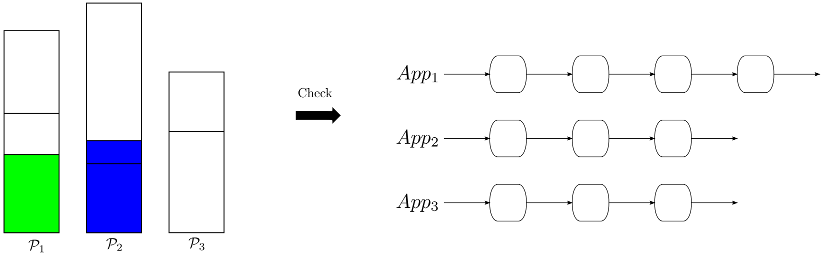
H3-energy

Iterate the process: increase processor speeds



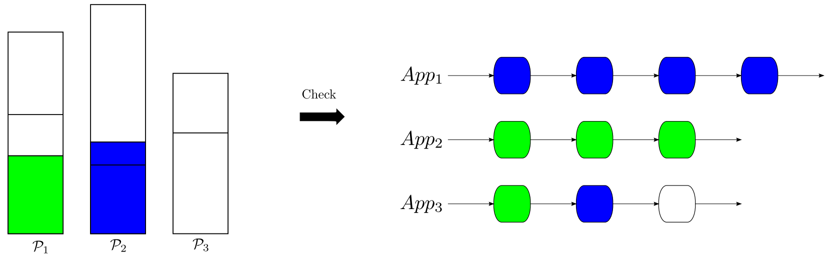
H3-energy

Iterate the process: increase processor speeds



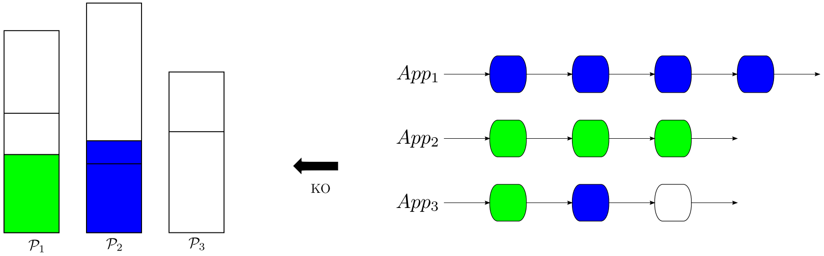
H3-energy

Iterate the process: increase processor speeds



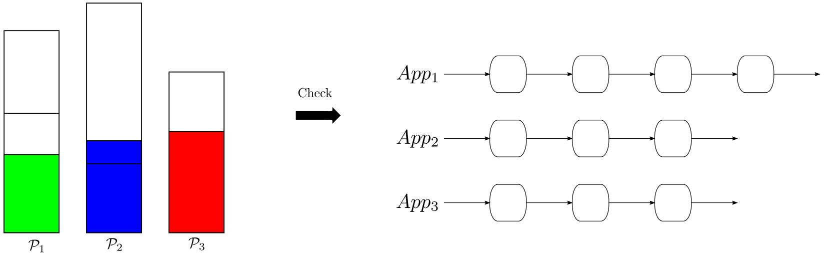
H3-energy

Iterate the process: increase processor speeds



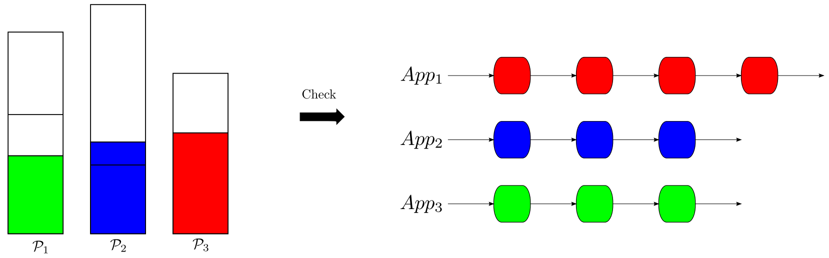
H3-energy

Iterate the process: increase processor speeds



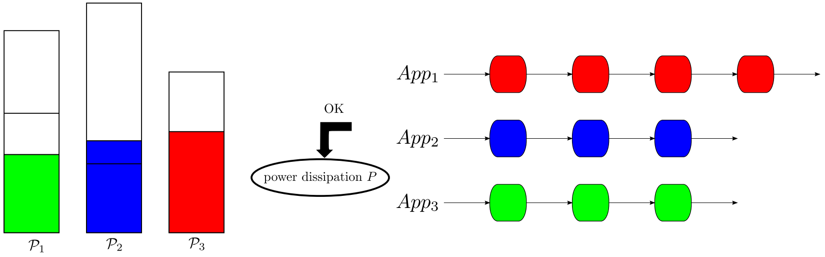
H3-energy

Iterate the process: increase processor speeds



H3-energy

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

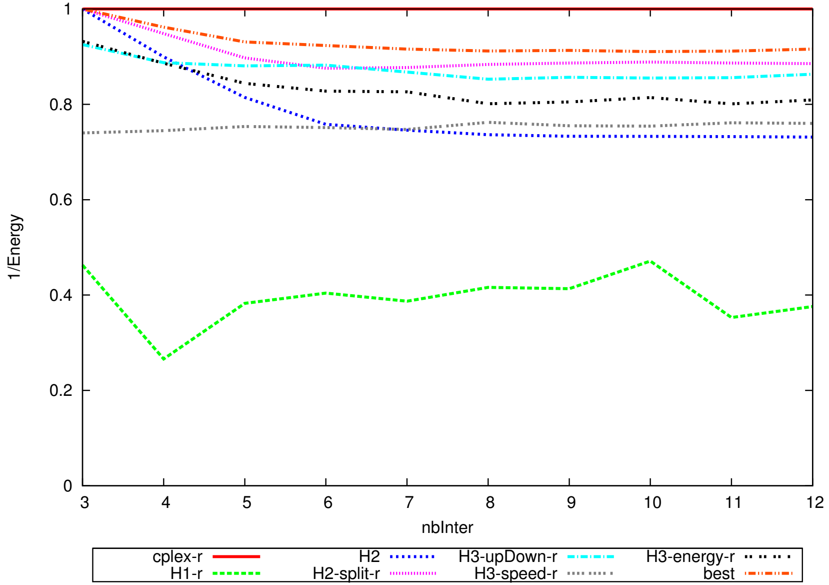
- Integer linear program to assess the absolute performance of the heuristics on small instances
- Small instances: two or three applications, around 15 stages per application, around 8 processors
- Execution time on 30 small instances: less than one second for all heuristics, one week for the ILP
- 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

