Mapping skeleton workflows onto heterogeneous platforms

Anne Benoit, Veronika Rehn, Yves Robert

GRAAL team, LIP
École Normale Supérieure de Lyon

December 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

- Workflow: several consecutive data-set enter pipeline
- Map each pipeline stage on a single processor (extended later)
- Goal: maximize throughput (extended later)

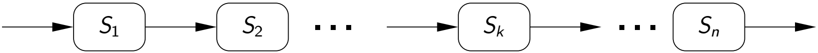
- Several mapping strategies

The pipeline application

Rule of the game

- Workflow: several consecutive data-set enter pipeline
- Map each pipeline stage on a single processor (extended later)
- Goal: maximize throughput (extended later)

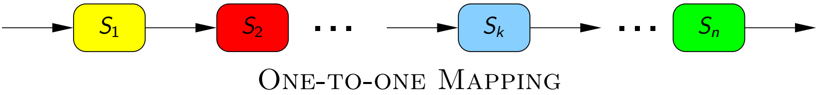
- Several mapping strategies



The pipeline application

Rule of the game

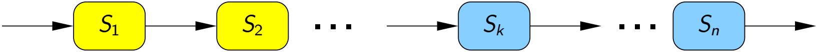
- Workflow: several consecutive data-set enter pipeline
- Map each pipeline stage on a single processor (extended later)
- Goal: maximize throughput (extended later)
- Several mapping strategies



Rule of the game

- Workflow: several consecutive data-set enter pipeline
- Map each pipeline stage on a single processor (extended later)
- Goal: maximize throughput (extended later)

- Several mapping strategies

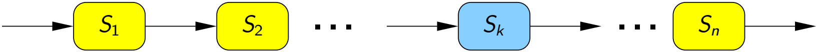


INTERVAL MAPPING

Rule of the game

- Workflow: several consecutive data-set enter pipeline
- Map each pipeline stage on a single processor (extended later)
- Goal: maximize throughput (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

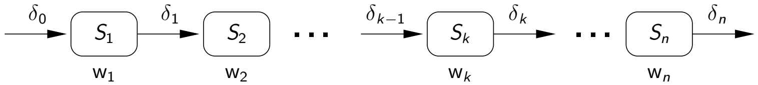
Practice Heuristics for INTERVAL MAPPING on clusters
Experiments to compare heuristics and evaluate their absolute performance

Outline

Outline

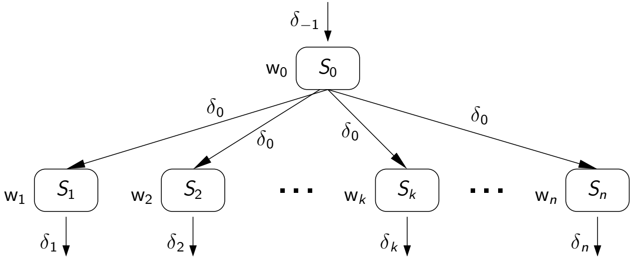
1. Framework
2. Working out an example
3. Part 1 - Communications, monolithic stages, mono-criterion
4. Part 2 - Simpler model with no communications, but with replication/DP and bi-criteria
5. Conclusion

The application: pipeline graphs



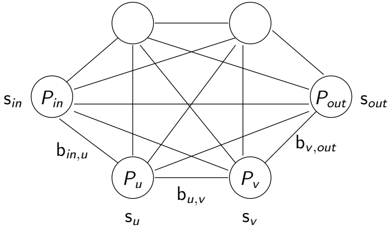
- n stages $\mathcal{S}_k$, $1 \leq k \leq$ n
- $\mathcal{S}_k$:
  - receives input of size δ_{k-1} from $\mathcal{S}_{k-1}$
  - performs w_k computations
  - outputs data of size δ_k to $\mathcal{S}_{k+1}$

The application: fork graphs



- n + 1 stages $\mathcal{S}_k$, $0 \leq k \leq$ n
  - $\mathcal{S}_0$: root stage
  - $\mathcal{S}_1$ to $\mathcal{S}_n$: independent stages
- A data-set goes through stage $\mathcal{S}_0$, 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

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_{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_{period} or T_{latency} or bi-criteria minimization

Rule of the game

- Consecutive data-sets fed into the workflow
- Period T_{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_{period} or T_{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 $\mathcal{S}_k$ on $P_1, \ldots, P_q$

$$\ldots\ \mathcal{S}_{k-1} \;--\; \begin{array}{l} \mathcal{S}_k \text{ on } P_1\text{: data sets } \mathbf{1, 4, 7, \ldots} \\ \mathcal{S}_k \text{ on } P_2\text{: data sets } \mathbf{2, 5, 8, \ldots} \\ \mathcal{S}_k \text{ on } P_3\text{: data sets } \mathbf{3, 5, 9, \ldots} \end{array} \;--\; \mathcal{S}_{k+1}\ \ldots$$

