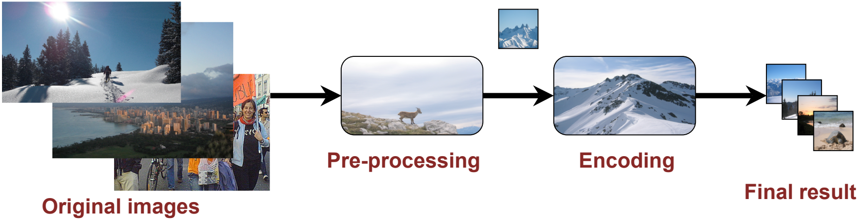
Scheduling pipelined applications: models, algorithms and complexity

Anne Benoit
GRAAL team, LIP
Ecole Normale Supérieure de Lyon, France

Habilitation à diriger des recherches
July 8, 2009

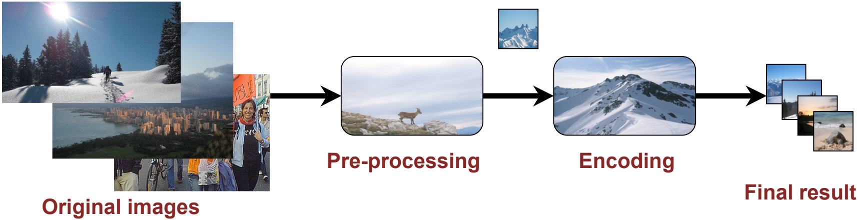
Scheduling pipelined applications: why?

- Stream of data to process: images, frames, matrices, etc.
- Encode images, factorize matrices
- Structured applications: several steps to process one data set
- Many processing resources: work on different data in parallel



Scheduling pipelined applications: why?

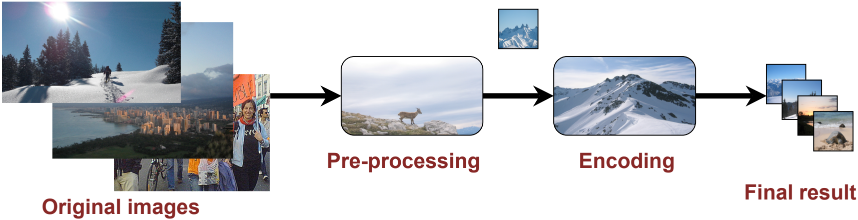
- Stream of data to process: images, frames, matrices, etc.
- Encode images, factorize matrices
- Structured applications: several steps to process one data set
- Many processing resources: work on different data in parallel



Large class of applications

Scheduling pipelined applications: why?

- Stream of data to process: images, frames, matrices, etc.
- Encode images, factorize matrices
- Structured applications: several steps to process one data set
- Many processing resources: work on different data in parallel



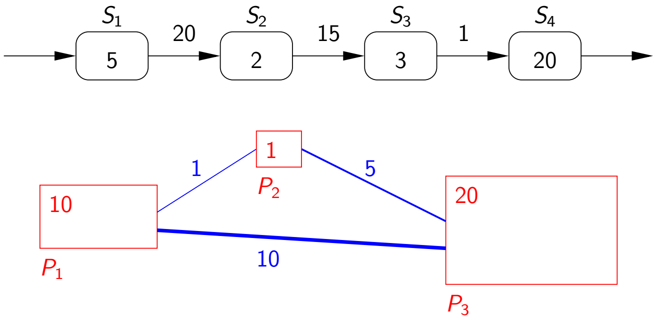
Large class of applications
Need to efficiently use computing resources

Motivating example

- 4 processing stages, 3 processors at our disposal
- Where/how can we execute the application?

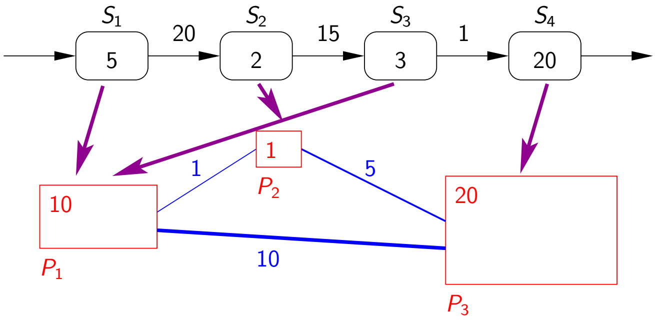
Motivating example

- 4 processing stages, 3 processors at our disposal
- Where/how can we execute the application?



Motivating example

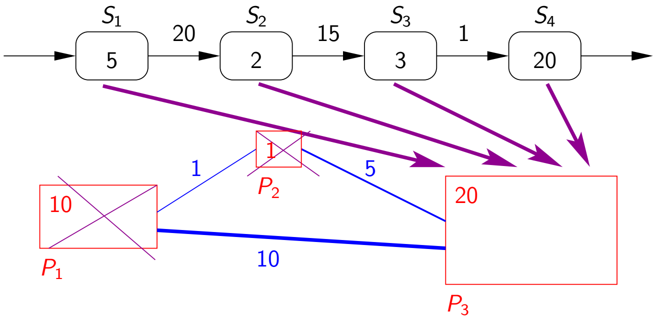
- 4 processing stages, 3 processors at our disposal
- Where/how can we execute the application?



- Use all resources greedily
- Many communications to pay, not efficient at all!

Motivating example

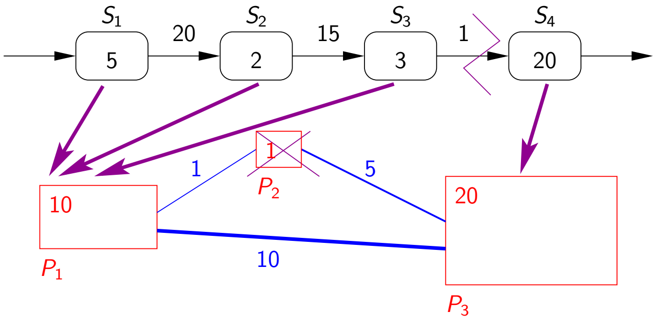
- 4 processing stages, 3 processors at our disposal
- Where/how can we execute the application?



- Everything on the fastest processor: no communications
- Optimal execution time to process one single data

Motivating example

- 4 processing stages, 3 processors at our disposal
- Where/how can we execute the application?



- Optimal throughput: processing of different data in parallel
- Resource selection: do not use the slowest processor

What is scheduling?

- Schedule an **application** onto a **computational platform**, with some **criteria** to optimize
- Target application: pipelined and structured
  - *Streaming* application (workflow, pipeline): several data sets are processed by a set of tasks (or pipeline stages)
  - *Structured* application: algorithmic skeletons, large class of applications build upon well-known paradigms, easier to program **and** to schedule
  - *Linear chain* application: linear dependencies between tasks
- Target platform: various models
  - Ranking from *fully homogeneous* to *fully heterogeneous*
  - Completely interconnected, subject to *failures*
  - Emphasis on different *communication models* (overlap or not, one- vs multi-port)

What is scheduling?

- Schedule an **application** onto a **computational platform**, with some **criteria** to optimize

- Target application: pipelined and structured
  - *Streaming* application (workflow, pipeline): several data sets are processed by a set of tasks (or pipeline stages)
  - *Structured* application: algorithmic skeletons, large class of applications build upon well-known paradigms, easier to program **and** to schedule
  - *Linear chain* application: linear dependencies between tasks

- Target platform: various models
  - Ranking from *fully homogeneous* to *fully heterogeneous*
  - Completely interconnected, subject to *failures*
  - Emphasis on different *communication models* (overlap or not, one- vs multi-port)

What is scheduling?

- Schedule an **application** onto a **computational platform**, with some **criteria** to optimize

- Target application: pipelined and structured
  - *Streaming* application (workflow, pipeline): several data sets are processed by a set of tasks (or pipeline stages)
  - *Structured* application: algorithmic skeletons, large class of applications build upon well-known paradigms, easier to program **and** to schedule
  - *Linear chain* application: linear dependencies between tasks

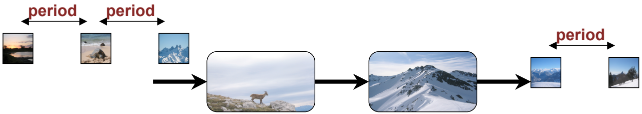
- Target platform: various models
  - Ranking from *fully homogeneous* to *fully heterogeneous*
  - Completely interconnected, subject to *failures*
  - Emphasis on different *communication models* (overlap or not, one- vs multi-port)

What is scheduling? ... the criteria

- Optimization criteria
  - *period* (inverse of throughput) and *latency* (execution time)
  - *reliability*, and also energy, stretch, ...
- Period $\mathcal{P}$: time interval between the beginning of execution of two consecutive data sets (inverse of throughput)
- Latency $\mathcal{L}$: maximal time elapsed between beginning and end of execution of a data set
- Reliability: inverse of $\mathcal{F}$, probability of failure of the application (i.e., some data sets will not be processed)

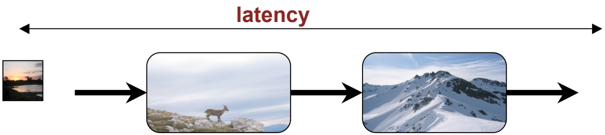
What is scheduling? ... the criteria

- Optimization criteria
  - *period* (inverse of throughput) and *latency* (execution time)
  - *reliability*, and also energy, stretch, ...
- Period $\mathcal{P}$: time interval between the beginning of execution of two consecutive data sets (inverse of throughput)
- Latency $\mathcal{L}$: maximal time elapsed between beginning and end of execution of a data set
- Reliability: inverse of $\mathcal{F}$, probability of failure of the application (i.e., some data sets will not be processed)



