

Independent and Divisible Tasks Scheduling on Heterogeneous Star-shaped Platforms with Limited Memory

Olivier Beaumont, Arnaud Legrand, Loris Marchal, Yves Robert

Laboratoire de l'Informatique du Parallélisme
École Normale Supérieure de Lyon, France

EUROMICRO PDP 2005

Outline

Master-Slave Paradigm to Schedule Independent Tasks

Introducing Memory Constraints - Complexity Results

Approximation Algorithms and Heuristics

Variations

Throughput Maximization

Divisible Load

Conclusion

Outline

Master-Slave Paradigm to Schedule Independent Tasks

Introducing Memory Constraints - Complexity Results

Approximation Algorithms and Heuristics

Variations

Throughput Maximization

Divisible Load

Conclusion

Outline

Master-Slave Paradigm to Schedule Independent Tasks

Introducing Memory Constraints - Complexity Results

Approximation Algorithms and Heuristics

Variations

Throughput Maximization

Divisible Load

Conclusion

Outline

Master-Slave Paradigm to Schedule Independent Tasks

Introducing Memory Constraints - Complexity Results

Approximation Algorithms and Heuristics

Variations

- Throughput Maximization

- Divisible Load

Conclusion

Outline

Master-Slave Paradigm to Schedule Independent Tasks

Introducing Memory Constraints - Complexity Results

Approximation Algorithms and Heuristics

Variations

- Throughput Maximization

- Divisible Load

Conclusion

Outline

Master-Slave Paradigm to Schedule Independent Tasks

Introducing Memory Constraints - Complexity Results

Approximation Algorithms and Heuristics

Variations

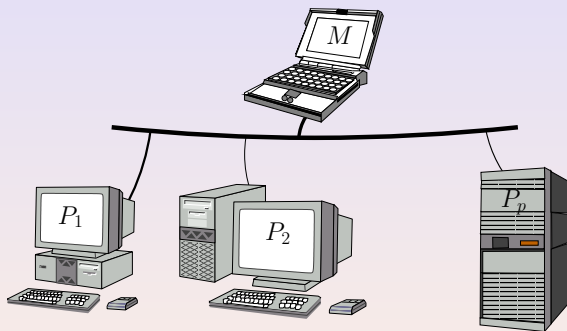
- Throughput Maximization

- Divisible Load

Conclusion

Independent Tasks Scheduling

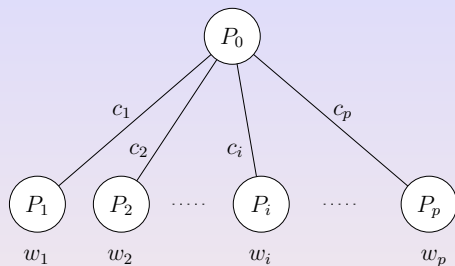
- ▶ A set of *independent identical* tasks to be processed by some slaves. How to get the best performances on an heterogeneous set of workstations ?



- ▶ **Applications:** cellular micro-physiology, protein conformations, particle detection, ...

Heterogeneous stars

- ▶ c_i = transfer time of one task to P_i
- ▶ w_i = processing time of one task on P_i
- ▶ 1-port communications
- ▶ Computation and communication overlap

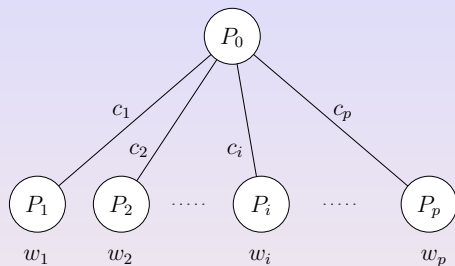


Given a master-slave platform $(c_i, w_i)_{1 \leq i \leq p}$, what is the minimum time needed to process n tasks?

Can be solved in time $O(n^2 p^2)$ with a non-trivial greedy algorithm [BLR02].

Heterogeneous stars

- ▶ c_i = transfer time of one task to P_i
- ▶ w_i = processing time of one task on P_i
- ▶ 1-port communications
- ▶ Computation and communication overlap

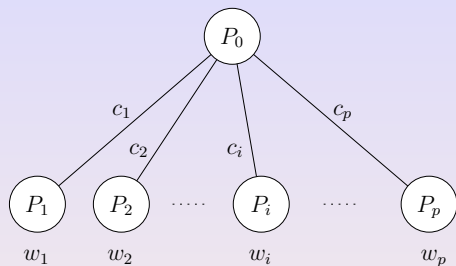


Given a master-slave platform $(c_i, w_i)_{1 \leq i \leq p}$, what is the minimum time needed to process n tasks?

Can be solved in time $O(n^2 p^2)$ with a non-trivial greedy algorithm [BLR02].

Heterogeneous stars

- ▶ c_i = transfer time of one task to P_i
- ▶ w_i = processing time of one task on P_i
- ▶ 1-port communications
- ▶ Computation and communication overlap

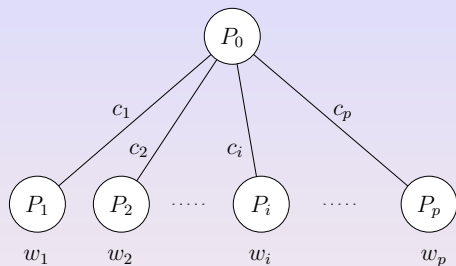


Given a master-slave platform $(c_i, w_i)_{1 \leq i \leq p}$, what is the minimum time needed to process n tasks ?