- 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 $\mathcal{S}_k$ on $P_1, \ldots, P_q$

$$
\ldots\ \mathcal{S}_{k-1}\ \begin{array}{c} / \\ -- \\ \backslash \end{array}\ \begin{array}{l} \mathcal{S}_k \text{ on } P_1\text{: data sets } \mathbf{1, 4, 7, \ldots} \\ \mathcal{S}_k \text{ on } P_2\text{: data sets } \mathbf{2, 5, 8, \ldots} \\ \mathcal{S}_k \text{ on } P_3\text{: data sets } \mathbf{3, 5, 9, \ldots} \end{array}\ \begin{array}{c} \backslash \\ -- \\ / \end{array}\ \mathcal{S}_{k+1}\ \ldots
$$

- 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

Data-parallelism

Data-parallelize stage $\mathcal{S}_k$ on $P_1, \ldots, P_q$

$\mathcal{S}_k \; (w = 16)$

● ● ● ●
● ● ● ●
● ● ● ●
● ● ● ●

$\Rightarrow$

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

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

Data-parallelism

Data-parallelize stage $\mathcal{S}_k$ on $P_1, \ldots, P_q$

$$
\begin{array}{c} \mathcal{S}_k\ (w = 16) \\ \bullet\bullet\bullet\bullet \\ \bullet\bullet\bullet\bullet \\ \bullet\bullet\bullet\bullet \\ \bullet\bullet\bullet\bullet \end{array}
\quad \Rightarrow \quad
\begin{array}{l}
\color{green}{P_1\ (s_1 = 2) : \bullet\bullet\bullet\bullet\bullet\bullet\bullet\bullet} \\
\color{blue}{P_2\ (s_2 = 1) : \bullet\bullet\bullet\bullet} \\
\color{purple}{P_3\ (s_3 = 1) : \bullet\bullet\bullet\bullet}
\end{array}
$$

- 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 $P_{\text{alloc}(j)}$

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

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

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 $e_m = \text{n}$)
- Interval I_j mapped onto processor $P_{\text{alloc}(j)}$

$$T_{\text{period}} = \max_{1 \leq j \leq m} \left\{ \frac{\delta_{d_j - 1}}{\text{b}_{\text{alloc}(j-1),\text{alloc}(j)}} + \frac{\sum_{i=d_j}^{e_j} \text{w}_i}{\text{s}_{\text{alloc}(j)}} + \frac{\delta_{e_j}}{\text{b}_{\text{alloc}(j),\text{alloc}(j+1)}} \right\}$$

$$T_{\text{latency}} = \sum_{1 \leq j \leq m} \left\{ \frac{\delta_{d_j - 1}}{\text{b}_{\text{alloc}(j-1),\text{alloc}(j)}} + \frac{\sum_{i=d_j}^{e_j} \text{w}_i}{\text{s}_{\text{alloc}(j)}} \right\} + \frac{\delta_{\text{n}}}{\text{b}_{\text{alloc}(m),\text{alloc}(m+1)}}$$

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 $e_m = \text{n}$)
- Interval I_j mapped onto processor $P_{\text{alloc}(j)}$

$$T_{\text{period}} = \max_{1 \leq j \leq m} \left\{ \frac{\delta_{d_j - 1}}{\text{b}_{\text{alloc}(j-1),\text{alloc}(j)}} + \frac{\sum_{i=d_j}^{e_j} \text{w}_i}{\text{s}_{\text{alloc}(j)}} + \frac{\delta_{e_j}}{\text{b}_{\text{alloc}(j),\text{alloc}(j+1)}} \right\}$$

$$T_{\text{latency}} = \sum_{1 \leq j \leq m} \left\{ \frac{\delta_{d_j - 1}}{\text{b}_{\text{alloc}(j-1),\text{alloc}(j)}} + \frac{\sum_{i=d_j}^{e_j} \text{w}_i}{\text{s}_{\text{alloc}(j)}} \right\} + \frac{\delta_{\text{n}}}{\text{b}_{\text{alloc}(m),\text{alloc}(m+1)}}$$