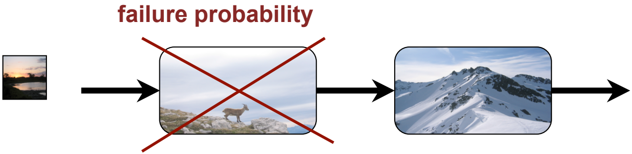
What is scheduling? ... the criteria

- Optimization criteria
  - *period* (inverse of throughput) and *latency* (execution time)
  - *reliability*, and also energy, stretch, ...
- Period $\mathcal{P}$: time interval between the beginning of execution of two consecutive data sets (inverse of throughput)
- Latency $\mathcal{L}$: maximal time elapsed between beginning and end of execution of a data set
- Reliability: inverse of $\mathcal{F}$, probability of failure of the application (i.e., some data sets will not be processed)



What is scheduling? ... the criteria

- Optimization criteria
  - *period* (inverse of throughput) and *latency* (execution time)
  - *reliability*, and also energy, stretch, ...
- Period $\mathcal{P}$: time interval between the beginning of execution of two consecutive data sets (inverse of throughput)
- Latency $\mathcal{L}$: maximal time elapsed between beginning and end of execution of a data set
- Reliability: inverse of $\mathcal{F}$, probability of failure of the application (i.e., some data sets will not be processed)



Outline

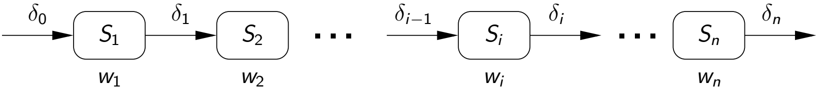
1. Models
   - Application model
   - Platform and communication models
2. Multi-criteria scheduling problems
   - Stage types and replication
   - Rule of the game
   - Optimization criteria
   - Define and classify problems
3. Complexity results
   - Mono-criterion problems
   - Bi-criteria problems
4. Conclusion

Outline

1. Models
   - Application model
   - Platform and communication models
2. Multi-criteria scheduling problems
   - Stage types and replication
   - Rule of the game
   - Optimization criteria
   - Define and classify problems
3. Complexity results
   - Mono-criterion problems
   - Bi-criteria problems
4. Conclusion

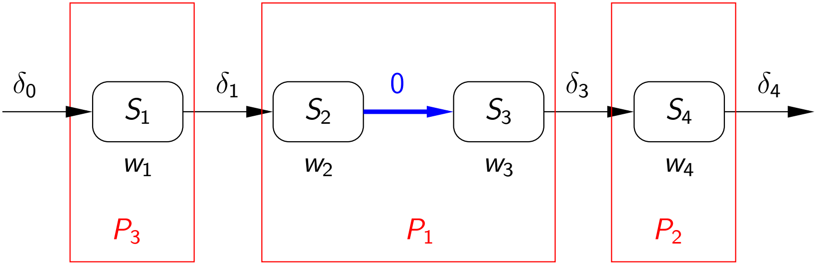
Application model

- Set of n application stages
- Computation cost of stage S_i: w_i
- Pipelined: each data set must be processed by all stages
- Linear dependencies between stages

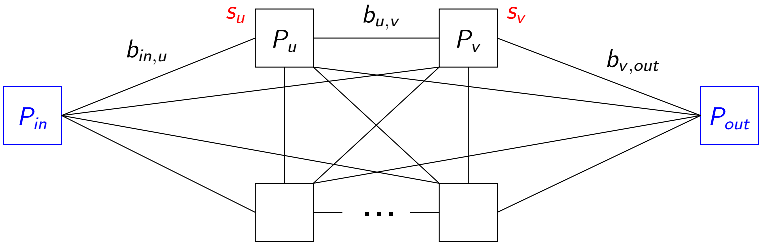


Application model: communication costs

- Two dependent stages $S_i \rightarrow S_{i+1}$:
  data must be transferred from S_i to S_{i+1}
- Fixed data size δ_i, communication cost to pay only if S_i and S_{i+1} are mapped onto different processors
  (i.e., no cost on blue arrow in the example)



Platform model



- $\mathrm{p}+2$ processors P_u, $0 \leq u \leq \mathrm{p}+1$
- $P_0 = P_{in}$: input data – $P_{p+1} = P_{out}$: output data
- P_1 to P_p: fully interconnected (clique)
- s_u: speed of processor P_u, $1 \leq u \leq p$, linear cost model
- bidirectional link $P_u \leftrightarrow P_v$, bandwidth $b_{u,v}$
- B_u^i / B_u^o: input/output network card capacity

Platform model: classification

Fully Homogeneous: Identical processors ($s_u = s$) and homogeneous communication devices ($b_{u,v} = b, \mathrm{B}^i_u = \mathrm{B}^i, \mathrm{B}^o_u = \mathrm{B}^o$): typical parallel machines

Communication Homogeneous: Homogeneous communication devices but different-speed processors ($s_u \neq s_v$): networks of workstations, clusters

Fully Heterogeneous: Fully heterogeneous architectures: hierarchical platforms, grids

Platform model: unreliable processors

- f_u: failure probability of processor P_u
  - independent of the duration of the application: global indicator of processor reliability
  - steady-state execution: loan/rent resources, cycle-stealing
  - fail-silent/fail-stop, no link failures (use different paths)
- *Failure Homogeneous*: Identically reliable processors ($f_u = f_v$), natural with *Fully Homogeneous*
- *Failure Heterogeneous*: Different failure probabilities ($f_u \neq f_v$), natural with *Communication Homogeneous* and *Fully Heterogeneous*

Platform model: unreliable processors

- f_u: failure probability of processor P_u
  - independent of the duration of the application: global indicator of processor reliability
  - steady-state execution: loan/rent resources, cycle-stealing
  - fail-silent/fail-stop, no link failures (use different paths)
- *Failure Homogeneous*: Identically reliable processors ($f_u = f_v$), natural with *Fully Homogeneous*
- *Failure Heterogeneous*: Different failure probabilities ($f_u \neq f_v$), natural with *Communication Homogeneous* and *Fully Heterogeneous*

Platform model: communications, a bit of history

Classical communication model in scheduling works:
macro-dataflow model

$$cost(T, T') = \begin{cases} 0 & \text{if } alloc(T) = alloc(T') \\ comm(T, T') & \text{otherwise} \end{cases}$$

- Task T communicates data to successor task T'
- $alloc(T)$: processor that executes T; $comm(T, T')$: defined by the application specification
- Two main assumptions:
  - (i) communication can occur as soon as data is available
  - (ii) no contention for network links
- (i) is reasonable, (ii) assumes infinite network resources!

Platform model: communications, a bit of history

Classical communication model in scheduling works:
macro-dataflow model

$$cost(T, T') = \begin{cases} 0 & \text{if } alloc(T) = alloc(T') \\ comm(T, T') & \text{otherwise} \end{cases}$$

- Task T communicates data to successor task T'
- $alloc(T)$: processor that executes T; $comm(T, T')$: defined by the application specification
- Two main assumptions:
  - (i) communication can occur as soon as data is available
  - (ii) no contention for network links
- (i) is reasonable, (ii) assumes infinite network resources!

Platform model: communications, a bit of history

Classical communication model in scheduling works:
macro-dataflow model

$$cost(T, T') = \begin{cases} 0 & \text{if } alloc(T) = alloc(T') \\ comm(T, T') & \text{otherwise} \end{cases}$$

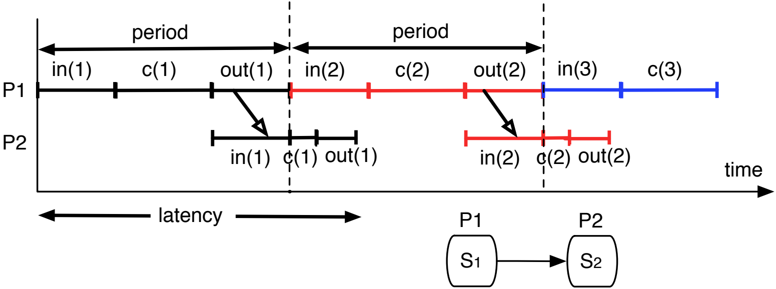
- Task T communicates data to successor task T'
- $alloc(T)$: processor that executes T; $comm(T, T')$: defined by the application specification
- Two main assumptions:
  - (i) communication can occur as soon as data is available
  - (ii) no contention for network links
- (i) is reasonable, (ii) assumes infinite network resources!

Platform model: one-port without overlap

- no overlap: at each time step, either computation or communication
- one-port: each processor can either send to or receive from a single other processor at any time step

Platform model: one-port without overlap

- no overlap: at each time step, either computation or communication
- one-port: each processor can either send to or receive from a single other processor at any time step

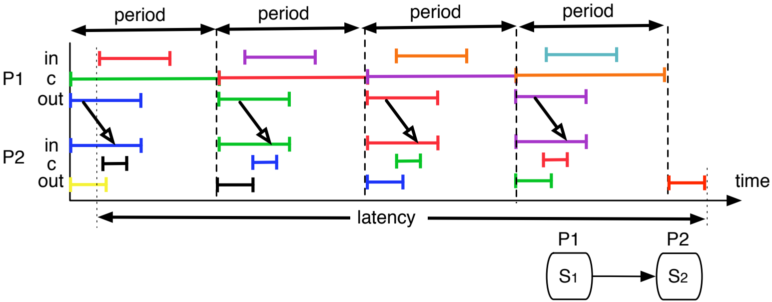


Platform model: bounded multi-port with overlap

- overlap: a processor can simultaneously compute and communicate
- bounded multi-port: simultaneous send and receive, but bound on the total outgoing/incoming communication (limitation of network card)

Platform model: bounded multi-port with overlap

- overlap: a processor can simultaneously compute and communicate
- bounded multi-port: simultaneous send and receive, but bound on the total outgoing/incoming communication (limitation of network card)



Platform model: communication models

- Multi-port: if several non-consecutive stages mapped onto the same processor, several concurrent communications
- Matches multi-threaded systems
- Fits well together with overlap
- One-port: radical option, where everything is serialized
- Natural to consider it without overlap
- Other communication models: more complicated such as protocols for path bandwidth allocation
- Intractable for algorithm design