Can be solved in time $O(n^2 p^2)$ with a non-trivial greedy algorithm [BLR02].

Heterogeneous stars

- ▶ c_i = transfer time of one task to P_i
- ▶ w_i = processing time of one task on P_i
- ▶ 1-port communications
- ▶ Computation and communication overlap

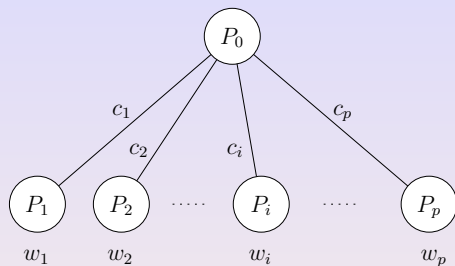


Given a master-slave platform $(c_i, w_i)_{1 \leq i \leq p}$, what is the minimum time needed to process n tasks ?

Can be solved in time $O(n^2 p^2)$ with a non-trivial greedy algorithm [BLR02].

Heterogeneous stars

- ▶ c_i = transfer time of one task to P_i
- ▶ w_i = processing time of one task on P_i
- ▶ 1-port communications
- ▶ Computation and communication overlap

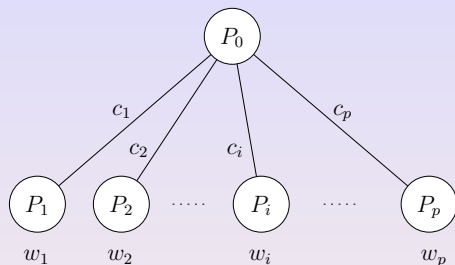


Given a master-slave platform $(c_i, w_i)_{1 \leq i \leq p}$, what is the minimum time needed to process n tasks ?

Can be solved in time $O(n^2 p^2)$ with a non-trivial greedy algorithm [BLR02].

Heterogeneous stars

- ▶ c_i = transfer time of one task to P_i
- ▶ w_i = processing time of one task on P_i
- ▶ 1-port communications
- ▶ Computation and communication overlap



Given a master-slave platform $(c_i, w_i)_{1 \leq i \leq p}$, what is the minimum time needed to process n tasks ?

Can be solved in time $O(n^2 p^2)$ with a **non-trivial greedy algorithm** [BLR02].

Outline

Master-Slave Paradigm to Schedule Independent Tasks

Introducing Memory Constraints - Complexity Results

Approximation Algorithms and Heuristics

Variations

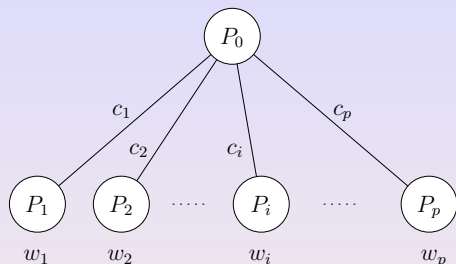
Throughput Maximization

Divisible Load

Conclusion

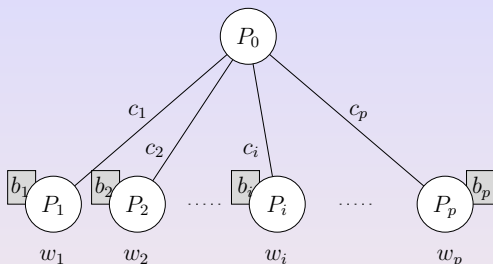
Introducing Memory Constraints

- ▶ c_i = transfer time of one task to P_i
- ▶ w_i = processing time of one task on P_i
- ▶ 1-port communications
- ▶ Computation and communication overlap
 - ▶ b_i = size of the buffer in P_i
maximum number of tasks that P_i can hold simultaneously



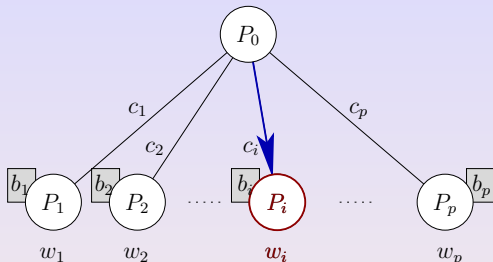
Introducing Memory Constraints

- ▶ c_i = transfer time of one task to P_i
- ▶ w_i = processing time of one task on P_i
- ▶ 1-port communications
- ▶ Computation and communication overlap
 - ▶ b_i = size of the buffer in P_i
maximum number of tasks that P_i can be hold simultaneously

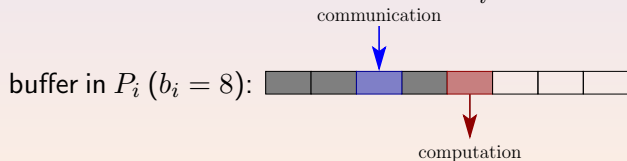


Introducing Memory Constraints

- ▶ c_i = transfer time of one task to P_i
- ▶ w_i = processing time of one task on P_i
- ▶ 1-port communications
- ▶ Computation and communication overlap



