Mapping skeleton workflows onto heterogeneous platforms

Anne Benoit and Yves Robert

GRAAL team, LIP
École Normale Supérieure de Lyon

July 2007
Introduction and motivation

- Mapping applications onto parallel platforms
  Difficult challenge
- Heterogeneous clusters, fully heterogeneous platforms
  Even more difficult!
- Structured programming approach
  - Easier to program (deadlocks, process starvation)
  - Range of well-known paradigms (pipeline, farm)
  - Algorithmic skeleton: help for mapping

Mapping skeletons (pipeline, fork) onto heterogeneous platforms
Introduction and motivation

- Mapping applications onto parallel platforms

  **Difficult challenge**

- Heterogeneous clusters, fully heterogeneous platforms

  **Even more difficult!**

- Structured programming approach

  - Easier to program (deadlocks, process starvation)
  - Range of well-known paradigms (pipeline, farm)
  - Algorithmic skeleton: help for mapping

Mapping skeletons (pipeline, fork) onto heterogeneous platforms
Introduction and motivation

- Mapping applications onto parallel platforms
  
  **Difficult challenge**

- Heterogeneous clusters, fully heterogeneous platforms
  
  **Even more difficult!**

- Structured programming approach
  
  - Easier to program (deadlocks, process starvation)
  - Range of well-known paradigms (pipeline, farm)
  - Algorithmic skeleton: help for mapping

Mapping skeletons (pipeline, fork) onto heterogeneous platforms
Introduction and motivation

- Mapping applications onto parallel platforms
  Difficult challenge

- Heterogeneous clusters, fully heterogeneous platforms
  Even more difficult!

- Structured programming approach
  - Easier to program (deadlocks, process starvation)
  - Range of well-known paradigms (pipeline, farm)
  - Algorithmic skeleton: help for mapping

Mapping skeletons (pipeline, fork) onto heterogeneous platforms
Introduction and motivation

- Mapping applications onto parallel platforms
  Difficult challenge

- Heterogeneous clusters, fully heterogeneous platforms
  Even more difficult!

- Structured programming approach
  - Easier to program (deadlocks, process starvation)
  - Range of well-known paradigms (pipeline, farm)
  - Algorithmic skeleton: help for mapping

Mapping skeletons (pipeline, fork) onto heterogeneous platforms
Introduction and motivation

- Mapping applications onto parallel platforms
  Difficult challenge

- Heterogeneous clusters, fully heterogeneous platforms
  Even more difficult!

- Structured programming approach
  - Easier to program (deadlocks, process starvation)
  - Range of well-known paradigms (pipeline, farm)
  - Algorithmic skeleton: help for mapping

Mapping skeletons (pipeline, fork) onto heterogeneous platforms
Introduction and motivation

- Mapping applications onto parallel platforms
  Difficult challenge

- Heterogeneous clusters, fully heterogeneous platforms
  Even more difficult!

- Structured programming approach
  - Easier to program (deadlocks, process starvation)
  - Range of well-known paradigms (pipeline, farm)
  - Algorithmic skeleton: help for mapping

Mapping skeletons (pipeline, fork) onto heterogeneous platforms
Rule of the game

- Map each pipeline stage on a single processor (extended later)
- Goal: minimize execution time (extended later)

Several mapping strategies

\[ S_1 \rightarrow S_2 \rightarrow \ldots \rightarrow S_k \rightarrow \ldots \rightarrow S_n \]

The pipeline application
Rule of the game

- Map each pipeline stage on a single processor (extended later)
- Goal: minimize execution time (extended later)
- Several mapping strategies

\[ S_1 \rightarrow S_2 \rightarrow \cdots \rightarrow S_k \rightarrow \cdots \rightarrow S_n \]

The pipeline application
Rule of the game

- Map each pipeline stage on a single processor (extended later)
- Goal: minimize execution time (extended later)
- Several mapping strategies

One-to-one Mapping
Rule of the game

- Map each pipeline stage on a single processor (extended later)
- Goal: minimize execution time (extended later)
- Several mapping strategies

\[ S_1 \rightarrow S_2 \rightarrow \ldots \rightarrow S_k \rightarrow \ldots \rightarrow S_n \]

**Interval Mapping**
Rule of the game

- Map each pipeline stage on a single processor (extended later)
- Goal: minimize execution time (extended later)
- Several mapping strategies

GENERAL MAPPING
Major contributions

Theory
- Formal approach to the problem, definition of replication and data-parallelism
- Problem complexity for several cases
- Integer linear program for exact resolution

Practice
- Heuristics for INTERVAL MAPPING on clusters
- Experiments to compare heuristics and evaluate their absolute performance
Major contributions

**Theory**  
Formal approach to the problem, definition of replication and data-parallelism  
Problem complexity for several cases  
Integer linear program for exact resolution

**Practice**  
Heuristics for *Interval Mapping* on clusters  
Experiments to compare heuristics and evaluate their absolute performance
Outline

1. Framework
2. Working out an example
3. Part 1 - Communications, monolithic stages, mono-criterion
4. Part 2 - Simpler model with no communications, but with replication/DP and bi-criteria
5. Conclusion
Outline

1. **Framework**

2. Working out an example

3. Part 1 - Communications, monolithic stages, mono-criterion

4. Part 2 - Simpler model with no communications, but with replication/DP and bi-criteria

5. Conclusion
The application: pipeline graphs

- $n$ stages $S_k$, $1 \leq k \leq n$
- $S_k$:
  - receives input of size $\delta_{k-1}$ from $S_{k-1}$
  - performs $w_k$ computations
  - outputs data of size $\delta_k$ to $S_{k+1}$
The application: fork graphs

- $n+1$ stages $S_k$, $0 \leq k \leq n$
  - $S_0$: root stage
  - $S_1$ to $S_n$: independent stages
- A data-set goes through stage $S_0$, then it can be executed simultaneously for all other stages
The platform