Two considered models: good trade-off realism/tractability

Platform model: communication models

- Multi-port: if several non-consecutive stages mapped onto the same processor, several concurrent communications
- Matches multi-threaded systems
- Fits well together with overlap

- One-port: radical option, where everything is serialized
- Natural to consider it without overlap

- Other communication models: more complicated such as protocols for path bandwidth allocation
- Intractable for algorithm design

Two considered models: good trade-off realism/tractability

Platform model: communication models

- Multi-port: if several non-consecutive stages mapped onto the same processor, several concurrent communications
- Matches multi-threaded systems
- Fits well together with overlap

- One-port: radical option, where everything is serialized
- Natural to consider it without overlap

- Other communication models: more complicated such as protocols for path bandwidth allocation
- Intractable for algorithm design

Two considered models: good trade-off realism/tractability

Platform model: communication models

- Multi-port: if several non-consecutive stages mapped onto the same processor, several concurrent communications
- Matches multi-threaded systems
- Fits well together with overlap

- One-port: radical option, where everything is serialized
- Natural to consider it without overlap

- Other communication models: more complicated such as protocols for path bandwidth allocation
- Intractable for algorithm design

Two considered models: good trade-off realism/tractability

Outline

1. Models
   - Application model
   - Platform and communication models
2. Multi-criteria scheduling problems
   - Stage types and replication
   - Rule of the game
   - Optimization criteria
   - Define and classify problems
3. Complexity results
   - Mono-criterion problems
   - Bi-criteria problems
4. Conclusion

Mapping: stage types and replication

- Monolithic stages: must be mapped on one single processor since computation for a data set may depend on result of previous computation
  Interval $[\mathcal{S}_i..\mathcal{S}_j]$ on P_1:

$$\ldots\ \mathcal{S}_{i-1} \rightarrow \mathcal{S}_i..\mathcal{S}_j \text{ on } P_1\text{: data sets } \mathbf{1, 2, 3, \ldots} \rightarrow \mathcal{S}_{j+1}\ \ldots$$

Mapping: stage types and replication

- Monolithic stages: must be mapped on one single processor since computation for a data set may depend on result of previous computation
- Dealable stages: can be replicated on several processors, but not parallel, *i.e.*, a data set must be entirely processed on a single processor (distribute work)

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

$$\ldots\ \mathcal{S}_{i-1}\ -- \begin{array}{l} \mathcal{S}_i..\mathcal{S}_j \text{ on } P_1\text{: data sets } \mathbf{1, 4, 7, \ldots} \\ \mathcal{S}_i..\mathcal{S}_j \text{ on } P_2\text{: data sets } \mathbf{2, 5, 8, \ldots} \\ \mathcal{S}_i..\mathcal{S}_j \text{ on } P_3\text{: data sets } \mathbf{3, 6, 9, \ldots} \end{array} --\ \mathcal{S}_{j+1}\ \ldots$$

Mapping: stage types and replication

- Monolithic stages: must be mapped on one single processor since computation for a data set may depend on result of previous computation
- Dealable stages: can be replicated on several processors, but not parallel, *i.e.*, a data set must be entirely processed on a single processor (distribute work)
- Data-parallel stages: inherently parallel stages, one data set can be computed in parallel by several processors (partition work)

Data parallelize single stage $\mathcal{S}_i$ on $P_1, \ldots, P_q$

$\mathcal{S}_i$ $(w = 16)$ ●●●● ●●●● ●●●● ●●●● $\Rightarrow$

P_1 $(s_1 = 2)$: ●●●●●●●●

P_2 $(s_2 = 1)$: ●●●●

P_3 $(s_3 = 1)$: ●●●●

Mapping: stage types and replication

- Monolithic stages: must be mapped on one single processor
- Dealable stages: can be replicated on several processors
- Data-parallel stages: inherently parallel stages, one data set can be computed in parallel by several processors

Mapping: stage types and replication

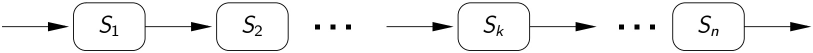
- Monolithic stages: must be mapped on one single processor
- Dealable stages: can be replicated on several processors
- Data-parallel stages: inherently parallel stages, one data set can be computed in parallel by several processors

- Replicating for reliability: one data set is processed several times on different processors (redundant work)

$$\ldots\ \mathcal{S}_{i-1} \quad -- \quad \begin{array}{l} \diagup \quad \mathcal{S}_i..\mathcal{S}_j \text{ on } P_1\text{: data sets } \mathbf{1, 2, 3, \ldots} \quad \diagdown \\ \quad\ \ \mathcal{S}_i..\mathcal{S}_j \text{ on } P_2\text{: data sets } \mathbf{1, 2, 3, \ldots} \\ \diagdown \quad \mathcal{S}_i..\mathcal{S}_j \text{ on } P_3\text{: data sets } \mathbf{1, 2, 3, \ldots} \quad \diagup \end{array} \quad -- \quad \mathcal{S}_{j+1}\ \ldots$$

Mapping strategies: rule of the game

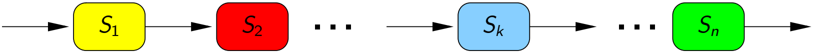
- Map each application stage onto one or more processors
- First simple scenario with no replication
- Allocation function $a : [1..n] \rightarrow [1..p]$
- $a(0) = 0$ $(= in)$ and $a(n+1) = p+1$ $(= out)$
- Several mapping strategies



The pipeline application

Mapping strategies: rule of the game

- Map each application stage onto one or more processors
- First simple scenario with no replication
- Allocation function $a : [1..n] \rightarrow [1..p]$
- $a(0) = 0$ $(= in)$ and $a(n+1) = p + 1$ $(= out)$
- Several mapping strategies



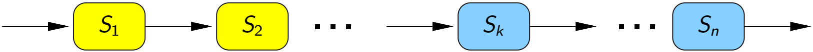
ONE-TO-ONE MAPPING: a is a one-to-one function, $n \leq p$

Mapping strategies: rule of the game

- Map each application stage onto one or more processors
- First simple scenario with no replication

- Allocation function $a : [1..n] \rightarrow [1..p]$
- $a(0) = 0$ $(= in)$ and $a(n+1) = p+1$ $(= out)$

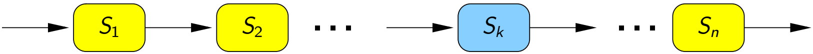
- Several mapping strategies



INTERVAL MAPPING: partition into $m \leq p$ intervals $I_j = [d_j, e_j]$

Mapping strategies: rule of the game

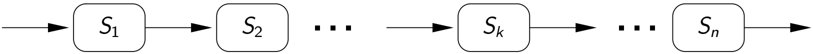
- Map each application stage onto one or more processors
- First simple scenario with no replication
- Allocation function $a : [1..n] \rightarrow [1..p]$
- $a(0) = 0$ $(= in)$ and $a(n+1) = p+1$ $(= out)$
- Several mapping strategies



GENERAL MAPPING: P_u is assigned any subset of stages

Mapping strategies: rule of the game

- Map each application stage onto one or more processors
- First simple scenario with no replication
- Allocation function $a : [1..n] \rightarrow [1..p]$
- $a(0) = 0$ $(= in)$ and $a(n+1) = p+1$ $(= out)$
- Several mapping strategies



- With replication: rules can be extended, $a(i)$ is a set of processor indices, difference between processors for reliability/performance

Mapping: objective function

Mono-criterion

- Minimize period $\mathcal{P}$ (inverse of throughput)
- Minimize latency $\mathcal{L}$ (time to process a data set)
- Minimize application failure probability $\mathcal{F}$

Mapping: objective function

Mono-criterion

- Minimize period $\mathcal{P}$ (inverse of throughput)
- Minimize latency $\mathcal{L}$ (time to process a data set)
- Minimize application failure probability $\mathcal{F}$

Multi-criteria

- How to define it?
  Minimize $\alpha.\mathcal{P} + \beta.\mathcal{L} + \gamma.\mathcal{F}$?
- Values which are not comparable

Mapping: objective function

Mono-criterion

- Minimize period $\mathcal{P}$ (inverse of throughput)
- Minimize latency $\mathcal{L}$ (time to process a data set)
- Minimize application failure probability $\mathcal{F}$

Multi-criteria

- How to define it?
  Minimize $\alpha.\mathcal{P} + \beta.\mathcal{L} + \gamma.\mathcal{F}$?
- Values which are not comparable

- Minimize $\mathcal{P}$ for a fixed latency and failure
- Minimize $\mathcal{L}$ for a fixed period and failure
- Minimize $\mathcal{F}$ for a fixed period and latency

Mapping: objective function

Mono-criterion

- Minimize period $\mathcal{P}$ (inverse of throughput)
- Minimize latency $\mathcal{L}$ (time to process a data set)
- Minimize application failure probability $\mathcal{F}$

Bi-criteria

- Period and Latency:
- Minimize $\mathcal{P}$ for a fixed latency
- Minimize $\mathcal{L}$ for a fixed period
- And so on...