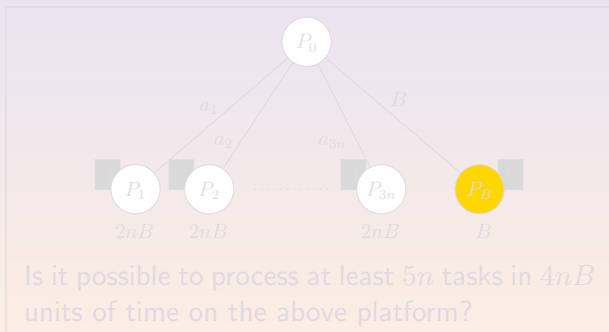
- ▶ b_i = size of the buffer in P_i
maximum number of tasks that P_i can be hold simultaneously



Introducing Memory Constraints

- ▶ suppose a bounded buffer of tasks for each processor
⇒ hard problem on star platforms!
- ▶ we prove that this problem is **strongly NP-hard** from **3-Dimensional matching**

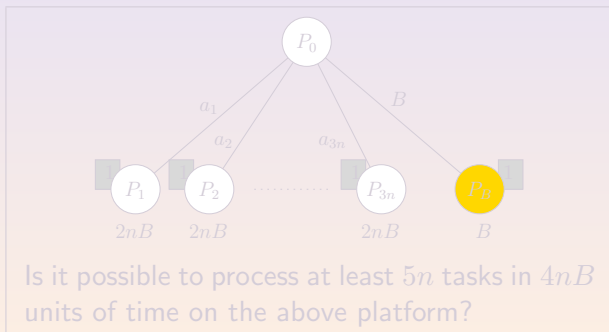
Given $3n$ integers a_1, \dots, a_{3n} and an integer B , is there a partition of the a_i 's into n groups of 3 integers, such that each group sums to B ?



Introducing Memory Constraints

- ▶ suppose a bounded buffer of tasks for each processor
⇒ hard problem on star platforms!
- ▶ we prove that this problem is **strongly NP-hard** from **3-Dimensional matching**

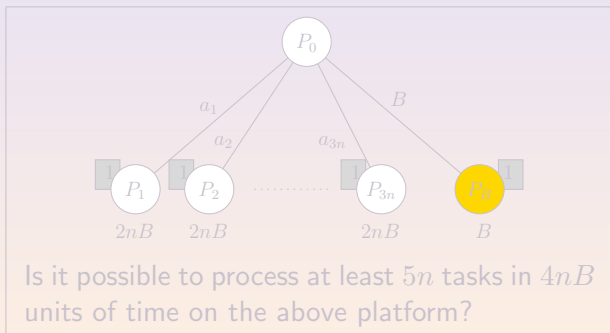
Given $3n$ integers a_1, \dots, a_{3n} and an integer B , is there a partition of the a_i 's into n groups of 3 integers, such that each group sums to B ?



Introducing Memory Constraints

- ▶ suppose a bounded buffer of tasks for each processor
⇒ hard problem on star platforms!
- ▶ we prove that this problem is **strongly NP-hard** from **3-Dimensional matching**

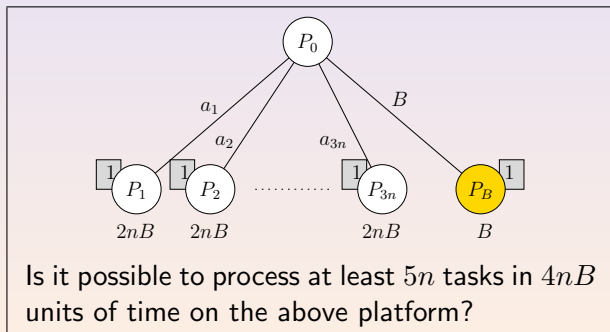
Given $3n$ integers a_1, \dots, a_{3n} and an integer B , is there a partition of the a_i 's into n groups of 3 integers, such that each group sums to B ?



Introducing Memory Constraints

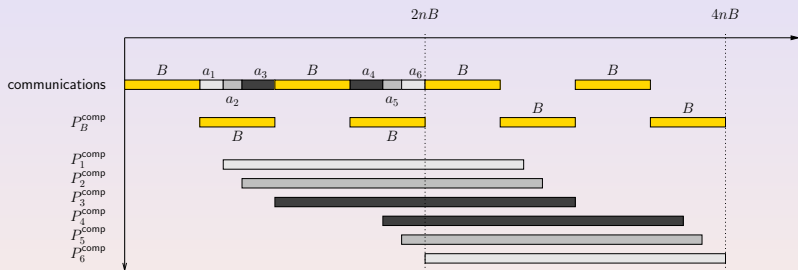
- ▶ suppose a bounded buffer of tasks for each processor
⇒ hard problem on star platforms!
- ▶ we prove that this problem is **strongly NP-hard** from **3-Dimensional matching**

Given $3n$ integers a_1, \dots, a_{3n} and an integer B , is there a partition of the a_i 's into n groups of 3 integers, such that each group sums to B ?



Introducing Memory constraints

Ordering the communications is hard !



Outline

Master-Slave Paradigm to Schedule Independent Tasks

Introducing Memory Constraints - Complexity Results

Approximation Algorithms and Heuristics

Variations

Throughput Maximization

Divisible Load

Conclusion

A simple 2-approximation

worse case: buffers holding only 1 task

⇒ no communication/computation overlap

n_i = number of task that P_i would process in the optimal solution without memory constraint

A task is sent to P_i as soon as

- ▶ the communication medium is free
- ▶ P_i is idle
- ▶ P_i has processed less than n_i tasks

This simple list scheduling is a 2-approximation: the makespan cannot be larger than twice the makespan of the optimal schedule without memory limitations.