- \( P_u \): processors, \( 1 \leq u \leq p \), fully interconnected
- \( s_u \): speed of processor \( P_u \)
- bidirectional link \( \text{link}_{u,v} : P_u \rightarrow P_v \), bandwidth \( b_{u,v} \)
- one-port model: each processor can either send, receive or compute at any time-step
Different platforms

**Fully Homogeneous** – Identical processors \((s_u = s)\) and links \((b_{u,v} = b)\): typical parallel machines

**Communication Homogeneous** – Different-speed processors \((s_u \neq s_v)\), identical links \((b_{u,v} = b)\): networks of workstations, clusters

**Fully Heterogeneous** – Fully heterogeneous architectures, \(s_u \neq s_v\) and \(b_{u,v} \neq b_{u',v'}\): hierarchical platforms, grids
Rule of the game

- Consecutive data-sets fed into the workflow
- **Period** $T_{\text{period}} = \text{time interval between beginning of execution of two consecutive data sets} \ (\text{throughput} = 1/T_{\text{period}})$
- **Latency** $T_{\text{latency}}(x) = \text{time elapsed between beginning and end of execution for a given data set } x$, and $T_{\text{latency}} = \max_x T_{\text{latency}}(x)$
- Map each pipeline/fork stage on one or several processors
- Goal: minimize $T_{\text{period}}$ or $T_{\text{latency}}$ or bi-criteria minimization
Rule of the game

- Consecutive data-sets fed into the workflow
- **Period** $T_{\text{period}} = \text{time interval between beginning of execution of two consecutive data sets (throughput=1/} T_{\text{period}}\right)$
- **Latency** $T_{\text{latency}}(x) = \text{time elapsed between beginning and end of execution for a given data set } x$, and $T_{\text{latency}} = \max_x T_{\text{latency}}(x)$
- Map each pipeline/fork stage on **one or several** processors
- Goal: minimize $T_{\text{period}}$ or $T_{\text{latency}}$ or bi-criteria minimization
Stage types

- **Monolithic stages**: must be mapped on one single processor since computation for a data-set may depend on result of previous computation.

- **Replicable stages**: can be replicated on several processors, but not parallel, *i.e.* a data-set must be entirely processed on a single processor.

- **Data-parallel stages**: inherently parallel stages, one data-set can be computed in parallel by several processors.
Replication

Replicate stage $S_k$ on $P_1, \ldots, P_q$

$S_k$ on $P_1$: data sets 1, 4, 7, \ldots

$S_k$ on $P_2$: data sets 2, 5, 8, \ldots

$S_k$ on $P_3$: data sets 3, 5, 9, \ldots

$S_{k+1}$ may be monolithic: output order must be respected

Round-robin rule to ensure output order

Cannot feed more fast processors than slow ones

Most efficient with similar-speed processors
Replication

Replicate stage $S_k$ on $P_1, \ldots, P_q$

\[
\begin{align*}
\uparrow & \quad S_k \text{ on } P_1: \text{ data sets } 1, 4, 7, \ldots \\
\downarrow \quad & S_k \text{ on } P_2: \text{ data sets } 2, 5, 8, \ldots \\
\downarrow \quad & S_k \text{ on } P_3: \text{ data sets } 3, 5, 9, \ldots \\
\downarrow & \quad S_{k+1} \text{ on } \ldots
\end{align*}
\]

- $S_{k+1}$ may be monolithic: output order must be respected
- Round-robin rule to ensure output order
- Cannot feed more fast processors than slow ones
- Most efficient with similar-speed processors
Data-parallelism

Data-parallelize stage $S_k$ on $P_1, \ldots, P_q$

$S_k \ (w = 16)$

$\Rightarrow$

$P_1 \ (s_1 = 2) : \bullet \bullet \bullet \bullet \bullet \bullet \bullet \bullet$

$P_2 \ (s_2 = 1) : \bullet \bullet \bullet \bullet \bullet \bullet \bullet \bullet$

$P_3 \ (s_3 = 1) : \bullet \bullet \bullet \bullet \bullet \bullet \bullet \bullet$

- Perfect sharing of the work
- Data-parallelize single stage only
**Data-parallelism**

**Data-parallelize** stage $S_k$ on $P_1, \ldots, P_q$

$S_k \ (w = 16)$  

\[ \begin{align*} 
&\Rightarrow P_1 \ (s_1 = 2) : \bullet \bullet \bullet \bullet \bullet \bullet \\
&\quad P_2 \ (s_2 = 1) : \bullet \bullet \bullet \\
&\quad P_3 \ (s_3 = 1) : \bullet \bullet \bullet 
\end{align*} \]

- Perfect sharing of the work
- Data-parallelize single stage only
**Interval Mapping** for pipeline graphs

- Several consecutive stages onto the same processor
- Increase computational load, reduce communications

Partition of \([1..n]\) into \(m\) intervals \(l_j = [d_j, e_j]\)
(with \(d_j \leq e_j\) for \(1 \leq j \leq m\), \(d_1 = 1\), \(d_{j+1} = e_j + 1\) for \(1 \leq j \leq m - 1\) and \(e_m = n\))

Interval \(l_j\) mapped onto processor \(P_{\text{alloc}(j)}\)

\[
T_{\text{period}} = \max_{1 \leq j \leq m} \left\{ \frac{\delta_{d_j-1}}{b_{\text{alloc}(j-1),\text{alloc}(j)}} + \sum_{i=d_j}^{e_j} \frac{w_i}{s_{\text{alloc}(j)}} + \frac{\delta_{e_j}}{b_{\text{alloc}(j),\text{alloc}(j+1)}} \right\}
\]

\[
T_{\text{latency}} = \sum_{1 \leq j \leq m} \left\{ \frac{\delta_{d_j-1}}{b_{\text{alloc}(j-1),\text{alloc}(j)}} + \sum_{i=d_j}^{e_j} \frac{w_i}{s_{\text{alloc}(j)}} \right\} + \frac{\delta_{n}}{b_{\text{alloc}(m),\text{alloc}(m+1)}}
\]