Formal definition of period and latency

- Allocation function: characterizes a mapping
- Not enough information to compute the actual schedule of the application = *time step at which each operation takes place*
- Time steps at which comm and comp begin and end
- Cyclic schedules which repeat for each data set (period λ)

- No deal replication: S_i, $u \in a(i)$, $v \in a(i+1)$, data set k
  - $BeginComp^k_{i,u}/EndComp^k_{i,u}$ = time step at which comp of S_i on P_u for data set k begins/ends
  - $BeginComm^k_{i,u,v}/EndComm^k_{i,u,v}$ = time step at which comm between P_u and P_v for output of S_i for k begins/ends

$$\begin{cases} BeginComp^k_{i,u} = BeginComp^0_{i,u} + \lambda \times k \\ EndComp^k_{i,u} = EndComp^0_{i,u} + \lambda \times k \\ BeginComm^k_{i,u,v} = BeginComm^0_{i,u,v} + \lambda \times k \\ EndComm^k_{i,u,v} = EndComm^0_{i,u,v} + \lambda \times k \end{cases}$$

Formal definition of period and latency

- Allocation function: characterizes a mapping
- Not enough information to compute the actual schedule of the application = *time step at which each operation takes place*
- Time steps at which comm and comp begin and end
- Cyclic schedules which repeat for each data set (period λ)

- No deal replication: S_i, $u \in a(i)$, $v \in a(i+1)$, data set k
  - $BeginComp^k_{i,u} / EndComp^k_{i,u}$ = time step at which comp of S_i on P_u for data set k begins/ends
  - $BeginComm^k_{i,u,v} / EndComm^k_{i,u,v}$ = time step at which comm between P_u and P_v for output of S_i for k begins/ends

$$\begin{cases} BeginComp^k_{i,u} = BeginComp^0_{i,u} + \lambda \times k \\ EndComp^k_{i,u} = EndComp^0_{i,u} + \lambda \times k \\ BeginComm^k_{i,u,v} = BeginComm^0_{i,u,v} + \lambda \times k \\ EndComm^k_{i,u,v} = EndComm^0_{i,u,v} + \lambda \times k \end{cases}$$

Formal definition of period and latency

- Allocation function: characterizes a mapping
- Not enough information to compute the actual schedule of the application = *time step at which each operation takes place*
- Time steps at which comm and comp begin and end
- Cyclic schedules which repeat for each data set (period λ)

- No deal replication: S_i, $u \in a(i)$, $v \in a(i+1)$, data set k
  - $BeginComp^k_{i,u}/EndComp^k_{i,u}$ = time step at which comp of S_i on P_u for data set k begins/ends
  - $BeginComm^k_{i,u,v}/EndComm^k_{i,u,v}$ = time step at which comm between P_u and P_v for output of S_i for k begins/ends

$$\begin{cases} BeginComp^k_{i,u} = BeginComp^0_{i,u} + \lambda \times k \\ EndComp^k_{i,u} = EndComp^0_{i,u} + \lambda \times k \\ BeginComm^k_{i,u,v} = BeginComm^0_{i,u,v} + \lambda \times k \\ EndComm^k_{i,u,v} = EndComm^0_{i,u,v} + \lambda \times k \end{cases}$$

Formal definition of period and latency: *operation list*

- Given communication model: set of rules to have a valid operation list (OL)
- Non-preemptive models, synchronous communications
- Period $\mathcal{P} = \lambda$
- Latency $\mathcal{L} = \max\{EndComm^{0}_{n,u,out} \mid u \in a(n)\}$
- With deal replication: extension of the definition, periodic schedule rather than cyclic one
- Most cases: formula to express period and latency, *no need for OL*

Now, ready to describe optimization problems

Formal definition of period and latency: *operation list*

- Given communication model: set of rules to have a valid operation list (OL)
- Non-preemptive models, synchronous communications
- Period $\mathcal{P} = \lambda$
- Latency $\mathcal{L} = \max\{EndComm^0_{n,u,out} \mid u \in a(n)\}$
- With deal replication: extension of the definition, periodic schedule rather than cyclic one
- Most cases: formula to express period and latency, *no need for OL*

Now, ready to describe optimization problems

Formal definition of period and latency: *operation list*

- Given communication model: set of rules to have a valid operation list (OL)
- Non-preemptive models, synchronous communications
- Period $\mathcal{P} = \lambda$
- Latency $\mathcal{L} = \max\{EndComm^{0}_{n,u,out} \mid u \in a(n)\}$
- With deal replication: extension of the definition, periodic schedule rather than cyclic one
- Most cases: formula to express period and latency, *no need for OL*

Now, ready to describe optimization problems

Formal definition of period and latency: *operation list*

- Given communication model: set of rules to have a valid operation list (OL)
- Non-preemptive models, synchronous communications
- Period $\mathcal{P} = \lambda$
- Latency $\mathcal{L} = \max\{EndComm^{0}_{n,u,out} \mid u \in a(n)\}$
- With deal replication: extension of the definition, periodic schedule rather than cyclic one
- Most cases: formula to express period and latency, *no need for OL*

Now, ready to describe optimization problems

Formal definition of period and latency: *operation list*

- Given communication model: set of rules to have a valid operation list (OL)
- Non-preemptive models, synchronous communications
- Period $\mathcal{P} = \lambda$
- Latency $\mathcal{L} = \max\{EndComm^0_{n,u,out} \mid u \in a(n)\}$
- With deal replication: extension of the definition, periodic schedule rather than cyclic one
- Most cases: formula to express period and latency, *no need for OL*

Now, ready to describe optimization problems

One-to-one and interval mappings, no replication

- **Latency**: max time required by a data to traverse all stages

$$\mathcal{L} = \sum_{1 \leq j \leq m} \left\{ \frac{\delta_{d_j - 1}}{b_{a(d_j - 1), a(d_j)}} + \frac{\sum_{i=d_j}^{e_j} w_i}{s_{a(d_j)}} \right\} + \frac{\delta_n}{b_{a(d_m), out}}$$

- **Period**: definition depends on comm model (different rules in the OL), but always longest cycle-time of a processor: $\mathcal{P}^{(interval)} = \max_{1 \leq j \leq m} cycletime(P_{a(d_j)})$
- One-port model without overlap:

$$\mathcal{P} = \max_{1 \leq j \leq m} \left\{ \frac{\delta_{d_j - 1}}{b_{a(d_j - 1), a(d_j)}} + \frac{\sum_{i=d_j}^{e_j} w_i}{s_{a(d_j)}} + \frac{\delta_{e_j}}{b_{a(d_j), a(e_j + 1)}} \right\}$$

- Bounded multi-port model with overlap:

One-to-one and interval mappings, no replication

- **Latency**: max time required by a data to traverse all stages

$$\mathcal{L} = \sum_{1 \leq j \leq m} \left\{ \frac{\delta_{d_j - 1}}{b_{a(d_j - 1), a(d_j)}} + \frac{\sum_{i=d_j}^{e_j} w_i}{s_{a(d_j)}} \right\} + \frac{\delta_n}{b_{a(d_m), out}}$$

- **Period**: definition depends on comm model (different rules in the OL), but always longest cycle-time of a processor:
  $\mathcal{P}^{(interval)} = \max_{1 \leq j \leq m} cycletime(P_{a(d_j)})$

- One-port model without overlap:

$$\mathcal{P} = \max_{1 \leq j \leq m} \left\{ \frac{\delta_{d_j - 1}}{b_{a(d_j - 1), a(d_j)}} + \frac{\sum_{i=d_j}^{e_j} w_i}{s_{a(d_j)}} + \frac{\delta_{e_j}}{b_{a(d_j), a(e_j + 1)}} \right\}$$

- Bounded multi-port model with overlap:

One-to-one and interval mappings, no replication

- **Latency**: max time required by a data to traverse all stages

$$\mathcal{L} = \sum_{1 \leq j \leq m} \left\{ \frac{\delta_{d_j - 1}}{b_{a(d_j-1),a(d_j)}} + \frac{\sum_{i=d_j}^{e_j} w_i}{s_{a(d_j)}} \right\} + \frac{\delta_n}{b_{a(d_m),out}}$$

- **Period**: definition depends on comm model (different rules in the OL), but always longest cycle-time of a processor:
  $\mathcal{P}^{(interval)} = \max_{1 \leq j \leq m} cycletime(P_{a(d_j)})$
- One-port model without overlap:

$$\mathcal{P} = \max_{1 \leq j \leq m} \left\{ \frac{\delta_{d_j - 1}}{b_{a(d_j-1),a(d_j)}} + \frac{\sum_{i=d_j}^{e_j} w_i}{s_{a(d_j)}} + \frac{\delta_{e_j}}{b_{a(d_j),a(e_j+1)}} \right\}$$

- Bounded multi-port model with overlap:

One-to-one and interval mappings, no replication

- **Latency**: max time required by a data to traverse all stages

$$\mathcal{L} = \sum_{1 \leq j \leq m} \left\{ \frac{\delta_{d_j - 1}}{b_{a(d_j-1),a(d_j)}} + \frac{\sum_{i=d_j}^{e_j} w_i}{s_{a(d_j)}} \right\} + \frac{\delta_n}{b_{a(d_m),out}}$$

- **Period**: definition depends on comm model (different rules in the OL), but always longest cycle-time of a processor:
  $\mathcal{P}^{(interval)} = \max_{1 \leq j \leq m} cycletime(P_{a(d_j)})$
- One-port model without overlap:

$$\mathcal{P} = \max_{1 \leq j \leq m} \left\{ \frac{\delta_{d_j - 1}}{b_{a(d_j-1),a(d_j)}} + \frac{\sum_{i=d_j}^{e_j} w_i}{s_{a(d_j)}} + \frac{\delta_{e_j}}{b_{a(d_j),a(e_j+1)}} \right\}$$

- Bounded multi-port model with overlap:

$$\mathcal{P} = \max_{1 \leq j \leq m} \left\{ max \left(\frac{\delta_{d_j - 1}}{\min\left(b_{a(d_j-1),a(d_j)}, \mathrm{B}^i_{a(d_j)}\right)}, \frac{\sum_{i=d_j}^{e_j} w_i}{s_{a(d_j)}}, \frac{\delta_{e_j}}{\min\left(b_{a(d_j),a(e_j+1)}, \mathrm{B}^o_{a(d_j)}\right)} \right) \right\}$$

Adding replication for reliability

- Each processor: failure probability $0 \leq f_u \leq 1$
- m intervals, set of processors $a(d_j)$ for interval j

$$\ldots \mathcal{S}_{d_j-1} \begin{array}{c} / \\ -- \\ \backslash \end{array} \begin{array}{l} I_j \text{ on } P_1\text{: data sets } \mathbf{1, 2, 3, \ldots} \\ I_j \text{ on } P_2\text{: data sets } \mathbf{1, 2, 3, \ldots} \\ I_j \text{ on } P_3\text{: data sets } \mathbf{1, 2, 3, \ldots} \end{array} \begin{array}{c} \backslash \\ -- \\ / \end{array} \mathcal{S}_{e_j+1} \ldots$$