A simple 2-approximation

worse case: buffers holding only 1 task

⇒ no communication/computation overlap

n_i = number of task that P_i would process in the optimal solution without memory constraint

A task is sent to P_i as soon as

- ▶ the communication medium is free
- ▶ P_i is idle
- ▶ P_i has processed less than n_i tasks

This simple list scheduling is a 2-approximation: the makespan cannot be larger than twice the makespan of the optimal schedule without memory limitations.

A simple 2-approximation

worse case: buffers holding only 1 task

⇒ no communication/computation overlap

n_i = number of task that P_i would process in the optimal solution without memory constraint

A task is sent to P_i as soon as

- ▶ the communication medium is free
- ▶ P_i is idle
- ▶ P_i has processed less than n_i tasks

This simple list scheduling is a 2-approximation: the makespan cannot be larger than twice the makespan of the optimal schedule without memory limitations.

A simple 2-approximation

worse case: buffers holding only 1 task
 \implies no communication/computation overlap

n_i = number of task that P_i would process in the optimal solution without memory constraint

A task is sent to P_i as soon as

- ▶ the communication medium is free
- ▶ P_i is idle
- ▶ P_i has processed less than n_i tasks

This simple **list scheduling** is a **2-approximation**: the makespan cannot be larger than twice the makespan of the optimal schedule without memory limitations.

Heuristic design - 1

- ▶ List-based heuristics
- ▶ Schedule a task as soon as possible

How to choose between several available processors ?

Different selection functions:

- ▶ \min_c : choose the smallest communication time
- ▶ \min_w : choose the smallest processing time
- ▶ mct : choose the processor that will finish this task the sooner

Heuristic design - 1

- ▶ List-based heuristics
- ▶ Schedule a task as soon as possible

How to choose between several available processors ?

Different selection functions:

- ▶ \min_c : choose the smallest communication time
- ▶ \min_w : choose the smallest processing time
- ▶ mct : choose the processor that will finish this task the sooner

Heuristic design - 1

- ▶ List-based heuristics
- ▶ Schedule a task as soon as possible

How to choose between several available processors ?

Different selection functions:

- ▶ min_c : choose the smallest communication time
- ▶ min_w : choose the smallest processing time
- ▶ mct : choose the processor that will finish this task the sooner

Heuristic design - 1

- ▶ List-based heuristics
- ▶ Schedule a task as soon as possible

How to choose between several available processors ?

Different selection functions:

- ▶ min_c : choose the smallest communication time
- ▶ min_w : choose the smallest processing time
- ▶ mct : choose the processor that will finish this task the sooner

Heuristic design - 1

- ▶ List-based heuristics
- ▶ Schedule a task as soon as possible

How to choose between several available processors ?

Different selection functions:

- ▶ min_c : choose the smallest communication time
- ▶ min_w : choose the smallest processing time
- ▶ mct : choose the processor that will finish this task the sooner

Heuristic design - 1

- ▶ List-based heuristics
- ▶ Schedule a task as soon as possible

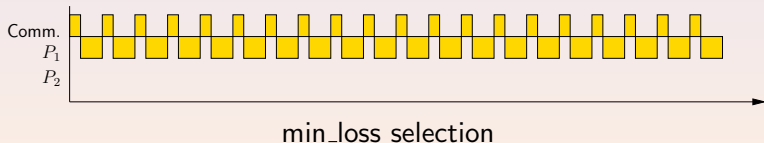
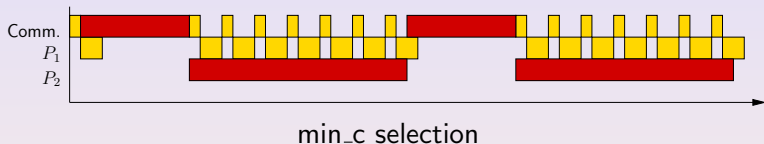
How to choose between several available processors ?

Different selection functions:

- ▶ min_c : choose the smallest communication time
- ▶ min_w : choose the smallest processing time
- ▶ mct : choose the processor that will finish this task the sooner

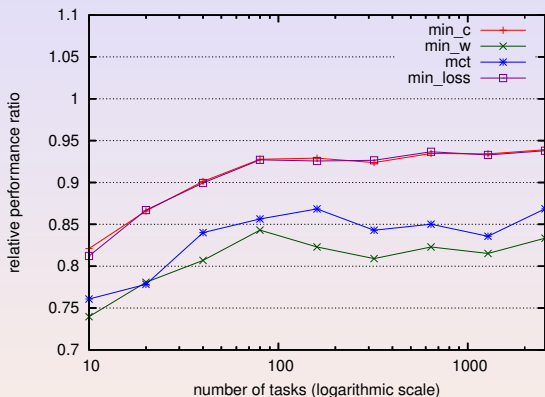
Heuristic design

- ▶ `min_loss` : send to the processor that avoids the most *starvation* of other processors (not a real list algorithm though).