Anne.Benoit@ens-lyon.fr
Tokyo, July 2007
Mapping skeleton workflows
POP’07 15/44
Interval Mapping for pipeline graphs

- Several consecutive stages onto the same processor
- Increase computational load, reduce communications
- Partition of $[1..n]$ into $m$ intervals $I_j = [d_j, e_j]$ (with $d_j \leq e_j$ for $1 \leq j \leq m$, $d_1 = 1$, $d_{j+1} = e_j + 1$ for $1 \leq j \leq m - 1$ and $e_m = n$)
- Interval $I_j$ mapped onto processor $P_{\text{alloc}(j)}$

$$T_{\text{period}} = \max_{1 \leq j \leq m} \left\{ \frac{\delta_{d_j-1}}{b_{\text{alloc}(j-1),\text{alloc}(j)}} + \frac{\sum_{i=d_j}^{e_j} w_i}{s_{\text{alloc}(j)}} + \frac{\delta_{e_j}}{b_{\text{alloc}(j),\text{alloc}(j+1)}} \right\}$$

$$T_{\text{latency}} = \sum_{1 \leq j \leq m} \left\{ \frac{\delta_{d_j-1}}{b_{\text{alloc}(j-1),\text{alloc}(j)}} + \frac{\sum_{i=d_j}^{e_j} w_i}{s_{\text{alloc}(j)}} \right\} + \frac{\delta_n}{b_{\text{alloc}(m),\text{alloc}(m+1)}}$$

Anne.Benoit@ens-lyon.fr
Tokyo, July 2007
Mapping skeleton workflows
POP’07 15/44
Interval Mapping for pipeline graphs

- Several consecutive stages onto the same processor
- Increase computational load, reduce communications
- Partition of \([1..n]\) into \(m\) intervals \(I_j = [d_j, e_j]\)
  (with \(d_j \leq e_j\) for \(1 \leq j \leq m\), \(d_1 = 1\), \(d_{j+1} = e_j + 1\) for \(1 \leq j \leq m - 1\) and \(e_m = n\))
- Interval \(I_j\) mapped onto processor \(P_{\text{alloc}(j)}\)

\[
T_{\text{period}} = \max_{1 \leq j \leq m} \left\{ \frac{\delta_{d_j-1}}{b_{\text{alloc}(j-1),\text{alloc}(j)}} + \frac{\sum_{i=d_j}^{e_j} w_i}{s_{\text{alloc}(j)}} + \frac{\delta_{e_j}}{b_{\text{alloc}(j),\text{alloc}(j+1)}} \right\}
\]

\[
T_{\text{latency}} = \sum_{1 \leq j \leq m} \left\{ \frac{\delta_{d_j-1}}{b_{\text{alloc}(j-1),\text{alloc}(j)}} + \frac{\sum_{i=d_j}^{e_j} w_i}{s_{\text{alloc}(j)}} \right\} + \frac{\delta_n}{b_{\text{alloc}(m),\text{alloc}(m+1)}}
\]
**Interval Mapping** for pipeline graphs

- Several consecutive stages onto the same processor
- Increase computational load, reduce communications
- Partition of \([1..n]\) into \(m\) intervals \(I_j = [d_j, e_j]\) (with \(d_j \leq e_j\) for \(1 \leq j \leq m\), \(d_1 = 1\), \(d_{j+1} = e_j + 1\) for \(1 \leq j \leq m - 1\) and \(e_m = n\))
- Interval \(I_j\) mapped onto processor \(P_{\text{alloc}(j)}\)

\[
T_{\text{period}} = \max_{1 \leq j \leq m} \left\{ \frac{\delta_{d_j-1}}{b_{\text{alloc}(j-1),\text{alloc}(j)}} + \frac{\sum_{i=d_j}^{e_j} w_i}{s_{\text{alloc}(j)}} + \frac{\delta_{e_j}}{b_{\text{alloc}(j),\text{alloc}(j+1)}} \right\}
\]

\[
T_{\text{latency}} = \sum_{1 \leq j \leq m} \left\{ \frac{\delta_{d_j-1}}{b_{\text{alloc}(j-1),\text{alloc}(j)}} + \frac{\sum_{i=d_j}^{e_j} w_i}{s_{\text{alloc}(j)}} \right\} + \frac{\delta_n}{b_{\text{alloc}(m),\text{alloc}(m+1)}}
\]
Simpler problem, replication and data-parallelism

- No communication costs nor overheads

- Cost to execute $S_i$ on $P_u$ alone: $\frac{w_i}{s_u}$

- Cost to data-parallelize $[S_i, S_j]$ ($i = j$ for pipeline; $0 < i \leq j$ or $i = j = 0$ for fork) on $k$ processors $P_{q_1}, \ldots, P_{q_k}$:

$$\frac{\sum_{\ell=i}^{j} w_{\ell}}{\sum_{u=1}^{k} s_{q_u}}$$

Cost = $T_{\text{period}}$ of assigned processors
Cost = delay to traverse the interval
Simpler problem, replication and data-parallelism

- No communication costs nor overheads

- Cost to execute $S_i$ on $P_u$ alone: $\frac{w_i}{s_u}$

- Cost to data-parallelize $[S_i, S_j]$ ($i = j$ for pipeline; $0 < i \leq j$ or $i = j = 0$ for fork) on $k$ processors $P_{q_1}, \ldots, P_{q_k}$:

$$\frac{\sum_{\ell=i}^{j} w_\ell}{\sum_{u=1}^{k} s_{q_u}}$$

Cost = $T_{\text{period}}$ of assigned processors

Cost = delay to traverse the interval
Simpler problem, replication and data-parallelism