Adding replication for reliability

- Each processor: failure probability $0 \leq f_u \leq 1$
- m intervals, set of processors $a(d_j)$ for interval j

$$\ldots \mathcal{S}_{d_j-1} \begin{array}{c} \diagup \\ -- \\ \diagdown \end{array} \begin{array}{l} I_j \text{ on } P_1\text{: data sets } \mathbf{1, 2, 3, \ldots} \\ I_j \text{ on } P_2\text{: data sets } \mathbf{1, 2, 3, \ldots} \\ I_j \text{ on } P_3\text{: data sets } \mathbf{1, 2, 3, \ldots} \end{array} \begin{array}{c} \diagdown \\ -- \\ \diagup \end{array} \mathcal{S}_{e_j+1} \ldots$$

$$\mathcal{F} = 1 - \prod_{1 \leq j \leq m} (1 - \prod_{u \in a(d_j)} f_u)$$

Adding replication for reliability

- Each processor: failure probability $0 \leq f_u \leq 1$
- m intervals, set of processors $a(d_j)$ for interval j

$$\ldots \mathcal{S}_{d_j-1} \begin{array}{c} \diagup \\ -- \\ \diagdown \end{array} \begin{array}{l} I_j \text{ on } P_1\text{: data sets } \mathbf{1, 2, 3, \ldots} \\ I_j \text{ on } P_2\text{: data sets } \mathbf{1, 2, 3, \ldots} \\ I_j \text{ on } P_3\text{: data sets } \mathbf{1, 2, 3, \ldots} \end{array} \begin{array}{c} \diagdown \\ -- \\ \diagup \end{array} \mathcal{S}_{e_j+1} \ldots$$

$$\mathcal{F} = 1 - \prod_{1 \leq j \leq m} \left(1 - \prod_{u \in a(d_j)} f_u\right)$$

- Consensus protocol: one surviving processor performs all outgoing communications
- Worst case scenario: new formulas for **latency** and **period** to account for redundant communications

Adding replication for period and latency

- Dealable stages: replication of stage or interval of stages.
  - No latency decrease; period may decrease (fewer data sets per processor)
  - **Latency**: longest path in DAG, no conflicts between data sets
  - **Period**: no communication: $trav_i/k$ if S_i onto k processors; with communications: cases with no critical resources, *need OL to define period*

$\mathcal{S}_i..\mathcal{S}_j$ on P_1: data sets **1, 4, 7, . . .**

$\ldots\ \mathcal{S}_{i-1}$ −− $\mathcal{S}_i..\mathcal{S}_j$ on P_2: data sets **2, 5, 8, . . .** −− $\mathcal{S}_{j+1}\ \ldots$

$\mathcal{S}_i..\mathcal{S}_j$ on P_3: data sets **3, 6, 9, . . .**

Adding replication for period and latency

- Dealable stages: replication of stage or interval of stages.
  - No latency decrease; period may decrease (fewer data sets per processor)
  - **Latency**: longest path in DAG, no conflicts between data sets
  - **Period**: no communication: $trav_i/k$ if S_i onto k processors; with communications: cases with no critical resources, *need OL to define period*

- Data-parallel stages: replication of single stage
  - Both latency and period may decrease
  - Becomes very difficult with communications

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

• • • •
• • • •
• • • •
• • • •

$\Rightarrow$

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

Adding replication for period and latency

- Dealable stages: replication of stage or interval of stages.
  - No latency decrease; period may decrease (fewer data sets per processor)
  - **Latency**: longest path in DAG, no conflicts between data sets
  - **Period**: no communication: $trav_i/k$ if S_i onto k processors; with communications: cases with no critical resources, *need OL to define period*
- Data-parallel stages: replication of single stage
  - Both latency and period may decrease
  - Becomes very difficult with communications

⇒ Model with no communication!

Adding replication for period and latency

- Dealable stages: replication of stage or interval of stages.
  - No latency decrease; period may decrease (fewer data sets per processor)
  - **Latency**: longest path in DAG, no conflicts between data sets
  - **Period**: no communication: $trav_i/k$ if S_i onto k processors; with communications: cases with no critical resources, *need OL to define period*

- Data-parallel stages: replication of single stage
  - Both latency and period may decrease
  - Becomes very difficult with communications

⇒ Model with no communication!

Replication for performance + replication for reliability: possible to mix both approaches, difficulties of both models

Moving to general mappings

- Failure probability: definition in the general case easy to derive (all kinds of replication)
- Latency: can be defined with a formula for *Communication Homogeneous* platforms with no data-parallelism
  - *Fully Heterogeneous*: longest path in DAG (poly. time)
  - With data-parallel stages: can be computed (without OL) only with no communication
- Period: case with no replication for period and latency
  - Bounded multi-port model with overlap: period = maximum cycle-time of processors; communications in parallel: input comms on data sets $k_1 + 1, \ldots, k_\ell + 1$; computes on $k_1, \ldots, k_\ell$, outputs $k_1 - 1, \ldots, k_\ell - 1$ $\rightarrow$ no conflicts;
  - Without overlap: conflicts similar to case with replication; NP-hard to decide how to order communications

Moving to general mappings

- Failure probability: definition in the general case easy to derive (all kinds of replication)
- **Latency**: can be defined with a formula for *Communication Homogeneous* platforms with no data-parallelism
  - *Fully Heterogeneous*: longest path in DAG (poly. time)
  - With data-parallel stages: can be computed (without OL) only with no communication
- **Period**: case with no replication for period and latency
  - Bounded multi-port model with overlap: period = maximum cycle-time of processors; communications in parallel: input comms on data sets $k_1 + 1, \ldots, k_\ell + 1$; computes on $k_1, \ldots, k_\ell$, outputs $k_1 - 1, \ldots, k_\ell - 1$ $\rightarrow$ no conflicts;
  - Without overlap: conflicts similar to case with replication; NP-hard to decide how to order communications

Moving to general mappings

- Failure probability: definition in the general case easy to derive (all kinds of replication)
- **Latency**: can be defined with a formula for *Communication Homogeneous* platforms with no data-parallelism
  - *Fully Heterogeneous*: longest path in DAG (poly. time)
  - With data-parallel stages: can be computed (without OL) only with no communication
- **Period**: case with no replication for period and latency
  - Bounded multi-port model with overlap: period = maximum cycle-time of processors; communications in parallel: input comms on data sets $k_1 + 1, \ldots, k_\ell + 1$; computes on $k_1, \ldots, k_\ell$, outputs $k_1 - 1, \ldots, k_\ell - 1$ $\rightarrow$ no conflicts;
  - Without overlap: conflicts similar to case with replication; NP-hard to decide how to order communications

Moving to general mappings

- Failure probability: definition in the general case easy to derive (all kinds of replication)
- **Latency**: can be defined with a formula for *Communication Homogeneous* platforms with no data-parallelism
  - *Fully Heterogeneous*: longest path in DAG (poly. time)
  - With data-parallel stages: can be computed (without OL) only with no communication
- **Period**: case with no replication for period and latency
  - Bounded multi-port model with overlap: period = maximum cycle-time of processors; communications in parallel: input comms on data sets $k_1 + 1, \ldots, k_\ell + 1$; computes on $k_1, \ldots, k_\ell$, outputs $k_1 - 1, \ldots, k_\ell - 1$ $\rightarrow$ no conflicts;
  - Without overlap: conflicts similar to case with replication; NP-hard to decide how to order communications

Outline

1. Models
   - Application model
   - Platform and communication models
2. Multi-criteria scheduling problems
   - Stage types and replication
   - Rule of the game
   - Optimization criteria
   - Define and classify problems
3. Complexity results
   - Mono-criterion problems
   - Bi-criteria problems
4. Conclusion

Failure probability

- Turns out simple for interval and general mappings: minimum reached by replicating (for reliability) the whole pipeline as a single interval on all processors: $\mathcal{F} = \prod_{u=1}^{p} \mathrm{f}_u$
- One-to-one mappings: polynomial for *Failure Homogeneous* platforms (balance number of processors to stages), NP-hard for *Failure Heterogeneous* platforms (3-PARTITION with n stages and $3n$ processors)

$\mathcal{F}$	**Failure-Hom.**	**Failure-Het.**
One-to-one	polynomial	NP-hard
Interval	polynomial	
General	polynomial	

Failure probability

- Turns out simple for interval and general mappings: minimum reached by replicating (for reliability) the whole pipeline as a single interval on all processors: $\mathcal{F} = \prod_{u=1}^{p} \mathsf{f}_u$
- One-to-one mappings: polynomial for *Failure Homogeneous* platforms (balance number of processors to stages), NP-hard for *Failure Heterogeneous* platforms (3-PARTITION with n stages and $3n$ processors)

$\mathcal{F}$	**Failure-Hom.**	**Failure-Het.**
One-to-one	polynomial	NP-hard
Interval	polynomial	
General	polynomial	

Failure probability

- Turns out simple for interval and general mappings: minimum reached by replicating (for reliability) the whole pipeline as a single interval on all processors: $\mathcal{F} = \prod_{u=1}^{p} \mathsf{f}_u$
- One-to-one mappings: polynomial for *Failure Homogeneous* platforms (balance number of processors to stages), NP-hard for *Failure Heterogeneous* platforms (3-PARTITION with n stages and $3n$ processors)

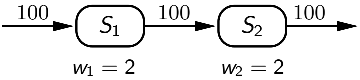
$\mathcal{F}$	**Failure-Hom.**	**Failure-Het.**
One-to-one	polynomial	NP-hard
Interval	polynomial	
General	polynomial	

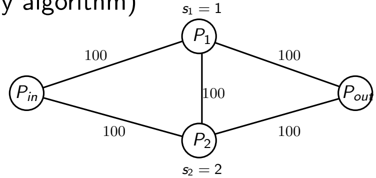
Latency

- Replication of dealable stages, replication for reliability: no impact on latency
- No data-parallelism: reduce communication costs
  - *Fully Homogeneous* and *Communication Homogeneous* platforms: map all stages onto fastest processor (1 interval); one-to-one mappings: most computationally expensive stages onto fastest processors (greedy algorithm)
  - *Fully Heterogeneous* platforms: problem of input/output communications: may need to split interval