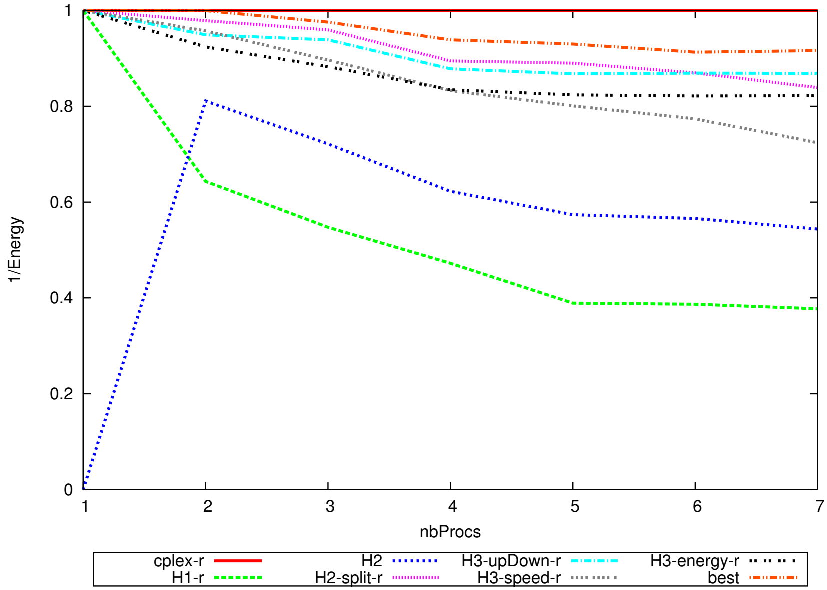
Experimental plan

- 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

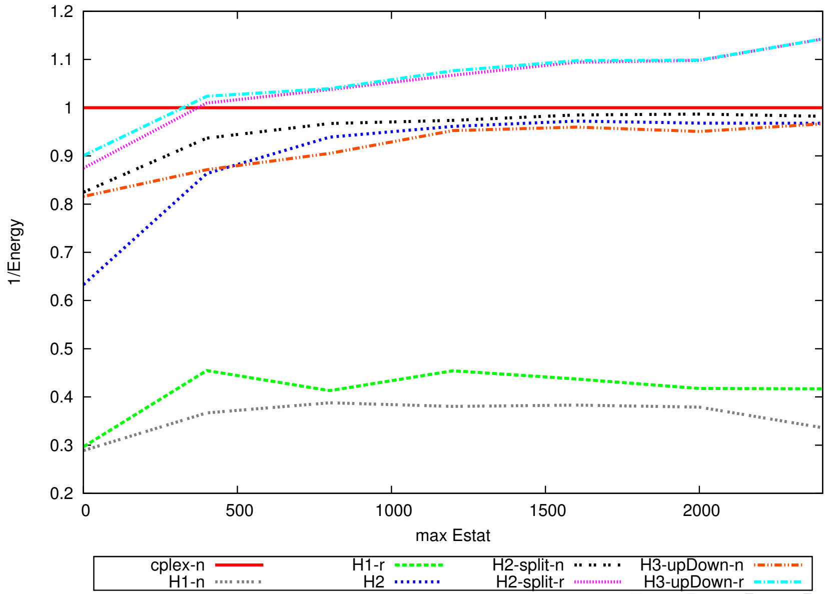
Increasing latency



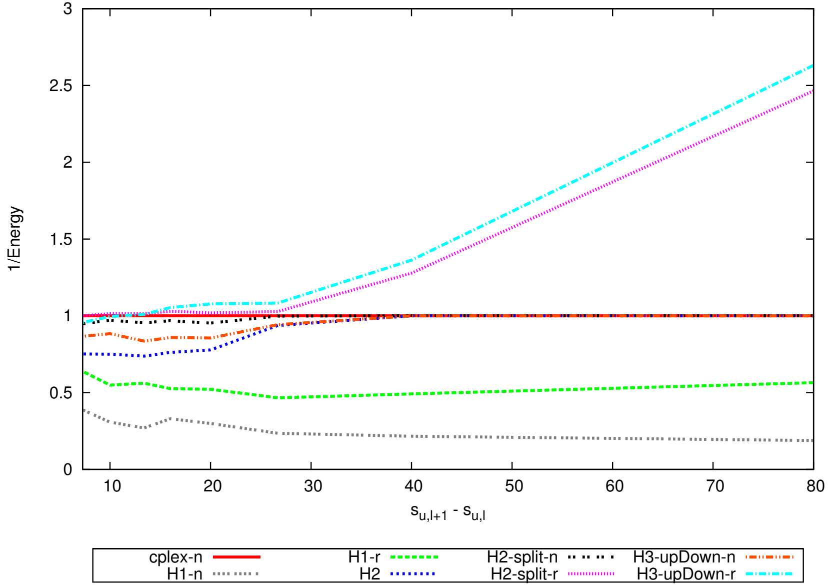
Increasing number of processors



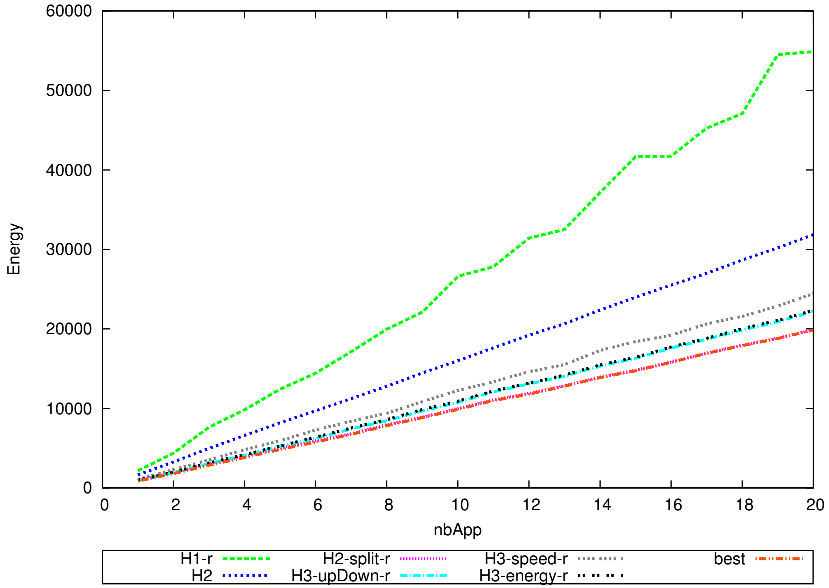
Impact of static power



Impact of mode distribution



Scalability



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

Conclusion

- 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

Conclusion

- 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

Other results and future work

- Pipelined linear chain applications
  - continuous speeds
  - approximation algorithms
- Series-parallel applications
  - Mapping onto chip multi-processors (CMP)
  - Period/power trade-offs
  - Impact of routing and power consumed by communications
- Replica placement
  - Power-aware algorithms
  - Pseudo-polynomial algorithms, problem NP-hard with M different possible speeds
- Reflexion on energy models
  - discrete vs continuous vs VDD-hopping vs incremental

Other results and future work

- Pipelined linear chain applications
  - continuous speeds
  - approximation algorithms
- Series-parallel applications
  - Mapping onto chip multi-processors (CMP)
  - Period/power trade-offs
  - Impact of routing and power consumed by communications
- Replica placement
  - Power-aware algorithms
  - Pseudo-polynomial algorithms, problem NP-hard with M different possible speeds
- Reflexion on energy models
  - discrete vs continuous vs VDD-hopping vs incremental

Other results and future work

- Pipelined linear chain applications
  - continuous speeds
  - approximation algorithms
- Series-parallel applications
  - Mapping onto chip multi-processors (CMP)
  - Period/power trade-offs
  - Impact of routing and power consumed by communications
- Replica placement
  - Power-aware algorithms
  - Pseudo-polynomial algorithms, problem NP-hard with M different possible speeds
- Reflexion on energy models
  - discrete vs continuous vs VDD-hopping vs incremental

Other results and future work

- Pipelined linear chain applications
  - continuous speeds
  - approximation algorithms
- Series-parallel applications
  - Mapping onto chip multi-processors (CMP)
  - Period/power trade-offs
  - Impact of routing and power consumed by communications
- Replica placement
  - Power-aware algorithms
  - Pseudo-polynomial algorithms, problem NP-hard with M different possible speeds
- Reflexion on energy models
  - discrete vs continuous vs VDD-hopping vs incremental