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)
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

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

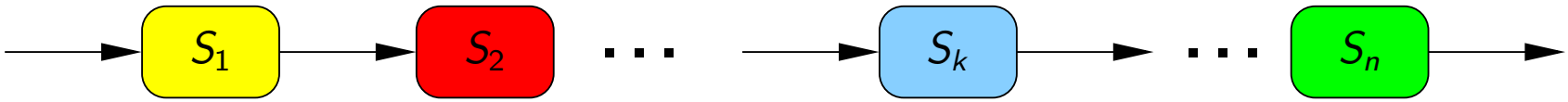
Rule of the game

- 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

Rule of the game

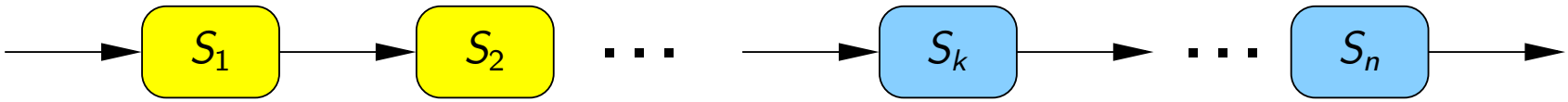
- Map each pipeline stage on a single processor
  (*extended later*: replication and data-parallelism)
- Goal: minimize execution time
  (*extended later*: throughput and latency)
- Several mapping strategies



One-to-one Mapping

Rule of the game

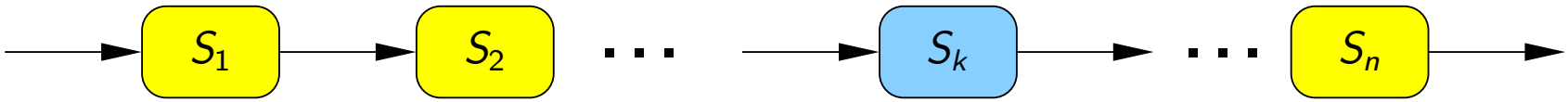
- Map each pipeline stage on a single processor
  (*extended later*: replication and data-parallelism)
- Goal: minimize execution time
  (*extended later*: throughput and latency)
- Several mapping strategies



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



General Mapping

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
$\longrightarrow$ 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
$\longrightarrow$ 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

