Complexity Results
for Throughput and Latency Optimization
of Replicated and Data-parallel Workflows

Anne Benoit and Yves Robert

GRAAL team, LIP
École Normale Supérieure de Lyon

September 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 skeleton workflows (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 skeleton workflows (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 skeleton workflows (pipeline, fork)
on to 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 skeleton workflows (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 skeleton workflows (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 skeleton workflows (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 skeleton workflows (pipeline, fork) onto heterogeneous platforms
Rule of the game

- Map each pipeline stage on a single processor (*extended later*: replication and data-parallelism)
- Goal: minimize execution time (*extended later*: throughput and latency)
- Several mapping strategies

The pipeline application
Rule of the game

- Map each pipeline stage on a single processor
  (extended later: replication and data-parallelism)
- Goal: minimize execution time
  (extended later: throughput and latency)
- Several mapping strategies

The pipeline application

\[
S_1 \rightarrow S_2 \rightarrow \cdots \rightarrow S_k \rightarrow \cdots \rightarrow S_n
\]
Rule of the game

- Map each pipeline stage on a single processor
  (*extended later: replication and data-parallelism*)

- Goal: minimize execution time
  (*extended later: throughput and latency*)

- Several mapping strategies

One-to-one Mapping
Rule of the game

- Map each pipeline stage on a single processor
  (*extended later: replication and data-parallelism*)

- Goal: minimize execution time
  (*extended later: throughput and latency*)

- 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: replication and data-parallelism)

- Goal: minimize execution time
  (extended later: throughput and latency)

- Several mapping strategies
Major contributions

**Theory**  Formal approach to the problem
Definition of replication and data-parallelism (stages on several processors)
Consider several optimization criteria
→ Problem complexity for several cases

**Practice**  Wait for my next talk!
Major contributions

Theory
Formal approach to the problem
Definition of replication and data-parallelism (stages on several processors)
Consider several optimization criteria
→ Problem complexity for several cases

Practice
Wait for my next talk!
Major contributions

**Theory**
Formal approach to the problem
Definition of replication and data-parallelism (stages on several processors)
Consider several optimization criteria
→ Problem complexity for several cases

**Practice**
Wait for my next talk!
Outline

1. Framework
2. Working out an example
3. Complexity results
4. Conclusion
Outline

1. Framework
2. Working out an example
3. Complexity results
4. 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

- $p$ processors $P_u$, $1 \leq u \leq p$, 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

**NO COMMUNICATIONS**

*Homogeneous* – Identical processors \((s_u = s)\): typical parallel machines

*Heterogeneous* – Different-speed processors \((s_u \neq s_v)\), identical links since we do not consider communications \((b_{u,v} = b)\): networks of workstations, clusters
Different platforms

NO COMMUNICATIONS

**Homogeneous** – Identical processors \((s_u = s)\): typical parallel machines

**Heterogeneous** – Different-speed processors \((s_u \neq s_v)\), identical links since we do not consider communications \((b_{u,v} = b)\): networks of workstations, clusters
Rule of the game

- Consecutive data-sets fed into the workflow
- **Period** $T_{\text{period}} = \text{time interval between beginning of execution of two consecutive data sets (throughput}=1/T_{\text{period}}$)
- **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** $T_{\text{period}} =$ time interval between beginning of execution of two consecutive data sets (throughput $= 1/T_{\text{period}}$)
- **Latency** $T_{\text{latency}}(x) =$ 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.
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 $P_1, \ldots, P_q$

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

- $S_{k+1}$ 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 $P_1, \ldots, P_q$

$S_k (w = 16)$

⇒

$P_1 (s_1 = 2)$: ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ●
$P_2 (s_2 = 1)$: ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ●
$P_3 (s_3 = 1)$: ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ●

- Perfect sharing of the work
- Data-parallelize single stage only
Data-parallelize stage $S_k$ on $P_1, \ldots, P_q$

\begin{align*}
S_k \,(w = 16) & \quad P_1 \,(s_1 = 2) : \quad \bullet \bullet \bullet \bullet \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet \quad \Rightarrow \quad P_2 \,(s_2 = 1) : \quad \bullet \bullet \bullet \\
\bullet \bullet \bullet \bullet & \quad P_3 \,(s_3 = 1) : \quad \bullet \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 $I_j = [d_j, e_j]$ (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 $P_{\text{alloc}(j)}$

\[
T_{\text{period}} = \max_{1 \leq j \leq m} \frac{\sum_{i=d_j}^{e_j} w_i}{s_{\text{alloc}(j)}}
\]
\[
T_{\text{latency}} = \sum_{1 \leq j \leq m} \frac{\sum_{i=d_j}^{e_j} w_i}{s_{\text{alloc}(j)}},
\]
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 $P_{\text{alloc}(j)}$