- No communication costs nor overheads
- Cost to execute $S_i$ on $P_u$ alone: $\frac{w_i}{s_u}$
- Cost to data-parallelize $[S_i, S_j]$ ($i = j$ for pipeline; $0 < i \leq j$ or $i = j = 0$ for fork) on $k$ processors $P_{q_1}, \ldots, P_{q_k}$:
  \[
  \frac{\sum_{\ell=i}^{j} w_\ell}{\sum_{u=1}^{k} s_{q_u}}
  \]
- Cost = $T_{\text{period}}$ of assigned processors
- Cost = delay to traverse the interval
Simpler problem, replication and data-parallelism

- Cost to replicate \([S_i, S_j]\) on \(k\) processors \(P_{q_1}, \ldots, P_{q_k}\):

\[
\frac{\sum_{\ell=i}^{j} w_\ell}{k \times \min_{1 \leq u \leq k} s_{qu}}.
\]

Cost = \(T_{\text{period}}\) of assigned processors
Delay to traverse the interval = time needed by slowest processor:

\[
t_{\text{max}} = \frac{\sum_{\ell=i}^{j} w_\ell}{\min_{1 \leq u \leq k} s_{qu}}
\]

- With these formulas: easy to compute \(T_{\text{period}}\) and \(T_{\text{latency}}\) for pipeline graphs
Cost to replicate \([S_i, S_j]\) on \(k\) processors \(P_{q_1}, \ldots, P_{q_k}\):

\[
\sum_{\ell=i}^{j} w_{\ell} \quad \frac{1}{k \times \min_{1 \leq u \leq k} s_{q_u}}.
\]

Cost = \(T_{\text{period}}\) of assigned processors

Delay to traverse the interval = time needed by slowest processor:

\[
t_{\text{max}} = \frac{\sum_{\ell=i}^{j} w_{\ell}}{\min_{1 \leq u \leq k} s_{q_u}}
\]

With these formulas: easy to compute \(T_{\text{period}}\) and \(T_{\text{latency}}\) for pipeline graphs
Outline

1. Framework

2. Working out an example

3. Part 1 - Communications, monolithic stages, mono-criterion

4. Part 2 - Simpler model with no communications, but with replication/DP and bi-criteria

5. Conclusion
Working out an example

\[ S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \]

14 → 4 → 2 → 4

Interval mapping, 4 processors, \( s_1 = 2 \) and \( s_2 = s_3 = s_4 = 1 \)

Optimal period?
Working out an example

\[ S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \]

14 4 2 4

Interval mapping, 4 processors, \( s_1 = 2 \) and \( s_2 = s_3 = s_4 = 1 \)

Optimal period?

\[ T_{\text{period}} \] = 7, \( S_1 \rightarrow P_1, S_2S_3 \rightarrow P_2, S_4 \rightarrow P_3 \) (\( T_{\text{latency}} \) = 17)

Optimal latency?

Anne.Benoit@ens-lyon.fr

Tokyo, July 2007
Working out an example

\[ S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \]

14 4 2 4

Interval mapping, 4 processors, \( s_1 = 2 \) and \( s_2 = s_3 = s_4 = 1 \)

**Optimal period?**

\[ T_{\text{period}} = 7, \ S_1 \rightarrow P_1, \ S_2S_3 \rightarrow P_2, \ S_4 \rightarrow P_3 \ (T_{\text{latency}} = 17) \]

**Optimal latency?**

\[ T_{\text{latency}} = 12, \ S_1S_2S_3S_4 \rightarrow P_1 \ (T_{\text{period}} = 12) \]

Min. latency if \( T_{\text{period}} \leq 10? \)
Working out an example

\[ S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \]
14 4 2 4

Interval mapping, 4 processors, \( s_1 = 2 \) and \( s_2 = s_3 = s_4 = 1 \)

Optimal period?
\[ T_{\text{period}} = 7, \ S_1 \rightarrow P_1, \ S_2S_3 \rightarrow P_2, \ S_4 \rightarrow P_3 \left( T_{\text{latency}} = 17 \right) \]

Optimal latency?
\[ T_{\text{latency}} = 12, \ S_1S_2S_3S_4 \rightarrow P_1 \left( T_{\text{period}} = 12 \right) \]

Min. latency if \( T_{\text{period}} \leq 10? \)
\[ T_{\text{latency}} = 14, \ S_1S_2S_3 \rightarrow P_1, \ S_4 \rightarrow P_2 \]
Example with replication and data-parallelism

\[ S_1 \to S_2 \to S_3 \to S_4 \]

14 \quad 4 \quad 2 \quad 4

Interval mapping, 4 processors, \( s_1 = 2 \) and \( s_2 = s_3 = s_4 = 1 \)

Replicate interval \([S_u..S_v]\) on \( P_1, \ldots, P_q \)

\[
\begin{align*}
\vdash & \quad S_u \ldots S_v \text{ on } P_1: \text{ data sets } 1, 4, 7, \ldots \\
\vdash & \quad S_u \ldots S_v \text{ on } P_2: \text{ data sets } 2, 5, 8, \ldots \\
\vdash & \quad S_u \ldots S_v \text{ on } P_3: \text{ data sets } 3, 5, 9, \ldots \\
\text{ } & \quad S \ldots \\
\end{align*}
\]

\[ T_{\text{period}} = \frac{\sum_{k=\nu}^{\nu} w_k}{q \times \min_i(s_i)} \quad \text{and} \quad T_{\text{latency}} = q \times T_{\text{period}} \]
Example with replication and data-parallelism

\[ S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \]
\[ 14 \quad 4 \quad 2 \quad 4 \]