Simulation study: Bandwidth-centric

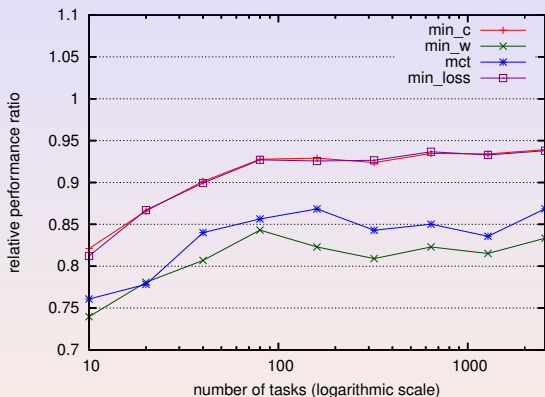
- ▶ 1 buffer (no possible overlap)
- ▶ comparison with an absolute upper bound.



- ▶ network = critical resource
- ▶ bandwidth-centric strategies give best results

Simulation study: Bandwidth-centric

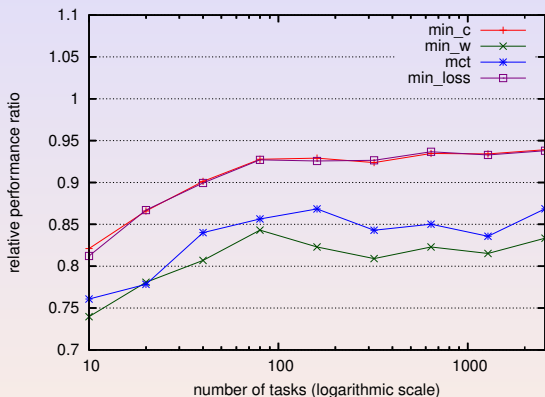
- ▶ 1 buffer (no possible overlap)
- ▶ comparison with an absolute upper bound.



- ▶ network = critical resource
- ▶ bandwidth-centric strategies give best results

Simulation study: Bandwidth-centric

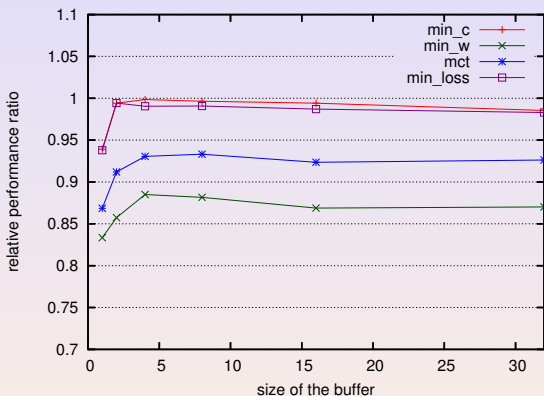
- ▶ 1 buffer (no possible overlap)
- ▶ comparison with an absolute upper bound.



- ▶ network = critical resource
- ▶ bandwidth-centric strategies give best results

Simulation study: 2 buffers are sufficient

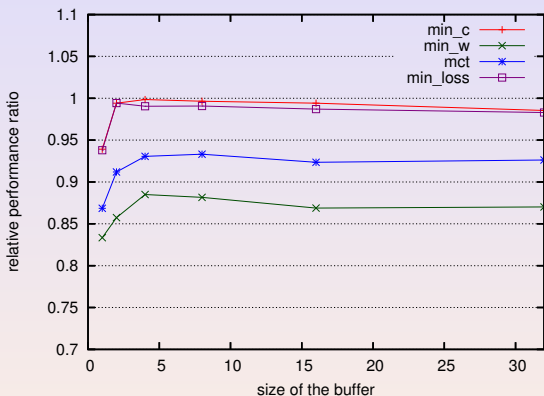
- ▶ for 1200 tasks
- ▶ comparison with an absolute upper bound.



- ▶ 2 buffers are sufficient in most situations
- ▶ Other studies show similar results [CCFK03]

Simulation study: 2 buffers are sufficient

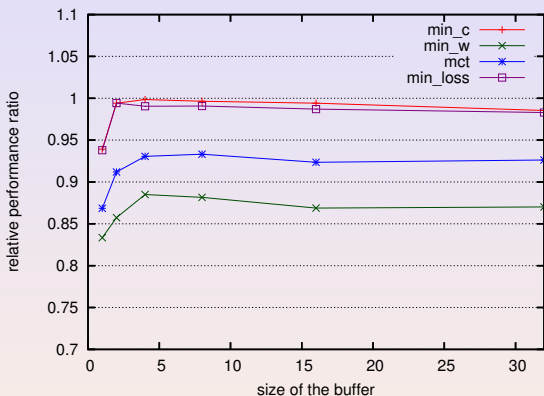
- ▶ for 1200 tasks
- ▶ comparison with an absolute upper bound.



- ▶ 2 buffers are sufficient in most situations
- ▶ Other studies show similar results [CCFK03]

Simulation study: 2 buffers are sufficient

- ▶ for 1200 tasks
- ▶ comparison with an absolute upper bound.