\[
T_{\text{period}} = \max_{1 \leq j \leq m} \frac{\sum_{i=d_j}^{e_j} w_i}{s_{\text{alloc}(j)}} \quad T_{\text{latency}} = \sum_{1 \leq j \leq m} \frac{\sum_{i=d_j}^{e_j} w_i}{s_{\text{alloc}(j)}}
\]
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 $P_{\text{alloc}(j)}$

\[ T_{\text{period}} = \max_{1 \leq j \leq m} \frac{\sum_{i=d_j}^{e_j} w_i}{s_{\text{alloc}(j)}} \quad T_{\text{latency}} = \sum_{1 \leq j \leq m} \frac{\sum_{i=d_j}^{e_j} w_i}{s_{\text{alloc}(j)}} \]
No data-parallelism 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 $P_{q_1}, \ldots, P_{q_k}$:

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

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

- No data-parallelism 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 = $T_{period}$ of assigned processors
Cost = delay to traverse the interval
Replication and data-parallelism

- No data-parallelism 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

Anne.Benoit@ens-lyon.fr September 2007
Cost to replicate \( [S_i, S_j] \) on \( k \) processors \( P_{q_1}, \ldots, P_{q_k} \):

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

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 $P_{q_1}, \ldots, P_{q_k}$:

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

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.
Outline

1. Framework
2. Working out an example
3. Complexity results
4. 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?
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?
\[
T_{\text{period}} = 7, \ S_1 \rightarrow P_1, \ S_2S_3 \rightarrow P_2, \ S_4 \rightarrow P_3 \ (T_{\text{latency}} = 17)
\]

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

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

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

\[
\begin{align*}
14 & \quad 4 & \quad 2 & \quad 4 \\
\end{align*}
\]

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

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

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

**Optimal period?**

\[ S_1 \xrightarrow{\text{DP}} P_1 P_2, \quad S_2 S_3 S_4 \xrightarrow{\text{REP}} P_3 P_4 \]

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

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

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

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

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

1. Framework
2. Working out an example
3. Complexity results
4. Conclusion
Complexity results

- Pipeline and fork graphs
- No communications
- Homogeneous or Heterogeneous platforms
- Interval Mapping only
- Replicable stages, and either data-parallelism or not
- Bi-criteria optimization
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>Poly (str)</td>
<td></td>
<td>Poly (DP)</td>
</tr>
<tr>
<td>Het. fork</td>
<td>NP-hard</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- $str =$ straightforward (map everything on the same proc...)
- $DP =$ dynamic programming
- $*$ = interesting case
Complexity results

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></td>
<td>Poly (DP)</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>

- str = straightforward (map everything on the same proc...)
- DP = dynamic programming
- * = 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></td>
<td>Poly (*)</td>
<td></td>
</tr>
<tr>
<td>Het. fork</td>
<td>NP-hard</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

- *str = straightforward (map everything on the same proc...)*
- *DP = dynamic programming*
- * = interesting case
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>

- str = straightforward (map everything on the same proc...)  
- DP = dynamic programming  
- * = interesting case
Complexity results

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></td>
<td>Poly (*)</td>
<td></td>
</tr>
<tr>
<td>Het. fork</td>
<td>NP-hard</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>Poly (*)</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Het. fork</td>
<td>NP-hard</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 \( P_1, \ldots, P_q \), 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
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 $K$.

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

- 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 \( K \).

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

Initialization:

\[
L(1, i, j) = \begin{cases} \frac{w}{s_i} & \text{if } \frac{w}{(j-i)s_i} \leq K \\ +\infty & \text{otherwise} \end{cases}
\]

\[
L(m, i, i) = \begin{cases} \frac{w_{m'}}{s_i} & \text{if } \frac{w_{m'}}{s_i} \leq K \\ +\infty & \text{otherwise} \end{cases}
\]
Dynamic programming algorithm

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

$$L(m, i, j) = \min_{1 \leq m' < m} \begin{cases} \frac{m \cdot w}{s_i} & \text{if } \frac{m \cdot w}{(j-i) \cdot s_i} \leq K \quad (1) \\ L(m', i, k) + L(m - m', k + 1, j) & (2) \end{cases}$$

- **Complexity** of the dynamic programming: $O(n^2 \cdot p^4)$
- 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. Complexity results
4. Conclusion
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. Some heuristics on particular cases (stay for next talk 😊).

Future work – Heuristics based on our polynomial algorithms for general application graphs structured as combinations of pipeline and fork kernels. Lots of open problems.
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.]