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 $e_m = \mathrm{n}$)
- Interval I_j mapped onto processor $P_{\mathrm{alloc}(j)}$

$$T_{\mathrm{period}} = \max_{1 \leq j \leq m} \left\{ \frac{\delta_{d_j - 1}}{\mathrm{b}_{\mathrm{alloc}(j-1),\mathrm{alloc}(j)}} + \frac{\sum_{i=d_j}^{e_j} \mathrm{w}_i}{\mathrm{s}_{\mathrm{alloc}(j)}} + \frac{\delta_{e_j}}{\mathrm{b}_{\mathrm{alloc}(j),\mathrm{alloc}(j+1)}} \right\}$$

$$T_{\mathrm{latency}} = \sum_{1 \leq j \leq m} \left\{ \frac{\delta_{d_j - 1}}{\mathrm{b}_{\mathrm{alloc}(j-1),\mathrm{alloc}(j)}} + \frac{\sum_{i=d_j}^{e_j} \mathrm{w}_i}{\mathrm{s}_{\mathrm{alloc}(j)}} \right\} + \frac{\delta_{\mathrm{n}}}{\mathrm{b}_{\mathrm{alloc}(m),\mathrm{alloc}(m+1)}}$$

Simpler problem, replication and data-parallelism

- No communication costs nor overheads
- Cost to execute $\mathcal{S}_i$ on P_u alone: $\frac{w_i}{s_u}$
- Cost to data-parallelize $[\mathcal{S}_i, \mathcal{S}_j]$ ($i = j$ 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

Simpler problem, replication and data-parallelism

- No communication costs nor overheads
- Cost to execute $\mathcal{S}_i$ on P_u alone: $\frac{w_i}{s_u}$
- Cost to data-parallelize $[\mathcal{S}_i, \mathcal{S}_j]$ ($i = j$ 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 $\mathcal{S}_i$ on P_u alone: $\frac{w_i}{s_u}$
- Cost to data-parallelize $[\mathcal{S}_i, \mathcal{S}_j]$ ($i = j$ 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

Simpler problem, replication and data-parallelism

- Cost to replicate $[\mathcal{S}_i, \mathcal{S}_j]$ on k processors $P_{q_1}, \ldots, P_{q_k}$:

$$\frac{\sum_{\ell=i}^{j} \mathsf{w}_\ell}{k \times \min_{1 \leq u \leq k} \mathsf{s}_{q_u}}.$$

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

$$t_{\max} = \frac{\sum_{\ell=i}^{j} \mathsf{w}_\ell}{\min_{1 \leq u \leq k} \mathsf{s}_{q_u}}$$

- With these formulas: easy to compute T_{period} and T_{latency} for pipeline graphs

Simpler problem, replication and data-parallelism

- Cost to replicate $[\mathcal{S}_i, \mathcal{S}_j]$ on k processors $P_{q_1}, \ldots, P_{q_k}$:

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

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

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

- With these formulas: easy to compute T_{period} and T_{latency} for pipeline graphs

Objective function?

Mono-criterion

- Minimize T_{period}
- Minimize T_{latency}

Bi-criteria

- How to define it?
  Minimize $\alpha.T_{\text{period}} + \beta.T_{\text{latency}}$?
- Values which are not comparable

- Minimize T_{period} for a fixed latency
- Minimize T_{latency} for a fixed period

Objective function?

Mono-criterion

- Minimize T_{period}
- Minimize T_{latency}

Bi-criteria

- How to define it?
  Minimize $\alpha.T_{\text{period}} + \beta.T_{\text{latency}}$?
- Values which are not comparable

- Minimize T_{period} for a fixed latency
- Minimize T_{latency} for a fixed period

Objective function?

Mono-criterion

- Minimize T_{period}
- Minimize T_{latency}

Bi-criteria

- How to define it?
  Minimize $\alpha.T_{\text{period}} + \beta.T_{\text{latency}}$?
- Values which are not comparable

- Minimize T_{period} for a fixed latency
- Minimize T_{latency} for a fixed period

Objective function?

Mono-criterion

- Minimize T_{period}
- Minimize T_{latency}

Bi-criteria

- How to define it?
  Minimize $\alpha.T_{\text{period}} + \beta.T_{\text{latency}}$?
- Values which are not comparable

- Minimize T_{period} for a fixed latency
- Minimize T_{latency} for a fixed period

Outline

Working out an example

$$\begin{array}{ccccccc} \mathcal{S}_1 & \longrightarrow & \mathcal{S}_2 & \longrightarrow & \mathcal{S}_3 & \longrightarrow & \mathcal{S}_4 \\ 14 & & 4 & & 2 & & 4 \end{array}$$

Interval mapping, 4 processors, $s_1 = 2$ and $s_2 = s_3 = s_4 = 1$

Optimal period?

Working out an example

$$\begin{array}{ccccccc} \mathcal{S}_1 & \rightarrow & \mathcal{S}_2 & \rightarrow & \mathcal{S}_3 & \rightarrow & \mathcal{S}_4 \\ 14 & & 4 & & 2 & & 4 \end{array}$$

Interval mapping, 4 processors, $s_1 = 2$ and $s_2 = s_3 = s_4 = 1$

Optimal period?