Latency

- Replication of dealable stages, replication for reliability: no impact on latency
- No data-parallelism: reduce communication costs
  - *Fully Homogeneous* and *Communication Homogeneous* platforms: map all stages onto fastest processor (1 interval); one-to-one mappings: most computationally expensive stages onto fastest processors (greedy algorithm)

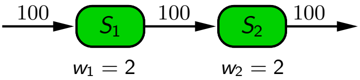


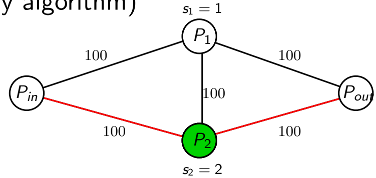


  - *Fully Heterogeneous* platforms: problem of input/output communications: may need to split interval

Latency

- Replication of dealable stages, replication for reliability: no impact on latency
- No data-parallelism: reduce communication costs
  - *Fully Homogeneous* and *Communication Homogeneous* platforms: map all stages onto fastest processor (1 interval); one-to-one mappings: most computationally expensive stages onto fastest processors (greedy algorithm)

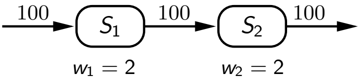


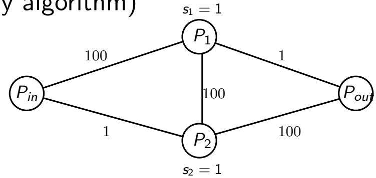


  - *Fully Heterogeneous* platforms: problem of input/output communications: may need to split interval

Latency

- Replication of dealable stages, replication for reliability: no impact on latency
- No data-parallelism: reduce communication costs
  - *Fully Homogeneous* and *Communication Homogeneous* platforms: map all stages onto fastest processor (1 interval); one-to-one mappings: most computationally expensive stages onto fastest processors (greedy algorithm)

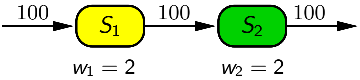


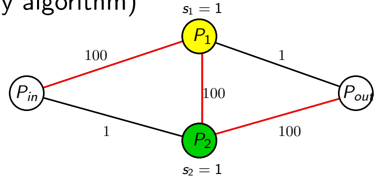


  - *Fully Heterogeneous* platforms: problem of input/output communications: may need to split interval

Latency

- Replication of dealable stages, replication for reliability: no impact on latency
- No data-parallelism: reduce communication costs
  - *Fully Homogeneous* and *Communication Homogeneous* platforms: map all stages onto fastest processor (1 interval); one-to-one mappings: most computationally expensive stages onto fastest processors (greedy algorithm)





  - *Fully Heterogeneous* platforms: problem of input/output communications: may need to split interval

Latency on *Fully Heterogeneous* platforms

- *Fully Heterogeneous* platforms: NP-hard for **one-to-one** and **interval** mappings (involved reductions), polynomial for **general** mappings (shortest paths)

Latency on *Fully Heterogeneous* platforms

- *Fully Heterogeneous* platforms: NP-hard for **one-to-one** and **interval** mappings (involved reductions), polynomial for **general** mappings (shortest paths)
- Idea of the reduction for **interval mappings**: from DISJOINT-CONNECTING-PATH (DCP)

Latency on *Fully Heterogeneous* platforms

Idea of the reduction for **interval mappings**: from DISJOINT-CONNECTING-PATH (DCP) $\rightarrow$ Instance of DCP: graph $G = (V, E)$, $k+1$ disjoint vertex pairs (x_i, y_i). Does G contain $k+1$ mutually vertex-disjoint paths connecting x_i to y_i?

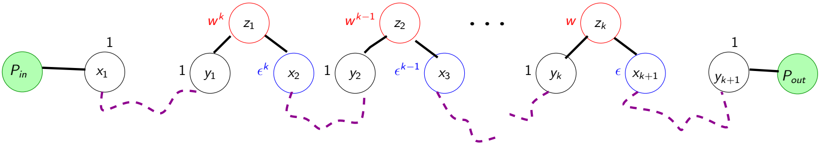
Latency on *Fully Heterogeneous* platforms

Idea of the reduction for **interval mappings**: from DISJOINT-CONNECTING-PATH (DCP) $\rightarrow$ Instance of DCP: graph $G = (V, E)$, $k+1$ disjoint vertex pairs (x_i, y_i). Does G contain $k+1$ mutually vertex-disjoint paths connecting x_i to y_i?

- The pipeline application:

$$\underbrace{w^k ... w^k}_{n-2} \quad w^{2k} \quad \epsilon^k \quad \underbrace{w^{k-1} ... w^{k-1}}_{n-2} \quad w^{2k-1} \quad \epsilon^{k-1} \quad \dots \quad \underbrace{w ... w}_{n-2} \quad w^{k+1} \quad \epsilon \quad \underbrace{1 ... 1}_{n}$$

- The execution platform:



- Latency $\mathcal{L} = 2n^2 w^k$?

Latency with data-parallelism

- With data-parallelism: model with no communication; polynomial with identical processor speeds (dynamic programming algorithm), NP-hard otherwise (2-PARTITION)
- Problem becomes NP-hard for *Communication Homogeneous platforms* because of data-parallelism

$\mathcal{L}$	Fully Hom.	Comm. Hom.	Hetero.
no DP, One-to-one	polynomial		NP-hard
no DP, Interval	polynomial		NP-hard
no DP, General	polynomial		
with DP, no coms	polynomial	NP-hard	

Latency with data-parallelism

- With data-parallelism: model with no communication; polynomial with identical processor speeds (dynamic programming algorithm), NP-hard otherwise (2-PARTITION)
- Problem becomes NP-hard for *Communication Homogeneous* platforms because of data-parallelism

$\mathcal{L}$	Fully Hom.	Comm. Hom.	Hetero.
no DP, One-to-one	polynomial		NP-hard
no DP, Interval	polynomial		NP-hard
no DP, General	polynomial		
with DP, no coms	polynomial	NP-hard	

Latency with data-parallelism

- With data-parallelism: model with no communication; polynomial with identical processor speeds (dynamic programming algorithm), NP-hard otherwise (2-PARTITION)
- Problem becomes NP-hard for *Communication Homogeneous* platforms because of data-parallelism

$\mathcal{L}$	**Fully Hom.**	**Comm. Hom.**	**Hetero.**
no DP, One-to-one	polynomial		NP-hard
no DP, Interval	polynomial		NP-hard
no DP, General	polynomial		
with DP, no coms	polynomial	NP-hard	

Period - Example with no comm, no replication

$$\begin{array}{ccccccc} \mathcal{S}_1 & \rightarrow & \mathcal{S}_2 & \rightarrow & \mathcal{S}_3 & \rightarrow & \mathcal{S}_4 \\ 2 & & 1 & & 3 & & 4 \end{array}$$

2 processors (P_1 and P_2) of speed 1

Optimal period?

Period - Example with no comm, no replication

$\mathcal{S}_1$	$\rightarrow$	$\mathcal{S}_2$	$\rightarrow$	$\mathcal{S}_3$	$\rightarrow$	$\mathcal{S}_4$
2		1		3		4

2 processors (P_1 and P_2) of speed 1

Optimal period?

$\mathcal{P} = 5, \quad \mathcal{S}_1\mathcal{S}_3 \rightarrow P_1, \mathcal{S}_2\mathcal{S}_4 \rightarrow P_2$

Perfect load-balancing in this case, but NP-hard (2-PARTITION)

Interval mapping?

Period - Example with no comm, no replication

$$\begin{array}{ccccccc} \mathcal{S}_1 & \rightarrow & \mathcal{S}_2 & \rightarrow & \mathcal{S}_3 & \rightarrow & \mathcal{S}_4 \\ 2 & & 1 & & 3 & & 4 \end{array}$$

2 processors (P_1 and P_2) of speed 1

Optimal period?

$\mathcal{P} = 5, \quad \mathcal{S}_1\mathcal{S}_3 \rightarrow P_1, \mathcal{S}_2\mathcal{S}_4 \rightarrow P_2$

Perfect load-balancing in this case, but NP-hard (2-PARTITION)

Interval mapping?

$\mathcal{P} = 6, \quad \mathcal{S}_1\mathcal{S}_2\mathcal{S}_3 \rightarrow P_1, \mathcal{S}_4 \rightarrow P_2$

Polynomial algorithm?

Period - Example with no comm, no replication

$$\begin{matrix} \mathcal{S}_1 & \rightarrow & \mathcal{S}_2 & \rightarrow & \mathcal{S}_3 & \rightarrow & \mathcal{S}_4 \\ 2 & & 1 & & 3 & & 4 \end{matrix}$$

2 processors (P_1 and P_2) of speed 1

Optimal period?

$\mathcal{P} = 5, \quad \mathcal{S}_1\mathcal{S}_3 \rightarrow P_1, \mathcal{S}_2\mathcal{S}_4 \rightarrow P_2$

Perfect load-balancing in this case, but NP-hard (2-PARTITION)

Interval mapping?

$\mathcal{P} = 6, \quad \mathcal{S}_1\mathcal{S}_2\mathcal{S}_3 \rightarrow P_1, \mathcal{S}_4 \rightarrow P_2$

Polynomial algorithm?

Chains-on-chains partitioning problem, use dynamic programming

Period - Example with no comm, no replication

$$\begin{array}{ccccccc} \mathcal{S}_1 & \rightarrow & \mathcal{S}_2 & \rightarrow & \mathcal{S}_3 & \rightarrow & \mathcal{S}_4 \\ 2 & & 1 & & 3 & & 4 \end{array}$$

P_1 of speed 2, and P_2 of speed 3

Optimal period?

$\mathcal{P} = 5, \quad \mathcal{S}_1\mathcal{S}_3 \rightarrow P_1, \mathcal{S}_2\mathcal{S}_4 \rightarrow P_2$

Perfect load-balancing in this case, but NP-hard (2-PARTITION)

Interval mapping?