- ▶ 2 buffers are sufficient in most situations
- ▶ Other studies show similar results [CCFK03]

Outline

Master-Slave Paradigm to Schedule Independent Tasks

Introducing Memory Constraints - Complexity Results

Approximation Algorithms and Heuristics

Variations

Throughput Maximization

Divisible Load

Conclusion

Throughput maximization

hardness may come from the metric **makespan**:

- ▶ setting up such a platform is worthwhile only if the application is large: big task number, long execution time
- ▶ running 5h or 5h02 is equivalent
- ▶ modeling a large distributed computing platform is hard and the accuracy of the parameters is even harder to guarantee;

concentrate steady-state, try to maximize throughput

steady-state notations: average quantities over time

- ▶ α_j : number of tasks processed by P_j per time-unit;

Throughput maximization

hardness may come from the metric **makespan**:

- ▶ setting up such a platform is worthwhile only if the application is large: big task number, long execution time
- ▶ running 5h or 5h02 is equivalent
- ▶ modeling a large distributed computing platform is hard and the accuracy of the parameters is even harder to guarantee;

concentrate **steady-state**, try to maximize **throughput**

steady-state notations: average quantities over time

- ▶ α_j : number of tasks processed by P_j per time-unit;

Throughput maximization

hardness may come from the metric **makespan**:

- ▶ setting up such a platform is worthwhile only if the application is large: big task number, long execution time
- ▶ running 5h or 5h02 is equivalent
- ▶ modeling a large distributed computing platform is hard and the accuracy of the parameters is even harder to guarantee;

concentrate **steady-state**, try to maximize **throughput**

steady-state notations: average quantities over time

- ▶ α_j : number of tasks processed by P_j per time-unit;

Throughput maximization

hardness may come from the metric **makespan**:

- ▶ setting up such a platform is worthwhile only if the application is large: big task number, long execution time
- ▶ running 5h or 5h02 is equivalent
- ▶ modeling a large distributed computing platform is hard and the accuracy of the parameters is even harder to guarantee;

concentrate **steady-state**, try to maximize **throughput**

steady-state notations: average quantities over time

- ▶ α_i : number of tasks processed by P_i per time-unit;

Throughput maximization

hardness may come from the metric **makespan**:

- ▶ setting up such a platform is worthwhile only if the application is large: big task number, long execution time
- ▶ running 5h or 5h02 is equivalent
- ▶ modeling a large distributed computing platform is hard and the accuracy of the parameters is even harder to guarantee;

concentrate **steady-state**, try to maximize **throughput**

steady-state notations: average quantities over time

- ▶ α_i : number of tasks processed by P_i per time-unit;

Throughput maximization

hardness may come from the metric **makespan**:

- ▶ setting up such a platform is worthwhile only if the application is large: big task number, long execution time
- ▶ running 5h or 5h02 is equivalent
- ▶ modeling a large distributed computing platform is hard and the accuracy of the parameters is even harder to guarantee;

concentrate **steady-state**, try to maximize **throughput**

steady-state notations: average quantities over time

- ▶ α_i : number of tasks processed by P_i per time-unit;

Bounding with Linear Programming

Any schedule verifies:

- ▶ $\forall i \alpha_i \times w_i \leq 1$ (processing time)
- ▶ $\forall i \sum_j \alpha_i \times c_{i,j} \leq 1$ (emissions are sequential)

$$\text{throughput} = \sum_i \alpha_i$$

smaller than the solution of the LP:

$$\begin{aligned} & \text{MAXIMIZE } \sum_i \alpha_i, \\ & \text{SUCH THAT} \\ & \begin{cases} \forall i \alpha_i \times w_i \leq 1 \\ \forall i \sum_j \text{sent}(P_i \rightarrow P_j) \times c_{i,j} \leq 1 \end{cases} \end{aligned}$$

Bounding with Linear Programming

Any schedule verifies:

- ▶ $\forall i \alpha_i \times w_i \leq 1$ (processing time)
- ▶ $\forall i \sum_j \alpha_i \times c_{i,j} \leq 1$ (emissions are sequential)

$$\text{throughput} = \sum_i \alpha_i$$

smaller than the solution of the LP:

$$\begin{array}{l} \text{MAXIMIZE } \sum_i \alpha_i, \\ \text{SUCH THAT} \\ \left\{ \begin{array}{l} \forall i \alpha_i \times w_i \leq 1 \\ \forall i \sum_j \text{sent}(P_i \rightarrow P_j) \times c_{i,j} \leq 1 \end{array} \right. \end{array}$$

Bounding with Linear Programming

Any schedule verifies:

- ▶ $\forall i \alpha_i \times w_i \leq 1$ (processing time)
- ▶ $\forall i \sum_j \alpha_i \times c_{i,j} \leq 1$ (emissions are sequential)

$$\text{throughput} = \sum_i \alpha_i$$

smaller than the solution of the LP:

$$\begin{array}{l} \text{MAXIMIZE } \sum_i \alpha_i, \\ \text{SUCH THAT} \\ \left\{ \begin{array}{l} \forall i \alpha_i \times w_i \leq 1 \\ \forall i \sum_j \text{sent}(P_i \rightarrow P_j) \times c_{i,j} \leq 1 \end{array} \right. \end{array}$$

Building an optimal periodic schedule

- ▶ from optimal solution of LP \Rightarrow **periodic** schedule with **optimal throughput**
- ▶ schedule **asymptotically optimal** with respect to the makespan (neglect initialization and clean-up phases)

approach much powerful than needed here: enables to target

- ▶ arbitrarily complex platform graphs (not only star-shaped)
- ▶ tasks with dependencies (DAGs with bounded depth [BLMR03])

however, the size of the period may be large,
which incurs big memory overhead

Building an optimal periodic schedule

- ▶ from optimal solution of LP \Rightarrow **periodic** schedule with **optimal throughput**
- ▶ schedule **asymptotically optimal** with respect to the makespan (neglect initialization and clean-up phases)

approach much powerful than needed here: enables to target

- ▶ arbitrarily complex platform graphs (not only star-shaped)
- ▶ tasks with dependencies (DAGs with bounded depth [BLMR03])