$T_{\text{period}} = 7$, $\mathcal{S}_1 \rightarrow P_1$, $\mathcal{S}_2\mathcal{S}_3 \rightarrow P_2$, $\mathcal{S}_4 \rightarrow P_3$ ($T_{\text{latency}} = 17$)

Optimal latency?

Working out an example

$$\begin{array}{ccccccc} \mathcal{S}_1 & \rightarrow & \mathcal{S}_2 & \rightarrow & \mathcal{S}_3 & \rightarrow & \mathcal{S}_4 \\ 14 & & 4 & & 2 & & 4 \end{array}$$

Interval mapping, 4 processors, $s_1 = 2$ and $s_2 = s_3 = s_4 = 1$

Optimal period?

$T_{\text{period}} = 7$, $\mathcal{S}_1 \rightarrow P_1$, $\mathcal{S}_2\mathcal{S}_3 \rightarrow P_2$, $\mathcal{S}_4 \rightarrow P_3$ ($T_{\text{latency}} = 17$)

Optimal latency?

$T_{\text{latency}} = 12$, $\mathcal{S}_1\mathcal{S}_2\mathcal{S}_3\mathcal{S}_4 \rightarrow P_1$ ($T_{\text{period}} = 12$)

Min. latency if $T_{\text{period}} \leq 10$?

Working out an example

$$\begin{array}{ccccccc} \mathcal{S}_1 & \rightarrow & \mathcal{S}_2 & \rightarrow & \mathcal{S}_3 & \rightarrow & \mathcal{S}_4 \\ 14 & & 4 & & 2 & & 4 \end{array}$$

Interval mapping, 4 processors, $s_1 = 2$ and $s_2 = s_3 = s_4 = 1$

Optimal period?

$T_{\text{period}} = 7$, $\mathcal{S}_1 \rightarrow P_1$, $\mathcal{S}_2\mathcal{S}_3 \rightarrow P_2$, $\mathcal{S}_4 \rightarrow P_3$ ($T_{\text{latency}} = 17$)

Optimal latency?

$T_{\text{latency}} = 12$, $\mathcal{S}_1\mathcal{S}_2\mathcal{S}_3\mathcal{S}_4 \rightarrow P_1$ ($T_{\text{period}} = 12$)

Min. latency if $T_{\text{period}} \leq 10$?

$T_{\text{latency}} = 14$, $\mathcal{S}_1\mathcal{S}_2\mathcal{S}_3 \rightarrow P_1$, $\mathcal{S}_4 \rightarrow P_2$

Example with replication and data-parallelism

$$\begin{array}{ccccccc} \mathcal{S}_1 & \rightarrow & \mathcal{S}_2 & \rightarrow & \mathcal{S}_3 & \rightarrow & \mathcal{S}_4 \\ 14 & & 4 & & 2 & & 4 \end{array}$$

Interval mapping, 4 processors, $s_1 = 2$ and $s_2 = s_3 = s_4 = 1$

Replicate interval $[S_u..\mathcal{S}_v]$ on $P_1, \ldots, P_q$

$$\begin{array}{ccccc} & \diagup & \mathcal{S}_u \ldots \mathcal{S}_v \text{ on } P_1\text{: data sets } \mathbf{1, 4, 7, \ldots} & \diagdown & \\ \ldots\ \mathcal{S} & -- & \mathcal{S}_u \ldots \mathcal{S}_v \text{ on } P_2\text{: data sets } \mathbf{2, 5, 8, \ldots} & -- & \mathcal{S}\ \ldots \\ & \diagdown & \mathcal{S}_u \ldots \mathcal{S}_v \text{ on } P_3\text{: data sets } \mathbf{3, 5, 9, \ldots} & \diagup & \end{array}$$

$$T_{\text{period}} = \frac{\sum_{k=u}^{v} w_k}{q \times \min_i(s_i)} \text{ and } T_{\text{latency}} = q \times T_{\text{period}}$$

Example with replication and data-parallelism

$$\begin{array}{ccccccc} \mathcal{S}_1 & \rightarrow & \mathcal{S}_2 & \rightarrow & \mathcal{S}_3 & \rightarrow & \mathcal{S}_4 \\ 14 & & 4 & & 2 & & 4 \end{array}$$

Interval mapping, 4 processors, $s_1 = 2$ and $s_2 = s_3 = s_4 = 1$

Data Parallelize single stage $\mathcal{S}_k$ on $P_1, \ldots, P_q$

$\mathcal{S}\ (w = 16)$

• • • •
• • • •
• • • •
• • • •

$\Rightarrow$

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

$$T_{\text{period}} = \frac{w_k}{\sum_{i=1}^{q} s_i} \text{ and } T_{\text{latency}} = T_{\text{period}}$$

Example with replication and data-parallelism

$$\begin{array}{ccccccc} \mathcal{S}_1 & \rightarrow & \mathcal{S}_2 & \rightarrow & \mathcal{S}_3 & \rightarrow & \mathcal{S}_4 \\ 14 & & 4 & & 2 & & 4 \end{array}$$

Interval mapping, 4 processors, $s_1 = 2$ and $s_2 = s_3 = s_4 = 1$

Optimal period?