Interval mapping, 4 processors, \( s_1 = 2 \) and \( s_2 = s_3 = s_4 = 1 \)

Data Parallelize single stage \( S_k \) on \( P_1, \ldots, P_q \)

\[ S \ (w = 16) \]
\[
\begin{array}{l}
\bullet \bullet \bullet \\
\bullet \bullet \bullet \\
\bullet \bullet \bullet
\end{array}
\Rightarrow
\begin{array}{l}
P_1 \ (s_1 = 2) : \bullet \bullet \bullet \bullet \bullet \bullet \bullet \\
P_2 \ (s_2 = 1) : \bullet \bullet \bullet \\
P_3 \ (s_3 = 1) : \bullet \bullet \bullet \bullet \\
\end{array}
\]

\[
T_{\text{period}} = \frac{w_k}{\sum_{i=1}^{q} s_i} \quad \text{and} \quad T_{\text{latency}} = T_{\text{period}}
\]
Example with replication and data-parallelism

\[ S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \]

14 4 2 4

Interval mapping, 4 processors, \( s_1 = 2 \) and \( s_2 = s_3 = s_4 = 1 \)

Optimal period?
Example with replication and data-parallelism

\[ S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \]
\[ 14 \quad 4 \quad 2 \quad 4 \]

Interval mapping, 4 processors, \( s_1 = 2 \) and \( s_2 = s_3 = s_4 = 1 \)

Optimal period?

\[ S_1 \xrightarrow{\text{DP}} P_1P_2, \quad S_2S_3S_4 \xrightarrow{\text{REP}} P_3P_4 \]

\[ T_{\text{period}} = \max\left(\frac{14}{2+1}, \frac{4+2+4}{2\times1}\right) = 5, \quad T_{\text{latency}} = 14.67 \]
Example with replication and data-parallelism

\[
S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4
\]

14 4 2 4

Interval mapping, 4 processors, \( s_1 = 2 \) and \( s_2 = s_3 = s_4 = 1 \)

Optimal period?

\[
S_1^{DP} \rightarrow P_1P_2, \quad S_2S_3S_4^{REP} \rightarrow P_3P_4
\]

\[
T_{\text{period}} = \max(\frac{14}{2+1}, \frac{4+2+4}{2\times1}) = 5, \quad T_{\text{latency}} = 14.67
\]

\[
S_1^{DP} \rightarrow P_2P_3P_4, \quad S_2S_3S_4 \rightarrow P_1
\]

\[
T_{\text{period}} = \max(\frac{14}{1+1+1}, \frac{4+2+4}{2}) = 5, \quad T_{\text{latency}} = 9.67 \quad \text{(optimal)}
\]
Outline

1 Framework

2 Working out an example

3 Part 1 - Communications, monolithic stages, mono-criterion

4 Part 2 - Simpler model with no communications, but with replication/DP and bi-criteria

5 Conclusion
Part 1

- Pipeline graph
- Different platforms, with communications
- Different mapping strategies
- Only monolithic stages: no replication nor data-parallelism
- Mono-criterion: period minimization

- Complexity results, heuristics and experiments
Part 1

- Pipeline graph
- Different platforms, with communications
- Different mapping strategies
- Only monolithic stages: no replication nor data-parallelism
- Mono-criterion: period minimization

- Complexity results, heuristics and experiments
### Complexity results

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>One-to-one Mapping</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Interval Mapping</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>General Mapping</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Complexity results

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>One-to-one Mapping</strong></td>
<td>polynomial</td>
<td>polynomial</td>
</tr>
<tr>
<td><strong>Interval Mapping</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>General Mapping</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Binary search polynomial algorithm for **ONE-TO-ONE MAPPING**
Complexity results

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>One-to-one Mapping</td>
<td>polynomial</td>
<td>polynomial</td>
</tr>
<tr>
<td>Interval Mapping</td>
<td>polynomial</td>
<td>NP-complete</td>
</tr>
<tr>
<td>General Mapping</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Binary search **polynomial algorithm** for **ONE-TO-ONE MAPPING**
- Dynamic programming algorithm for **INTERVAL MAPPING** on Hom. platforms (**NP-hard otherwise**)
### Complexity results

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>One-to-one Mapping</td>
<td>polynomial</td>
<td>polynomial</td>
</tr>
<tr>
<td>Interval Mapping</td>
<td>polynomial</td>
<td>NP-complete</td>
</tr>
<tr>
<td>General Mapping</td>
<td>same complexity as Interval</td>
<td></td>
</tr>
</tbody>
</table>

- Binary search **polynomial algorithm** for **One-to-one Mapping**
- Dynamic programming algorithm for **Interval Mapping** on Hom. platforms (**NP-hard otherwise**)
- General mapping: same complexity as **Interval Mapping**
## Complexity results

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>One-to-one Mapping</strong></td>
<td>polynomial</td>
<td>polynomial</td>
</tr>
<tr>
<td><strong>Interval Mapping</strong></td>
<td>polynomial</td>
<td>NP-complete</td>
</tr>
<tr>
<td><strong>General Mapping</strong></td>
<td>same complexity as Interval</td>
<td></td>
</tr>
</tbody>
</table>

- Binary search **polynomial algorithm** for **ONE-TO-ONE MAPPING**
- Dynamic programming algorithm for **INTERVAL MAPPING** on Hom. platforms (**NP-hard otherwise**)
- General mapping: same complexity as **INTERVAL MAPPING**
- All problem instances NP-complete on **Fully Heterogeneous platforms**
One-to-one/Comm. Hom.: binary search algorithm

- Work with fastest n processors, numbered $P_1$ to $P_n$, where
  $s_1 \leq s_2 \leq \ldots \leq s_n$
- Mark all stages $S_1$ to $S_n$ as free
- **For $u = 1$ to $n$**
  - Pick up any free stage $S_k$ s.t. $\delta_{k-1}/b + w_k/s_u + \delta_k/b \leq T_{\text{period}}$
  - Assign $S_k$ to $P_u$, and mark $S_k$ as already assigned
  - If no stage found return "failure"