$\mathcal{P} = 6, \quad \mathcal{S}_1\mathcal{S}_2\mathcal{S}_3 \rightarrow P_1, \mathcal{S}_4 \rightarrow P_2$

Polynomial algorithm?

Chains-on-chains partitioning problem, use dynamic programming

Heterogeneous platform?

Period - Example with no comm, no replication

$$\begin{array}{ccccccc} \mathcal{S}_1 & \rightarrow & \mathcal{S}_2 & \rightarrow & \mathcal{S}_3 & \rightarrow & \mathcal{S}_4 \\ 2 & & 1 & & 3 & & 4 \end{array}$$

P_1 of speed 2, and P_2 of speed 3

Optimal period?

$\mathcal{P} = 5, \quad \mathcal{S}_1\mathcal{S}_3 \rightarrow P_1, \mathcal{S}_2\mathcal{S}_4 \rightarrow P_2$

Perfect load-balancing in this case, but NP-hard (2-PARTITION)

Interval mapping?

$\mathcal{P} = 6, \quad \mathcal{S}_1\mathcal{S}_2\mathcal{S}_3 \rightarrow P_1, \mathcal{S}_4 \rightarrow P_2$

Polynomial algorithm?

Chains-on-chains partitioning problem, use dynamic programming

Heterogeneous platform?

$\mathcal{P} = 2, \quad \mathcal{S}_1\mathcal{S}_2\mathcal{S}_3 \rightarrow P_2, \mathcal{S}_4 \rightarrow P_1$

Heterogeneous chains-on-chains, NP-hard

Period - Complexity

$\mathcal{P}$	**Fully Hom.**	**Comm. Hom.**	**Hetero.**
One-to-one	polynomial	polynomial	NP-hard
Interval	polynomial	NP-hard	NP-hard
General	NP-hard	NP-hard	

- With replication?
  - No change in complexity except one-to-one/comm-hom (the problem becomes NP-hard, reduction from 2-PARTITION, enforcing use of data-parallelism) and general/fully-hom (the problem becomes polynomial)
  - Other NP-completeness proofs remain valid
  - Fully homogeneous platforms: one interval replicated onto all processors (works also for general mappings); greedy assignment for one-to-one mappings

Period - Complexity

$\mathcal{P}$	Fully Hom.	Comm. Hom.	Hetero.
One-to-one	polynomial	polynomial	NP-hard
Interval	polynomial	NP-hard	NP-hard
General	NP-hard	NP-hard	

- With replication?
  - No change in complexity except one-to-one/comm-hom (the problem becomes NP-hard, reduction from 2-PARTITION, enforcing use of data-parallelism) and general/fully-hom (the problem becomes polynomial)
  - Other NP-completeness proofs remain valid
  - Fully homogeneous platforms: one interval replicated onto all processors (works also for general mappings); greedy assignment for one-to-one mappings

Period - Complexity

$\mathcal{P}$	Fully Hom.	Comm. Hom.	Hetero.
One-to-one	polynomial	polynomial, NP-hard (rep)	NP-hard
Interval	polynomial	NP-hard	NP-hard
General	NP-hard, poly (rep)	NP-hard	

- With replication?
  - No change in complexity except one-to-one/comm-hom (the problem becomes NP-hard, reduction from 2-PARTITION, enforcing use of data-parallelism) and general/fully-hom (the problem becomes polynomial)
  - Other NP-completeness proofs remain valid
  - Fully homogeneous platforms: one interval replicated onto all processors (works also for general mappings); greedy assignment for one-to-one mappings

Bi-criteria period/latency

- Most problems NP-hard because of **period**
- Dynamic programming algorithm for fully homogeneous platforms
- Integer linear program for interval mappings, fully heterogeneous platforms, bi-criteria, without overlap
- Variables:
  - *Obj*: period or latency of the pipeline, depending on the objective function
  - $x_{i,u}$: 1 if S_i on P_u (0 otherwise)
  - $z_{i,u,v}$: 1 if S_i on P_u and S_{i+1} on P_v (0 otherwise)
  - first_u and last_u: integer denoting first and last stage assigned to P_u (to enforce interval constraints)

Bi-criteria period/latency

- Most problems NP-hard because of **period**
- Dynamic programming algorithm for fully homogeneous platforms
- Integer linear program for interval mappings, fully heterogeneous platforms, bi-criteria, without overlap
- Variables:
  - *Obj*: period or latency of the pipeline, depending on the objective function
  - $x_{i,u}$: 1 if S_i on P_u (0 otherwise)
  - $z_{i,u,v}$: 1 if S_i on P_u and S_{i+1} on P_v (0 otherwise)
  - first_u and last_u: integer denoting first and last stage assigned to P_u (to enforce interval constraints)

Bi-criteria period/latency

- Most problems NP-hard because of **period**
- Dynamic programming algorithm for fully homogeneous platforms
- Integer linear program for interval mappings, fully heterogeneous platforms, bi-criteria, without overlap
- Variables:
  - *Obj*: period or latency of the pipeline, depending on the objective function
  - $x_{i,u}$: 1 if S_i on P_u (0 otherwise)
  - $z_{i,u,v}$: 1 if $\mathcal{S}_i$ on P_u and S_{i+1} on P_v (0 otherwise)
  - first_u and last_u: integer denoting first and last stage assigned to P_u (to enforce interval constraints)

Linear program: constraints

Constraints on processors and links:

- $\forall i \in [0..\text{n}+1], \quad \sum_u x_{i,u} = 1$
- $\forall i \in [0..\text{n}], \quad \sum_{u,v} z_{i,u,v} = 1$
- $\forall i \in [0..\text{n}], \forall u,v \in [0..\text{p}+1], x_{i,u} + x_{i+1,v} \leq 1 + z_{i,u,v}$

Constraints on intervals:

- $\forall i \in [1..\text{n}], \forall u \in [1..\text{p}], \quad \text{first}_u \leq i.x_{i,u} + \text{n}.(1 - x_{i,u})$
- $\forall i \in [1..\text{n}], \forall u \in [1..\text{p}], \quad \text{last}_u \geq i.x_{i,u}$
- $\forall i \in [1..\text{n}-1], \forall u,v \in [1..\text{p}], u \neq v,$
  $\text{last}_u \leq i.z_{i,u,v} + \text{n}.(1 - z_{i,u,v})$
- $\forall i \in [1..\text{n}-1], \forall u,v \in [1..\text{p}], u \neq v, \quad \text{first}_v \geq (i+1).z_{i,u,v}$

Linear program: constraints

Constraints on processors and links:

- $\forall i \in [0..\mathrm{n}+1], \quad \sum_u x_{i,u} = 1$
- $\forall i \in [0..\mathrm{n}], \quad \sum_{u,v} z_{i,u,v} = 1$
- $\forall i \in [0..\mathrm{n}], \forall u, v \in [0..\mathrm{p}+1], x_{i,u} + x_{i+1,v} \leq 1 + z_{i,u,v}$

Constraints on intervals:

- $\forall i \in [1..\mathrm{n}], \forall u \in [1..\mathrm{p}], \quad \mathrm{first}_u \leq i.x_{i,u} + \mathrm{n}.(1 - x_{i,u})$
- $\forall i \in [1..\mathrm{n}], \forall u \in [1..\mathrm{p}], \quad \mathrm{last}_u \geq i.x_{i,u}$
- $\forall i \in [1..\mathrm{n}-1], \forall u, v \in [1..\mathrm{p}], u \neq v,$
  $\mathrm{last}_u \leq i.z_{i,u,v} + \mathrm{n}.(1 - z_{i,u,v})$
- $\forall i \in [1..\mathrm{n}-1], \forall u, v \in [1..\mathrm{p}], u \neq v, \quad \mathrm{first}_v \geq (i+1).z_{i,u,v}$

Linear program: constraints

$$\forall u \in [1..p], \sum_{i=1}^{n} \left\{ \left(\sum_{t \neq u} \frac{\delta_{i-1}}{b} z_{i-1,t,u} \right) + \frac{w_i}{s_u} x_{i,u} + \left(\sum_{v \neq u} \frac{\delta_i}{b} z_{i,u,v} \right) \right\} \leq \mathcal{P}$$

$$\sum_{u=1}^{p} \sum_{i=1}^{n} \left[\left(\sum_{t \neq u, t \in [0..p+1]} \frac{\delta_{i-1}}{b} z_{i-1,t,u} \right) + \frac{w_i}{s_u} x_{i,u} \right] + \left(\sum_{u \in [0..p]} \frac{\delta_n}{b} z_{n,u,out} \right) \leq \mathcal{L}$$

Min period with fixed latency

$Obj = \mathcal{P}$

$\mathcal{L}$ is fixed

Min latency with fixed period

$Obj = \mathcal{L}$

$\mathcal{P}$ is fixed

Linear program: constraints

$$\forall u \in [1..p], \sum_{i=1}^{n} \left\{ \left(\sum_{t \neq u} \frac{\delta_{i-1}}{b} z_{i-1,t,u} \right) + \frac{w_i}{s_u} x_{i,u} + \left(\sum_{v \neq u} \frac{\delta_i}{b} z_{i,u,v} \right) \right\} \leq \mathcal{P}$$

$$\sum_{u=1}^{p} \sum_{i=1}^{n} \left[\left(\sum_{t \neq u, t \in [0..p+1]} \frac{\delta_{i-1}}{b} z_{i-1,t,u} \right) + \frac{w_i}{s_u} x_{i,u} \right] + \left(\sum_{u \in [0..p]} \frac{\delta_n}{b} z_{n,u,out} \right) \leq \mathcal{L}$$

Min period with fixed latency

$$Obj = \mathcal{P}$$

$\mathcal{L}$ is fixed

Min latency with fixed period

$$Obj = \mathcal{L}$$

$\mathcal{P}$ is fixed

Other multi-criteria problems

- Latency/reliability: two "easy" instances, polynomial bi-criteria algorithms, single interval often optimal
- Reliability/period: mixes difficulties, period often NP-hard and reliability strongly non-linear
- **Tri-criteria**: even more difficult
- **Experimental approach**, design of polynomial heuristics for such difficult problem instances