Example with replication and data-parallelism

$$\begin{array}{ccccccc} \mathcal{S}_1 & \rightarrow & \mathcal{S}_2 & \rightarrow & \mathcal{S}_3 & \rightarrow & \mathcal{S}_4 \\ 14 & & 4 & & 2 & & 4 \end{array}$$

Interval mapping, 4 processors, $s_1 = 2$ and $s_2 = s_3 = s_4 = 1$

Optimal period?

$$\mathcal{S}_1 \overset{\text{DP}}{\longrightarrow} P_1P_2,\ \mathcal{S}_2\mathcal{S}_3\mathcal{S}_4 \overset{\text{REP}}{\longrightarrow} P_3P_4$$

$$T_{\text{period}} = \max(\tfrac{14}{2+1}, \tfrac{4+2+4}{2\times 1}) = 5,\ T_{\text{latency}} = 14.67$$

Example with replication and data-parallelism

$$\begin{array}{ccccccc} \mathcal{S}_1 & \rightarrow & \mathcal{S}_2 & \rightarrow & \mathcal{S}_3 & \rightarrow & \mathcal{S}_4 \\ 14 & & 4 & & 2 & & 4 \end{array}$$

Interval mapping, 4 processors, $s_1 = 2$ and $s_2 = s_3 = s_4 = 1$

Optimal period?

$$\mathcal{S}_1 \overset{\text{DP}}{\rightarrow} P_1 P_2,\ \mathcal{S}_2\mathcal{S}_3\mathcal{S}_4 \overset{\text{REP}}{\rightarrow} P_3 P_4$$

$$T_{\text{period}} = \max\left(\tfrac{14}{2+1}, \tfrac{4+2+4}{2\times 1}\right) = 5,\ T_{\text{latency}} = 14.67$$

$$\mathcal{S}_1 \overset{\text{DP}}{\rightarrow} P_2 P_3 P_4,\ \mathcal{S}_2\mathcal{S}_3\mathcal{S}_4 \rightarrow P_1$$

$T_{\text{period}} = \max\left(\tfrac{14}{1+1+1}, \tfrac{4+2+4}{2}\right) = 5,\ T_{\text{latency}} = 9.67$ (optimal)

Outline

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

	Fully Hom.	Comm. Hom.
One-to-one Mapping		
Interval Mapping		
General Mapping		

Complexity results

	Fully Hom.	Comm. Hom.
One-to-one Mapping	polynomial	polynomial
Interval Mapping		
General Mapping		

- Binary search polynomial algorithm for ONE-TO-ONE MAPPING
-
-
-

Complexity results

	Fully Hom.	Comm. Hom.
One-to-one Mapping	polynomial	polynomial
Interval Mapping	polynomial	NP-complete
General Mapping		

- Binary search polynomial algorithm for One-to-one Mapping
- Dynamic programming algorithm for Interval Mapping on Hom. platforms (NP-hard otherwise)
-
-

Complexity results

	Fully Hom.	**Comm. Hom.**
One-to-one Mapping	polynomial	polynomial
Interval Mapping	polynomial	NP-complete
General Mapping	same complexity as Interval	

- 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

	Fully Hom.	**Comm. Hom.**
One-to-one Mapping	polynomial	polynomial
Interval Mapping	polynomial	NP-complete
General Mapping	same complexity as Interval	

- 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 $\mathcal{S}_1$ to $\mathcal{S}_n$ as free
- **For** $u = 1$ **to** n
  - Pick up any free stage $\mathcal{S}_k$ s.t. $\delta_{k-1}/b + w_k/s_u + \delta_k/b \leq T_{period}$
  - Assign $\mathcal{S}_k$ to P_u, and mark $\mathcal{S}_k$ 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 $\mathrm{s}_1 \leq \mathrm{s}_2 \leq \ldots \leq \mathrm{s}_\mathrm{n}$
- Mark all stages $\mathcal{S}_1$ to $\mathcal{S}_\mathrm{n}$ as free
- **For** $u = 1$ **to** n
  - Pick up any free stage $\mathcal{S}_k$ s.t. $\delta_{k-1}/\mathrm{b} + \mathrm{w}_k/\mathrm{s}_u + \delta_k/\mathrm{b} \leq T_\mathrm{period}$
  - Assign $\mathcal{S}_k$ to P_u, and mark $\mathcal{S}_k$ 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 $(\delta_{in} + \delta_{out})/\mathrm{b} + \sum_{i \in Interval} w_i/s_u$