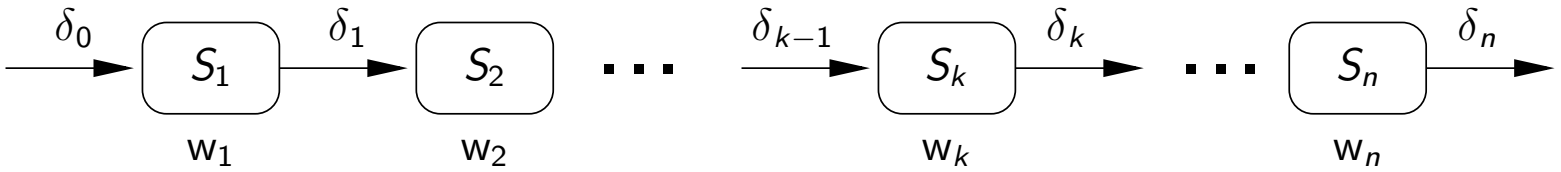
1 Framework

2 Working out an example

3 Complexity results

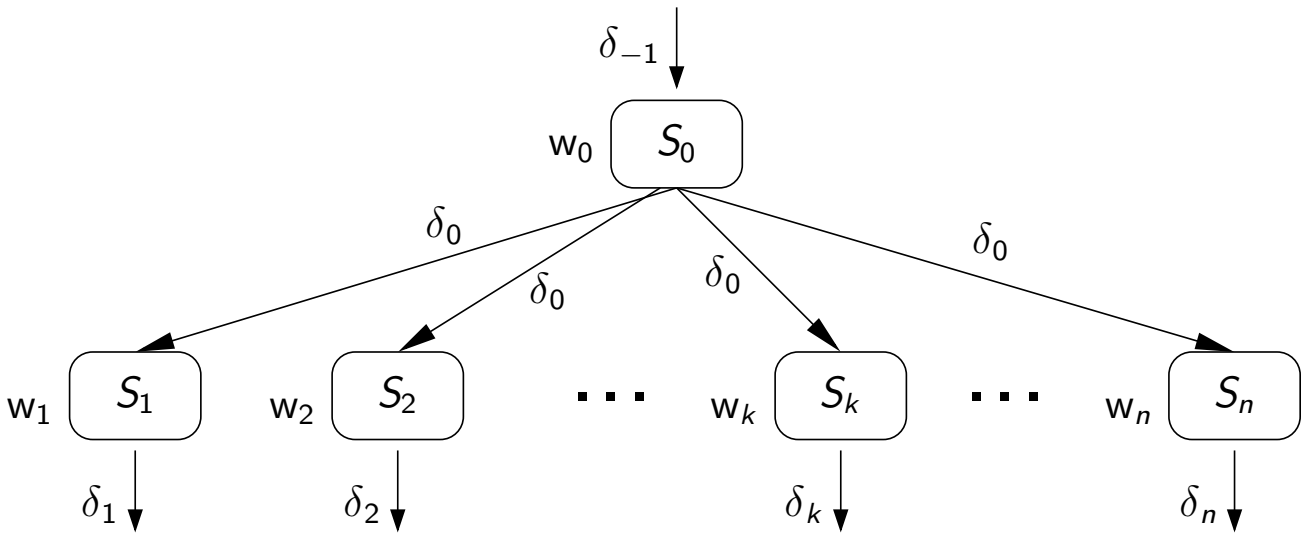
4 Conclusion

The application: pipeline graphs



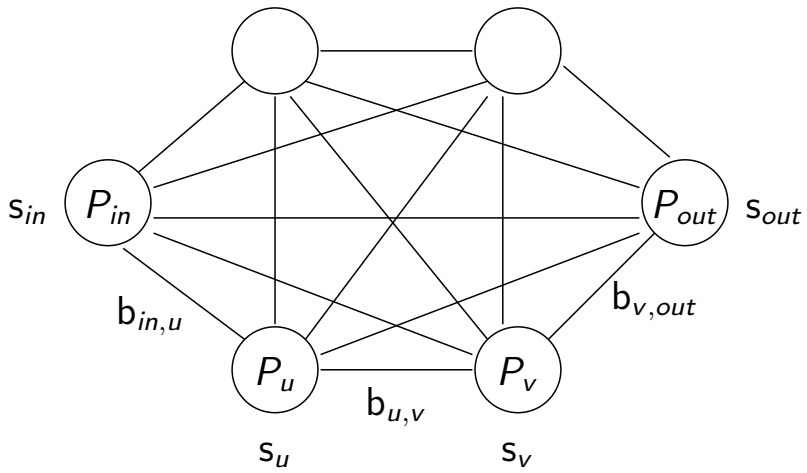
- n stages $\mathcal{S}_k$, $1 \leq k \leq n$
- $\mathcal{S}_k$:
  - receives input of size δ_{k-1} from $\mathcal{S}_{k-1}$
  - performs w_k computations
  - outputs data of size δ_k to $\mathcal{S}_{k+1}$

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 $link_{u,v} : P_u \longrightarrow P_v$, 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_{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

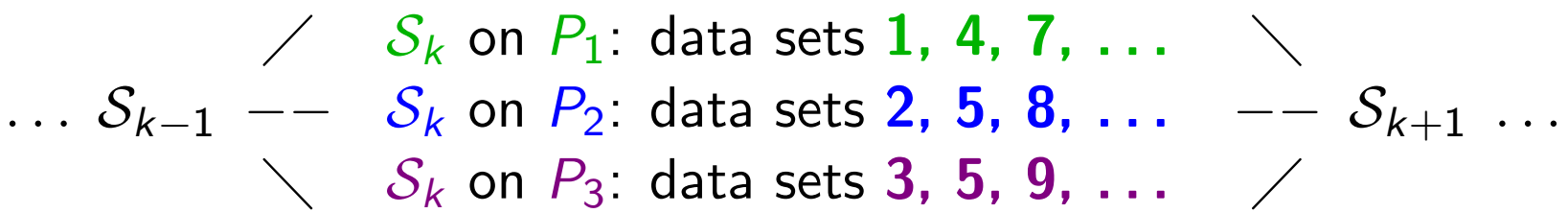
- Consecutive data-sets fed into the workflow
- Period T_{period} = time interval between beginning of execution of two consecutive data sets (throughput=$1/T_{\text{period}}$)
- Latency $T_{\text{latency}}(x)$ = time elapsed between beginning and end of execution for a given data set x, and $T_{\text{latency}} = \max_x T_{\text{latency}}(x)$
- 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