- **Proof:** exchange argument
One-to-one/Comm. Hom.: binary search algorithm

- Work with fastest \( n \) processors, numbered \( P_1 \) to \( P_n \), where \( s_1 \leq s_2 \leq \ldots \leq s_n \)
- Mark all stages \( S_1 \) to \( S_n \) as free
- For \( u = 1 \) to \( n \)
  - Pick up any free stage \( S_k \) s.t. \( \delta_{k-1}/b + w_k/s_u + \delta_k/b \leq T_{\text{period}} \)
  - Assign \( S_k \) to \( P_u \), and mark \( S_k \) as already assigned
  - If no stage found return "failure"

- Proof: exchange argument
Greedy heuristics

Target clusters: *Com. hom.* platforms and **Interval Mapping**

**H1a-GR:** random – fixed intervals

**H1b-GRIL:** random interval length

**H2-GSW:** biggest $\sum w$ – Place interval with most computations on fastest processor

**H3-GSD:** biggest $\delta_{in} + \delta_{out}$ – Intervals are sorted by communications ($\delta_{in} + \delta_{out}$)

  *in*: first stage of interval; (*out* − 1): last one

**H4-GP:** biggest period on fastest processor – Balancing computation and communication: processors sorted by decreasing speed $s_u$; for current processor $u$, choose interval with biggest period

  $$(\delta_{in} + \delta_{out})/b + \sum_{i \in \text{Interval}} w_i/s_u$$
Sophisticated heuristics

**H5-BS121:** binary search for **One-to-one Mapping** – optimal algorithm for **One-to-one Mapping**. When $p < n$, application cut in fixed intervals of length $L$.

**H6-SPL:** splitting intervals – Processors sorted by decreasing speed, all stages to first processor. At each step, select used proc $j$ with largest period, split its interval (give fraction of stages to $j'$): minimize $\max(\text{period}(j), \text{period}(j'))$ and split if maximum period improved.

**H7a-BSL and H7b-BSC:** binary search (longest/closest) – Binary search on period $P$: start with stage $s = 1$, build intervals $(s, s')$ fitting on processors. For each $u$, and each $s' \geq s$, compute period $(s..s', u)$ and check whether it is smaller than $P$. **H7a:** maximizes $s'$; **H7b:** chooses the closest period.
Plan of experiments

- Assess performance of polynomial heuristics
  - Random applications, \( n = 1 \) to 50 stages
  - Random platforms, \( p = 10 \) and \( p = 100 \) processors
  - \( b = 10 \) (comm. hom.), proc. speed between 1 and 20
  - Relevant parameters: ratios \( \frac{\delta}{b} \) and \( \frac{w}{s} \)
  - Average over 100 similar random appli/platform pairs
Plan of experiments

- Assess performance of polynomial heuristics
- Random applications, $n = 1$ to 50 stages
- Random platforms, $p = 10$ and $p = 100$ processors
- $b = 10$ (comm. hom.), proc. speed between 1 and 20
- Relevant parameters: ratios $\frac{\delta}{b}$ and $\frac{w}{s}$
- Average over 100 similar random appli/platform pairs
Experiment 1 - balanced comm/comp, hom comm

- $\delta_i = 10$, computation time between 1 and 20
- 10 processors

![Graph showing performance comparison of different algorithm implementations.](image-url)
Experiment 1 - balanced comm/comp, hom comm

- $\delta_i = 10$, computation time between 1 and 20
- 100 processors

![Graph showing the relationship between maximum period and number of stages for different algorithms.](image)

- H1a-GreedyRandom
- H1b-GreedyRandomIntervalLength
- H2-GreedySumW
- H3-GreedySumDinDout
- H4-GreedyPeriod
- H5-BinarySearch1to1
- H6-SPLitting
- H7a-BinarySearchLongest
- H7b-BinarySearchClosest
Experiment 2 - balanced comm/comp, het comm

- communication time between 1 and 100
- computation time between 1 and 20

![Graph showing the relationship between maximum period and number of stages for various algorithms.](image)
Experiment 2 - balanced comm/comp, het comm

- communication time between 1 and 100
- computation time between 1 and 20
Experiment 3 - large computations

- communication time between 1 and 20
- computation time between 10 and 1000
Experiment 3 - large computations

- communication time between 1 and 20
- computation time between 10 and 1000

Anne.Benoit@ens-lyon.fr  Tokyo, July 2007
Experiment 4 - small computations

- communication time between 1 and 20
- computation time between 0.01 and 10
Experiment 4 - small computations

- communication time between 1 and 20
- computation time between 0.01 and 10
Summary of experiments

- Much more efficient than random mappings
- Three dominant heuristics for different cases
  - Insignificant communications (hom. or small) and many processors: H5-BS121 (One-to-one Mapping)
  - Insignificant communications (hom. or small) and few processors: H7b-BSC (binary search: clever choice where to split)
  - Important communications (het. or big): H6-SPL (splitting choice relevant for any number of processors)
Summary of experiments

- Much more efficient than random mappings
- Three dominant heuristics for different cases
- Insignificant communications (hom. or small) and many processors: H5-BS121 (One-to-one Mapping)
- Insignificant communications (hom. or small) and few processors: H7b-BSC (binary search: clever choice where to split)
- Important communications (het. or big): H6-SPL (splitting choice relevant for any number of processors)
Outline

1. Framework
2. Working out an example
3. Part 1 - Communications, monolithic stages, mono-criterion
4. Part 2 - Simpler model with no communications, but with replication/DP and bi-criteria
5. Conclusion
Part 2