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 $max(period(j), period(j'))$ 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

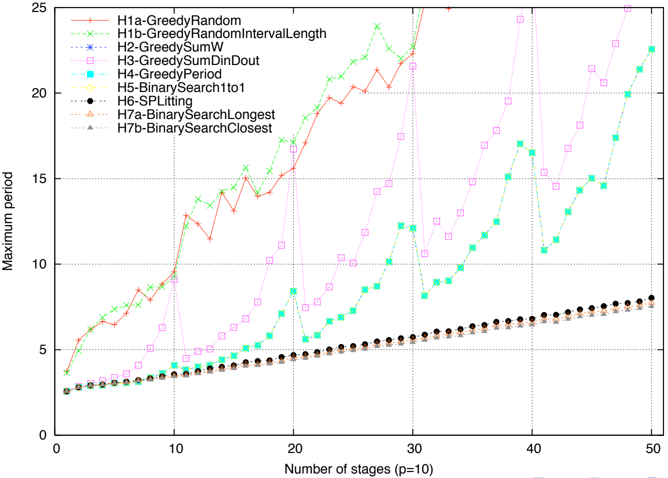
- Assess performance of polynomial heuristics
- Random applications, $n = 1$ to 50 stages
- Random platforms, $p = 10$ and $p = 100$ processors
- $b = 10$ (comm. hom.), proc. speed between 1 and 20
- Relevant parameters: ratios $\frac{\delta}{b}$ and $\frac{w}{s}$
- Average over 100 similar random appli/platform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

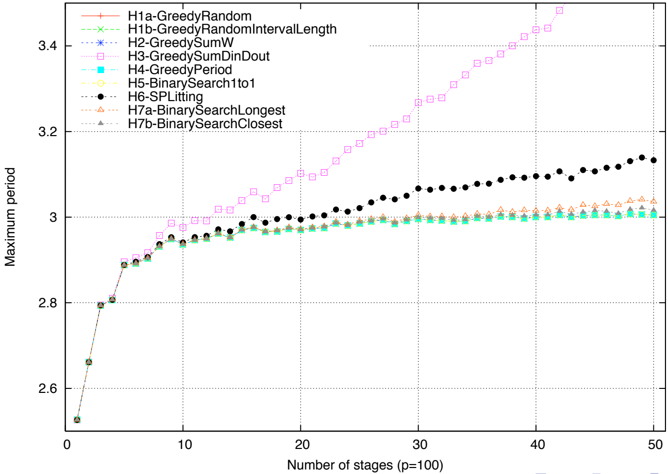
Experiment 1 - balanced comm/comp, hom comm

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



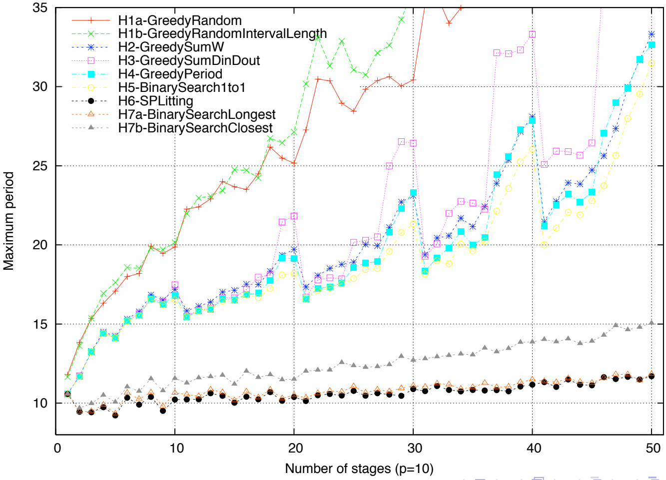
Experiment 1 - balanced comm/comp, hom comm