however, the size of the period may be large,
which incurs big memory overhead

Building an optimal periodic schedule

- ▶ from optimal solution of LP \Rightarrow **periodic** schedule with **optimal throughput**
- ▶ schedule **asymptotically optimal** with respect to the makespan (neglect initialization and clean-up phases)

approach much powerful than needed here: enables to target

- ▶ arbitrarily complex platform graphs (not only star-shaped)
- ▶ tasks with dependencies (DAGs with bounded depth [BLMR03])

however, the size of the period may be large,
which incurs big memory overhead

Building an optimal periodic schedule

- ▶ from optimal solution of LP \Rightarrow **periodic** schedule with **optimal throughput**
- ▶ schedule **asymptotically optimal** with respect to the makespan (neglect initialization and clean-up phases)

approach much powerful than needed here: enables to target

- ▶ arbitrarily complex platform graphs (not only star-shaped)
- ▶ tasks with dependencies (DAGs with bounded depth [BLMR03])

however, the size of the period may be large,
which incurs big memory overhead

Building an optimal periodic schedule

- ▶ from optimal solution of LP \Rightarrow **periodic** schedule with **optimal throughput**
- ▶ schedule **asymptotically optimal** with respect to the makespan (neglect initialization and clean-up phases)

approach much powerful than needed here: enables to target

- ▶ arbitrarily complex platform graphs (not only star-shaped)
- ▶ tasks with dependencies (DAGs with bounded depth [BLMR03])

however, the size of the period may be large,
which incurs **big memory overhead**

Building an optimal periodic schedule

- ▶ from optimal solution of LP \Rightarrow **periodic** schedule with **optimal throughput**
- ▶ schedule **asymptotically optimal** with respect to the makespan (neglect initialization and clean-up phases)

approach much powerful than needed here: enables to target

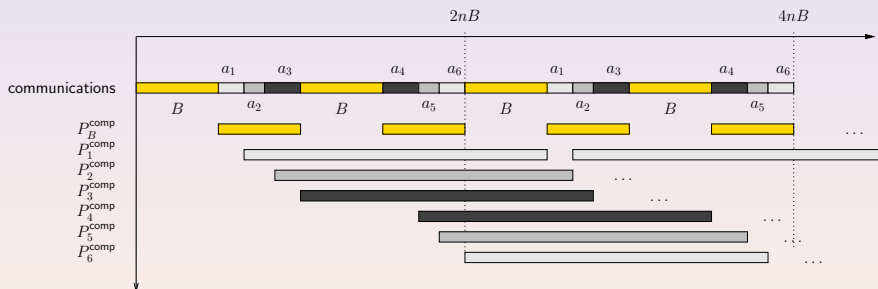
- ▶ arbitrarily complex platform graphs (not only star-shaped)
- ▶ tasks with dependencies (DAGs with bounded depth [BLMR03])

however, the size of the period may be large,
which incurs **big memory overhead**

What about the steady-state optimization ?

same problem as in the makespan optimization case:
with bounded buffer on each processor

The problems gets strongly NP-hard again. ☹️



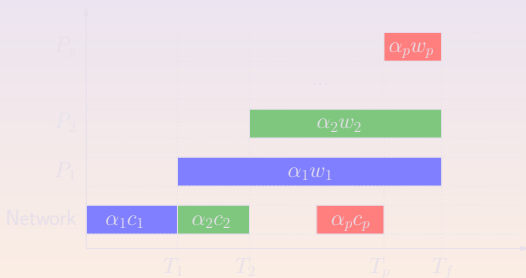
Divisible load theory: one-round

- divisible task: ▶ perfectly parallel job
▶ can be arbitrarily split into several independent parts

Linear model:

- ▶ Transfer time of X units of load to P_i : $c_i \times X$
▶ Processing time of X units of load to P_i : $w_i \times X$

One round distribution:



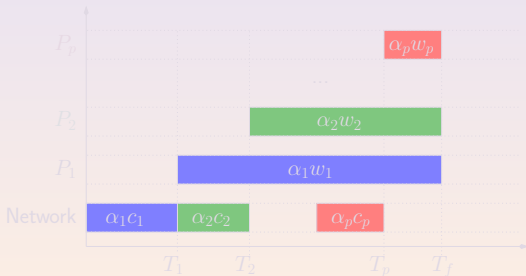
Divisible load theory: one-round

- divisible task: ▶ perfectly parallel job
▶ can be arbitrarily split into several independent parts

Linear model:

- ▶ Transfer time of X units of load to P_i : $c_i \times X$
- ▶ Processing time of X units of load to P_i : $w_i \times X$

One round distribution:



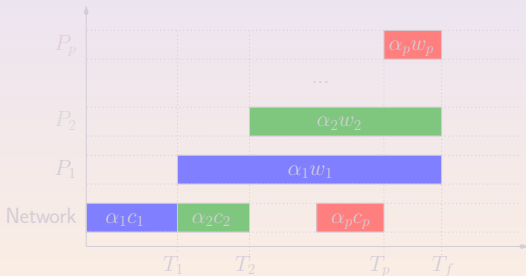
Divisible load theory: one-round

- divisible task: ▶ perfectly parallel job
▶ can be arbitrarily split into several independent parts

Linear model:

- ▶ Transfer time of X units of load to P_i : $c_i \times X$
- ▶ Processing time of X units of load to P_i : $w_i \times X$