- Pipeline graph
- Different platforms, with communications
- Different mapping strategies
- Only monolithic stages: no replication nor data-parallelism
- Mono-criterion: period minimization

- Complexity results, heuristics and experiments
Pipeline and fork graphs
Different platforms, with communications
Different mapping strategies
Only monolithic stages: no replication nor data-parallelism
Mono-criterion: period minimization

Complexity results, heuristics and experiments
Part 2

- Pipeline and fork graphs
- Different platforms, without communications
- Different mapping strategies
- Only monolithic stages: no replication nor data-parallelism
- Mono-criterion: period minimization

- Complexity results, heuristics and experiments
Part 2

- Pipeline **and fork** graphs
- Different platforms, **without** communications
- **Interval Mapping only**
- Only monolithic stages: no replication nor data-parallelism
- Mono-criterion: period minimization

- Complexity results, heuristics and experiments
Part 2

- Pipeline and fork graphs
- Different platforms, without communications
- INTERVAL MAPPING only
- Replicable stages, and either data-parallelism or not
- Mono-criterion: period minimization

- Complexity results, heuristics and experiments
Part 2

- Pipeline and fork graphs
- Different platforms, without communications
- **Interval Mapping** only
- Replicable stages, and either data-parallelism or not
- Bi-criteria optimization

- Complexity results, heuristics and experiments
Part 2

- Pipeline and fork graphs
- Different platforms, without communications
- INTERVAL MAPPING only
- Replicable stages, and either data-parallelism or not
- Bi-criteria optimization

- Complexity results only
## Complexity results

Without data-parallelism, *Homogeneous* platforms

<table>
<thead>
<tr>
<th>Objective</th>
<th>period</th>
<th>latency</th>
<th>bi-criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hom. pipeline</td>
<td>-</td>
<td>Poly (str)</td>
<td></td>
</tr>
<tr>
<td>Het. pipeline</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hom. fork</td>
<td>-</td>
<td>Poly (DP)</td>
<td>NP-hard</td>
</tr>
<tr>
<td>Het. fork</td>
<td>Poly (str)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
With data-parallelism, *Homogeneous* platforms

<table>
<thead>
<tr>
<th>Objective</th>
<th>period</th>
<th>latency</th>
<th>bi-criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hom. pipeline</td>
<td></td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Het. pipeline</td>
<td>Poly (DP)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hom. fork</td>
<td>-</td>
<td>Poly (DP)</td>
<td></td>
</tr>
<tr>
<td>Het. fork</td>
<td>Poly (str)</td>
<td>NP-hard</td>
<td></td>
</tr>
</tbody>
</table>
## Complexity results

Without data-parallelism, *Heterogeneous* platforms

<table>
<thead>
<tr>
<th>Objective</th>
<th>period</th>
<th>latency</th>
<th>bi-criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hom. pipeline</td>
<td>Poly (*)</td>
<td>-</td>
<td>Poly (*)</td>
</tr>
<tr>
<td>Het. pipeline</td>
<td>NP-hard (**)</td>
<td>Poly (str)</td>
<td>NP-hard</td>
</tr>
<tr>
<td>Hom. fork</td>
<td>Poly (*)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Het. fork</td>
<td>NP-hard</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
With data-parallelism, *Heterogeneous* platforms

<table>
<thead>
<tr>
<th>Objective</th>
<th>period</th>
<th>latency</th>
<th>bi-criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hom. pipeline</td>
<td></td>
<td>NP-hard</td>
<td></td>
</tr>
<tr>
<td>Het. pipeline</td>
<td></td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Hom. fork</td>
<td></td>
<td>NP-hard</td>
<td></td>
</tr>
<tr>
<td>Het. fork</td>
<td></td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>
Most interesting case:

Without data-parallelism, *Heterogeneous* platforms

<table>
<thead>
<tr>
<th>Objective</th>
<th>period</th>
<th>latency</th>
<th>bi-criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hom. pipeline</td>
<td>Poly (*)</td>
<td>-</td>
<td>Poly (*)</td>
</tr>
<tr>
<td>Het. pipeline</td>
<td>NP-hard (**)</td>
<td>Poly (str)</td>
<td>NP-hard</td>
</tr>
<tr>
<td>Hom. fork</td>
<td>NP-hard</td>
<td>Poly (*)</td>
<td>-</td>
</tr>
<tr>
<td>Het. fork</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
No data-parallelism, *Heterogeneous* platforms

- For pipeline, **minimizing the latency** is straightforward: map all stages on fastest proc
- **Minimizing the period** is NP-hard (involved reduction similar to the heterogeneous chain-to-chain one) for general pipeline
- **Homogeneous pipeline**: all stages have same workload \( w \): in this case, polynomial complexity.

- Polynomial bi-criteria algorithm for homogeneous pipeline
No data-parallelism, *Heterogeneous* platforms

- For pipeline, **minimizing the latency** is straightforward: map all stages on fastest proc.
- **Minimizing the period** is NP-hard (involved reduction similar to the heterogeneous chain-to-chain one) for general pipeline.
- **Homogeneous pipeline**: all stages have same workload $w$: in this case, polynomial complexity.
- **Polynomial bi-criteria algorithm for homogeneous pipeline**
Lemma: form of the solution

Pipeline, no data-parallelism, *Heterogeneous* platform

**Lemma**

*If an optimal solution which minimizes pipeline period uses* \( q \) *processors, consider* \( q \) *fastest processors* \( P_1, \ldots, P_q \), *ordered by non-decreasing speeds: \( s_1 \leq \ldots \leq s_q \). There exists an optimal solution which replicates intervals of stages onto* \( k \) *intervals of processors* \( I_r = [P_{d_r}, P_{e_r}] \), *with* \( 1 \leq r \leq k \leq q \), *\( d_1 = 1 \), \( e_k = q \), *and* \( e_r + 1 = d_{r+1} \) *for* \( 1 \leq r < k \).*