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



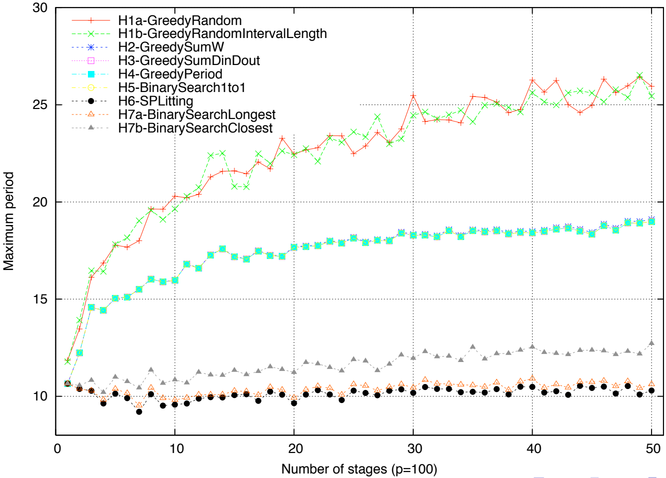
Experiment 2 - balanced comm/comp, het comm

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



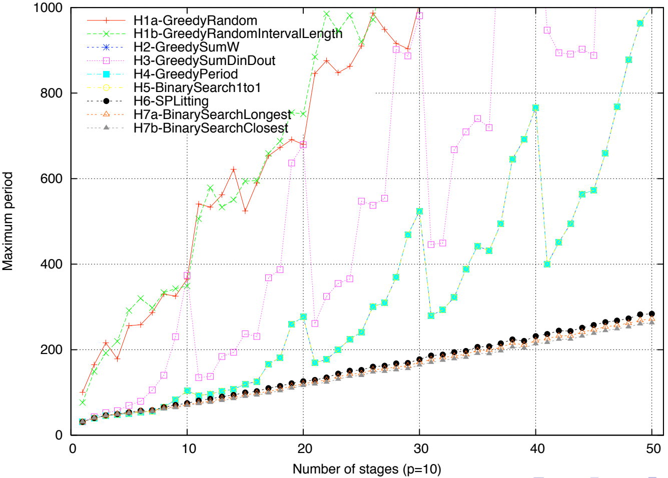
Experiment 2 - balanced comm/comp, het comm