Other multi-criteria problems

- Latency/reliability: two "easy" instances, polynomial bi-criteria algorithms, single interval often optimal
- Reliability/period: mixes difficulties, period often NP-hard and reliability strongly non-linear
- **Tri-criteria**: even more difficult
- **Experimental approach**, design of polynomial heuristics for such difficult problem instances

Other multi-criteria problems

- Latency/reliability: two "easy" instances, polynomial bi-criteria algorithms, single interval often optimal
- Reliability/period: mixes difficulties, period often NP-hard and reliability strongly non-linear
- **Tri-criteria**: even more difficult
- **Experimental approach**, design of polynomial heuristics for such difficult problem instances

Other multi-criteria problems

- Latency/reliability: two "easy" instances, polynomial bi-criteria algorithms, single interval often optimal
- Reliability/period: mixes difficulties, period often NP-hard and reliability strongly non-linear
- **Tri-criteria**: even more difficult
- **Experimental approach**, design of polynomial heuristics for such difficult problem instances

Outline

1. Models
   - Application model
   - Platform and communication models
2. Multi-criteria scheduling problems
   - Stage types and replication
   - Rule of the game
   - Optimization criteria
   - Define and classify problems
3. Complexity results
   - Mono-criterion problems
   - Bi-criteria problems
4. Conclusion

Related work

Subhlok and Vondran: Pipeline on hom platforms: extended

Chains-to-chains: Heterogeneous, replicate/data-parallelize

Qishi Wu et al: Directed platform graphs (WAN); unbounded multi-port with overlap; mono-criterion problems

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

Scheduling task graphs on heterogeneous platforms: Acyclic task graphs scheduled on different speed processors [Topcuoglu et al.]. Communication contention: one-port model [Beaumont et al.]

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

Conclusion

- Definition of the ingredients of scheduling: applications, platforms, multi-criteria objective functions
- Surprisingly difficult problems: given a mapping, how to order communications to obtain the optimal period?
- Replication for performance and general mappings add one level of difficulty
- Cases in which application throughput not dictated by a critical resource

- Full mono-criterion complexity study, hints of multi-criteria complexity results, linear program formulation

Conclusion

- Definition of the ingredients of scheduling: applications, platforms, multi-criteria objective functions
- Surprisingly difficult problems: given a mapping, how to order communications to obtain the optimal period?
- Replication for performance and general mappings add one level of difficulty
- Cases in which application throughput not dictated by a critical resource
- Full mono-criterion complexity study, hints of multi-criteria complexity results, linear program formulation

Other results

- Extension to dynamic platforms, or how to handle uncertainties?
  - Markovian-based model to compute the throughput of a given mapping with PEPA, performance evaluation process algebra
  - More accurate capture of the behavior with non-markovian model based on timed Petri nets: identification of non-critical resource cases
  - Failure probability related to time: problems become incredibly difficult
- Extension to more complex applications
  - Web service applications with filtering property on stages: same challenges as for standard pipelined applications
  - Results extended for fork or fork-join graphs, additional complexity for general DAGs
  - More complex problems of replica placement optimization, and in-network stream processing applications

Other results

- Extension to dynamic platforms, or how to handle uncertainties?
  - Markovian-based model to compute the throughput of a given mapping with PEPA, performance evaluation process algebra
  - More accurate capture of the behavior with non-markovian model based on timed Petri nets: identification of non-critical resource cases
  - Failure probability related to time: problems become incredibly difficult
- Extension to more complex applications
  - Web service applications with filtering property on stages: same challenges as for standard pipelined applications
  - Results extended for fork or fork-join graphs, additional complexity for general DAGs
  - More complex problems of replica placement optimization, and in-network stream processing applications

Other results

- Extension to dynamic platforms, or how to handle uncertainties?
  - Markovian-based model to compute the throughput of a given mapping with PEPA, performance evaluation process algebra
  - More accurate capture of the behavior with non-markovian model based on timed Petri nets: identification of non-critical resource cases
  - Failure probability related to time: problems become incredibly difficult
- Extension to more complex applications
  - Web service applications with filtering property on stages: same challenges as for standard pipelined applications
  - Results extended for fork or fork-join graphs, additional complexity for general DAGs
  - More complex problems of replica placement optimization, and in-network stream processing applications

Other results

- Extension to dynamic platforms, or how to handle uncertainties?
  - Markovian-based model to compute the throughput of a given mapping with PEPA, performance evaluation process algebra
  - More accurate capture of the behavior with non-markovian model based on timed Petri nets: identification of non-critical resource cases
  - Failure probability related to time: problems become incredibly difficult
- Extension to more complex applications
  - Web service applications with filtering property on stages: same challenges as for standard pipelined applications
  - Results extended for fork or fork-join graphs, additional complexity for general DAGs
  - More complex problems of replica placement optimization, and in-network stream processing applications

On-going and future work

- Experiments on linear chain applications: design of multi-criteria heuristics and experiments on real applications such as a pipelined-version of MPEG-4 encoder
- Other research directions on linear chains:
  - Complexity of period and latency minimization once a mapping is given
  - Multi-application setting and energy minimization
  - Trade-offs between replication for reliability and deal replication
- New applications: Filtering applications, micro-factories with task failures

On-going and future work

- Experiments on linear chain applications: design of multi-criteria heuristics and experiments on real applications such as a pipelined-version of MPEG-4 encoder
- Other research directions on linear chains:
  - Complexity of period and latency minimization once a mapping is given
  - Multi-application setting and energy minimization
  - Trade-offs between replication for reliability and deal replication
- New applications: Filtering applications, micro-factories with task failures

On-going and future work

- Experiments on linear chain applications: design of multi-criteria heuristics and experiments on real applications such as a pipelined-version of MPEG-4 encoder
- Other research directions on linear chains:
  - Complexity of period and latency minimization once a mapping is given
  - Multi-application setting and energy minimization
  - Trade-offs between replication for reliability and deal replication
- New applications: Filtering applications, micro-factories with task failures

Future work

Dynamic platforms and variability

- StochaGrid and ALEAE projects
- Adding non-determinism to the timed Petri net model
- Extend work with more sophisticated failure model to heterogeneous platforms

Come up with a good and realistic model
for platform failure and variability

Future work

Dynamic platforms and variability

- StochaGrid and ALEAE projects
- Adding non-determinism to the timed Petri net model
- Extend work with more sophisticated failure model to heterogeneous platforms

Come up with a good and realistic model
for platform failure and variability

Future work

Dynamic platforms and variability

- StochaGrid and ALEAE projects
- Adding non-determinism to the timed Petri net model
- Extend work with more sophisticated failure model to heterogeneous platforms

Come up with a good and realistic model
for platform failure and variability

Future work

Dynamic platforms and variability

- StochaGrid and ALEAE projects
- Adding non-determinism to the timed Petri net model
- Extend work with more sophisticated failure model to heterogeneous platforms

Come up with a good and realistic model
for platform failure and variability

Lessons learnt

- Today, resources are abundant and ubiquitous, and **demand-driven scheduling** will do the job

Lessons learnt

- Today, resources are abundant and ubiquitous, and **demand-driven scheduling** will do the job

Don't believe that!

Lessons learnt

- ~~Today, resources are abundant and ubiquitous, and **demand-driven scheduling** will do the job~~

  Don't believe that!

- Dynamic scheduler lacks efficiency, *resource selection* is mandatory (remember first motivating example): greedy or random scheduler is likely to fail

Lessons learnt

- ~~Today, resources are abundant and ubiquitous, and **demand-driven scheduling** will do the job~~

  Don't believe that!

- Dynamic scheduler lacks efficiency, *resource selection* is mandatory (remember first motivating example): greedy or random scheduler is likely to fail

  $\Rightarrow$ Need of a smart **static scheduler**

Lessons learnt

- ~~Today, resources are abundant and ubiquitous, and **demand-driven scheduling** will do the job~~

  Don't believe that!

- Dynamic scheduler lacks efficiency, *resource selection* is mandatory (remember first motivating example): greedy or random scheduler is likely to fail
  $\Rightarrow$ Need of a smart **static scheduler**

- New architectures: hierarchy of multicores, virtualization
  $\Rightarrow$ Further need of **even smarter schedulers**

Lessons learnt

- ~~Today, resources are abundant and ubiquitous, and **demand-driven scheduling** will do the job~~

Don't believe that!

- Dynamic scheduler lacks efficiency, *resource selection* is mandatory (remember first motivating example): greedy or random scheduler is likely to fail
  $\Rightarrow$ Need of a smart **static scheduler**
- New architectures: hierarchy of multicores, virtualization
  $\Rightarrow$ Further need of **even smarter schedulers**

Important to work on this subject, many new challenges

Thanks to all my co-authors

- Fanny Dufossé, Matthieu Gallet, Loris Marchal, Jean-François Pineau, Veronika Rehn-Sonigo, Yves Robert, Eric Thierry, and Frédéric Vivien
  from LIP, ENS Lyon, France
- Alexandru Dobrila, Jean-Marc Nicod, and Laurent Philippe
  from University of Franche-Comté, France
- Leonardo Brenner, Bruno Gaujal, and Brigitte Plateau
  from LIG, Grenoble, France
- Mourad Hakem, IUT Belfort, France
- John Chick, Murray Cole, Stephen Gilmore, and Jane Hillston
  from University of Edinburgh, UK
- Kunal Agrawal, MIT, USA
- Marco Aldinucci, University of Torino, Italy
- Henri Casanova, University of Hawai'i, USA
- Jan Duennweber, University of Muenster, Germany
- Paulo Fernandes, PUCRS, Porto Alegre, Brazil
- Sergei Gorlatch, University of Muenster, Germany
- Harald Kosch, Univerity Passau, Germany
- Arnold Rosenberg, Colorado State University, USA
- William J. Stewart, NCSU, Raleigh, USA