**Proof:** exchange argument, which does not increase latency
Lemma: form of the solution

Pipeline, no data-parallelism, *Heterogeneous* platform

Lemma

*If an optimal solution which minimizes pipeline period uses q processors, consider q fastest processors $P_1, ..., P_q$, ordered by non-decreasing speeds: $s_1 \leq ... \leq s_q$. There exists an optimal solution which replicates intervals of stages onto k intervals of processors $I_r = [P_{d_r}, P_{e_r}]$, with $1 \leq r \leq k \leq q$, $d_1 = 1$, $e_k = q$, and $e_r + 1 = d_{r+1}$ for $1 \leq r < k$.*

Proof: exchange argument, which does not increase latency
Binary-search/Dynamic programming algorithm

- Given latency $L$, given period $K$
- Loop on number of processors $q$
- Dynamic programming algorithm to minimize latency
- Success if $L$ is obtained
  - Binary search on $L$ to minimize latency for fixed period
  - Binary search on $K$ to minimize period for fixed latency
Given latency $L$, given period $K$

Loop on number of processors $q$

Dynamic programming algorithm to minimize latency

Success if $L$ is obtained

Binary search on $L$ to minimize latency for fixed period

Binary search on $K$ to minimize period for fixed latency
Dynamic programming algorithm

- Compute $L(n, 1, q)$, where $L(m, i, j)$ = minimum latency to map $m$ pipeline stages on processors $P_i$ to $P_j$, while fitting in period $K$.

$$L(m, i, j) = \min_{1 \leq m' < m} \begin{cases} \frac{m.w}{s_i} & \text{if } \frac{m.w}{(j-i).s_i} \leq K \\ L(m', i, k) + L(m-m', k+1, j) & \text{if } i \leq k < j \end{cases} (1)$$

- Case (1): replicating $m$ stages onto processors $P_i, ..., P_j$
- Case (2): splitting the interval
Dynamic programming algorithm

- Compute $L(n, 1, q)$, where $L(m, i, j)$ = minimum latency to map $m$ pipeline stages on processors $P_i$ to $P_j$, while fitting in period $K$.

$$L(m, i, j) = \min_{1 \leq m' < m} \left\{ \begin{array}{ll} \frac{m.w}{s_i} & \text{if } \frac{m.w}{(j-i).s_i} \leq K \\ L(m', i, k) + L(m - m', k + 1, j) & \text{otherwise} \end{array} \right. \quad (1)$$

$$L(m, i, i) = \left\{ \begin{array}{ll} \frac{w}{s_i} & \text{if } \frac{w}{(j-i).s_i} \leq K \\ +\infty & \text{otherwise} \end{array} \right. \quad (2)$$

Initialization:

$$L(1, i, j) = \left\{ \begin{array}{ll} \frac{w}{s_i} & \text{if } \frac{w}{(j-i).s_i} \leq K \\ +\infty & \text{otherwise} \end{array} \right.$$
Dynamic programming algorithm

- Compute $L(n, 1, q)$, where $L(m, i, j) = \text{minimum latency to map } m \text{ pipeline stages on processors } P_i \text{ to } P_j$, while fitting in period $K$.

$$L(m, i, j) = \min_{1 \leq m' < m} \left\{ \begin{array}{ll}
\frac{m.w}{s_i} & \text{if } \frac{m.w}{(j-i).s_i} \leq K \\
L(m', i, k) + L(m - m', k + 1, j) & i \leq k < j
\end{array} \right.$$

- **Complexity** of the dynamic programming: $O(n^2.p^4)$
- Number of iterations of the binary search formally bounded, very small number of iterations in practice.
Outline

1. Framework

2. Working out an example

3. Part 1 - Communications, monolithic stages, mono-criterion

4. Part 2 - Simpler model with no communications, but with replication/DP and bi-criteria

5. Conclusion
Related work

Subhlok and Vondran— Extension of their work (pipeline on hom platforms)

Chains-to-chains— In our work possibility to replicate or data-parallelize

Mapping pipelined computations onto clusters and grids— DAG
[Taura et al.], DataCutter [Saltz et al.]

Energy-aware mapping of pipelined computations [Melhem et al.],
three-criteria optimization

Mapping pipelined computations onto special-purpose architectures—
FPGA arrays [Fabiani et al.]. Fault-tolerance for embedded systems [Zhu et al.]

Mapping skeletons onto clusters and grids— Use of stochastic
process algebra [Benoit et al.]
Conclusion

**Theoretical side** – Complexity results for several cases

Solid theoretical foundation for study of single/bi-criteria mappings, with possibility to replicate and data-parallelize application stages

**Practical side**

- Optimal polynomial algorithms, heuristics for NP-hard instances of the problem
- Experiments: Comparison of heuristics performance
- Linear program to assess the absolute performance of the heuristics, which turns out to be quite good
Future work

Short term

- **Heuristics** for *Fully Heterogeneous* platforms and other NP-hard instances of the problem
- Extension to **DAG-trees** (a DAG which is a tree when un-oriented)

Longer term

- **Heuristics** based on our polynomial algorithms for general application graphs structured as combinations of pipeline and fork kernels
- **Real experiments** on heterogeneous clusters, using an already-implemented skeleton library and MPI
- **Comparison** of effective performance against theoretical performance
Open problems

- Replication for fault-tolerance vs replication for parallelism
  - compute several time the same data-set in case of failure
  - uses more resources and does not decrease period or latency
  - increases robustness

- **Energy savings**
  - processors that can run at different frequencies
  - trade-off between energy consumption and speed

- Simultaneous execution of several (concurrent) workflows
  - competition for CPU and network resources
  - fairness between applications (stretch)
  - sensitivity to application/platform parameter changes