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



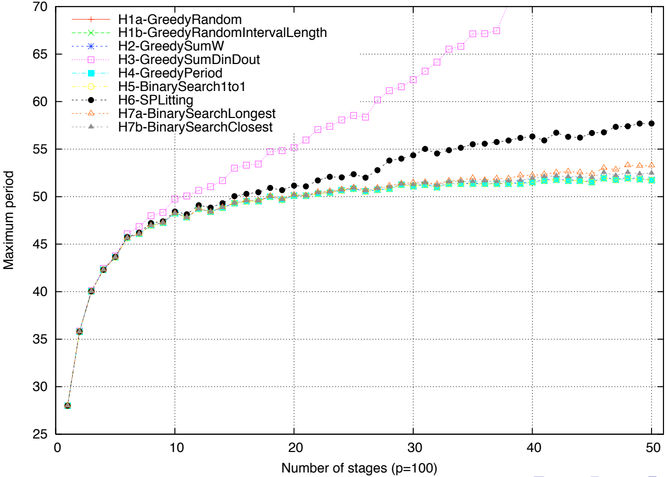
Experiment 3 - large computations

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



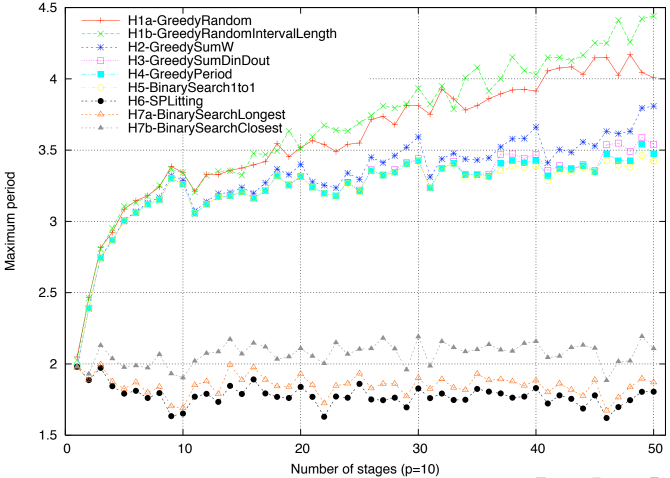
Experiment 3 - large computations

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



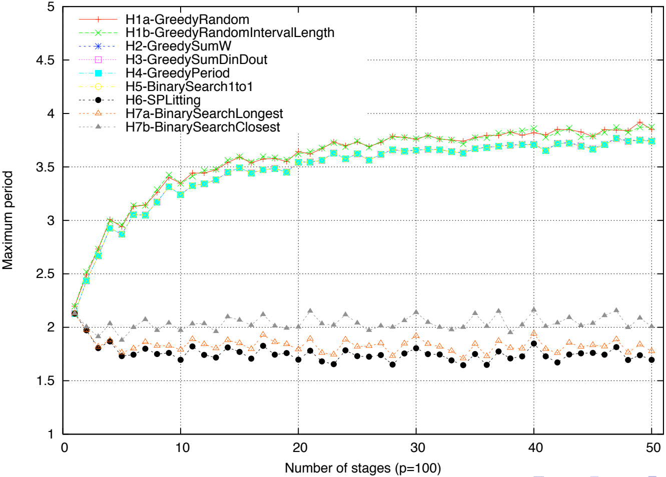
Experiment 4 - small computations

- 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

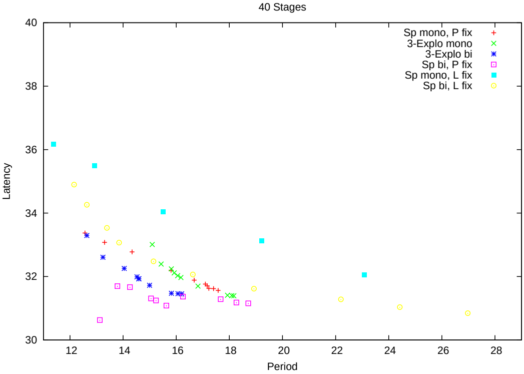
- Much more efficient than random mappings
- Three dominant heuristics for different cases
- Insignificant communications (hom. or small) and many processors: H5-BS121 (One-to-one Mapping)
- Insignificant communications (hom. or small) and few processors: H7b-BSC (binary search: clever choice where to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

Bi-criteria problem

- set of heuristics and experiments
- balanced comm/comp, het comm (Exp. 2)

100 processors.
40 stages.

☺ Sp bi P
☹ 3-Explo mono



Outline

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

Part 2

- 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>
<tr><th>Objective</th><th>period</th><th>latency</th><th>bi-criteria</th></tr>
<tr><td>Hom. pipeline</td><td colspan="3">-</td></tr>
<tr><td>Het. pipeline</td><td colspan="3">Poly (str)</td></tr>
<tr><td>Hom. fork</td><td>-</td><td colspan="2">Poly (DP)</td></tr>
<tr><td>Het. fork</td><td>Poly (str)</td><td colspan="2">NP-hard</td></tr>
</table>

Complexity results

With data-parallelism, *Homogeneous* platforms

Objective	period	latency	bi-criteria
Hom. pipeline	-		
Het. pipeline	Poly (DP)		
Hom. fork	-	Poly (DP)	
Het. fork	Poly (str)	NP-hard	

Complexity results

Without data-parallelism, *Heterogeneous* platforms

Objective	period	latency	bi-criteria
Hom. pipeline	Poly (*)	-	Poly (*)
Het. pipeline	NP-hard (**)	Poly (str)	NP-hard
Hom. fork	Poly (*)		
Het. fork	NP-hard	-	

Complexity results

With data-parallelism, *Heterogeneous* platforms

Objective	period	latency	bi-criteria
Hom. pipeline	NP-hard		
Het. pipeline	-		
Hom. fork	NP-hard		
Het. fork	-		

Complexity results

Most interesting case:

Without data-parallelism, *Heterogeneous* platforms

<table>
<tr><th>Objective</th><th>period</th><th>latency</th><th>bi-criteria</th></tr>
<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 colspan="3">Poly (*)</td></tr>
<tr><td>Het. fork</td><td>NP-hard</td><td colspan="2">-</td></tr>
</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

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

Dynamic programming algorithm

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

$$L(m,i,j) = \min_{\substack{1 \le m' < m \\ i \le k < j}} \begin{cases} \frac{m.\mathrm{w}}{\mathrm{s}_i} \quad \text{if } \frac{m.\mathrm{w}}{(j-i).\mathrm{s}_i} \le K & (1) \\ L(m', i, k) + L(m - m', k+1, j) & (2) \end{cases}$$

- Case (1): replicating m stages onto processors $P_i, \ldots, P_j$
- 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.

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

Initialization:

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

$$L(m,i,i) = \begin{cases} \frac{m.w}{s_i} & \text{if } \frac{m.w}{s_i} \leq K \\ +\infty & \text{otherwise} \end{cases}$$

Dynamic programming algorithm

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

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

- Complexity of the dynamic programming: $O(\mathrm{n}^2.\mathrm{p}^4)$
- 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

Also in the pipeline

Bi-criteria

- Several heuristics and experiments not detailed in this talk
- Bi-criteria linear program
- Real experiments on a JPEG encoder pipeline application

Three-criteria

- Introduction of failure probabilities to the model
- 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
  - three objectives: min latency and period, max reliability
- Complexity analysis

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

- 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