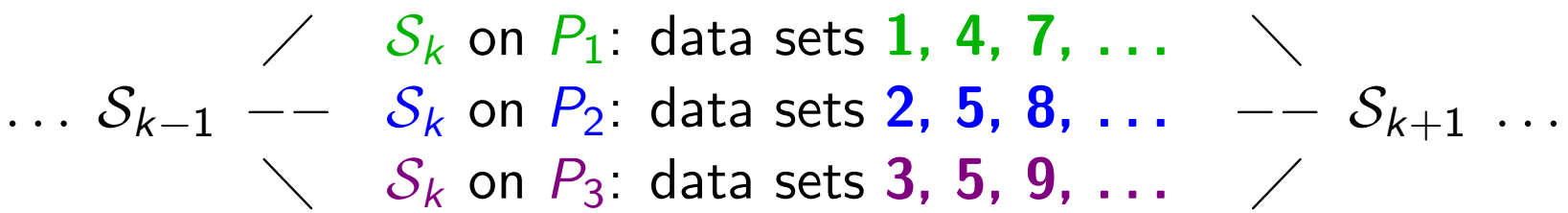
Replicate stage $\mathcal{S}_k$ on $P_1, \ldots, P_q$



- 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 $P_1, \ldots, P_q$

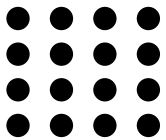


- S_{k+1} may be monolithic: output order must be respected
- Round-robin rule to ensure output order
- Cannot feed more fast processors than slow ones
- Most efficient with similar-speed processors

Data-parallelism

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

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



$\Rightarrow$

P_1 $(s_1 = 2)$:

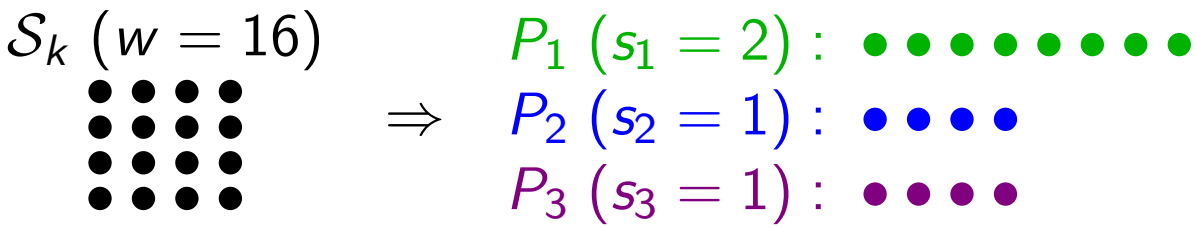
P_2 $(s_2 = 1)$:

P_3 $(s_3 = 1)$:

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

Data-parallelism

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



- 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 $P_{alloc(j)}$

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

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

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

Replication and data-parallelism

- No data-parallelism 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

Replication and data-parallelism

- No data-parallelism 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

Replication and data-parallelism

- No data-parallelism 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

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 $= T_{\text{period}}$ 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

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 $= T_{\text{period}}$ 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

Outline

1 Framework

2 Working out an example

3 Complexity results

4 Conclusion

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?

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

$\mathcal{S}_1$	$\rightarrow$	$\mathcal{S}_2$	$\rightarrow$	$\mathcal{S}_3$	$\rightarrow$	$\mathcal{S}_4$
14		4		2		4

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

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?

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

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

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?

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

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

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

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

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

Complexity results

Without data-parallelism, *Homogeneous* platforms

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

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

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	

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

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	-	

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

Complexity results

With data-parallelism, *Heterogeneous* platforms

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

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

Complexity results

Most interesting case:

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

Most interesting case:

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	-	

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 $P_1, ..., P_q$, 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

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.

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

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

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.

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

Initialization:

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

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

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.

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

- 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 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.]