One round distribution:



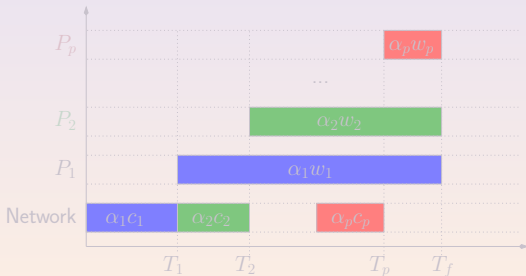
Divisible load theory: one-round

- divisible task:
- ▶ perfectly parallel job
 - ▶ can be arbitrarily split into several independent parts

Linear model:

- ▶ Transfer time of X units of load to P_i : $c_i \times X$
- ▶ Processing time of X units of load to P_i : $w_i \times X$

One round distribution:



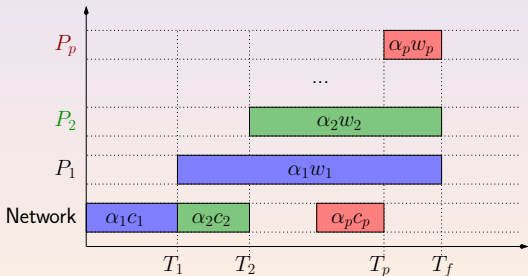
Divisible load theory: one-round

- divisible task: ▶ perfectly parallel job
▶ can be arbitrarily split into several independent parts

Linear model:

- ▶ Transfer time of X units of load to P_i : $c_i \times X$
▶ Processing time of X units of load to P_i : $w_i \times X$

One round distribution:



Heterogeneous stars

One can prove that:

- ▶ all processors should be used;
- ▶ all processors should finish at the same moment;
- ▶ data should be sent by the order of increasing c_i .

without memory limitations:

Processing power does not matter,
only the communication capabilities!

Heterogeneous stars

One can prove that:

- ▶ all processors should be used;
- ▶ all processors should finish at the same moment;
- ▶ data should be sent by the order of increasing c_i .

without memory limitations:

Processing power does not matter,
only the communication capabilities!

Heterogeneous stars

One can prove that:

- ▶ all processors should be used;
- ▶ all processors should finish at the same moment;
- ▶ data should be sent by the order of increasing c_i .

without memory limitations:

Processing power does not matter,
only the communication capabilities!

Heterogeneous stars

One can prove that:

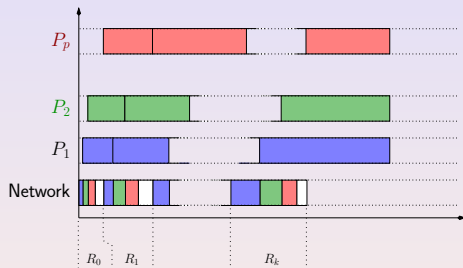
- ▶ all processors should be used;
- ▶ all processors should finish at the same moment;
- ▶ data should be sent by the order of increasing c_i .

without memory limitations:

Processing power does not matter,
only the communication capabilities!

Affine Multiple rounds

- ▶ allow for **multiple rounds** \Rightarrow better results
- ▶ flaw of modeling: infinite number of rounds (with infinitely small chunks) gives optimal results



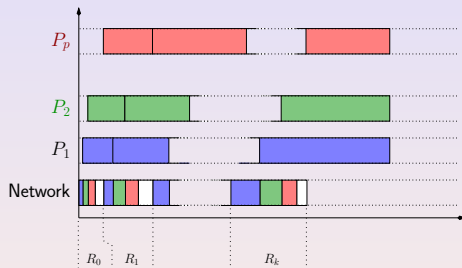
introduce latencies: \rightarrow affine model

- ▶ Transfer time of X units of load to P_i : $C_i + c_i \times X$

complexity of the 1-round (or multi-rounds) affine divisible load distribution is still open!

Affine Multiple rounds

- ▶ allow for **multiple rounds** \Rightarrow better results
- ▶ flaw of modeling: infinite number of rounds (with infinitely small chunks) gives optimal results



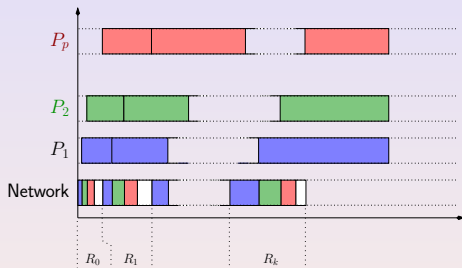
introduce latencies: \rightarrow affine model

- ▶ Transfer time of X units of load to P_i : $C_i + c_i \times X$

complexity of the 1-round (or multi-rounds) affine divisible load distribution is still open!

Affine Multiple rounds

- ▶ allow for **multiple rounds** \Rightarrow better results
- ▶ flaw of modeling: infinite number of rounds (with infinitely small chunks) gives optimal results



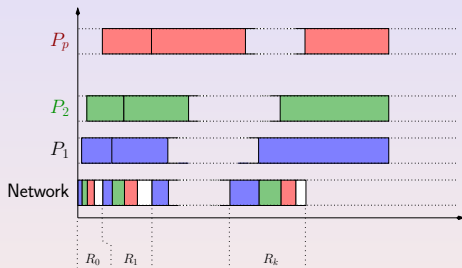
introduce **latencies**: \rightarrow **affine** model

- ▶ Transfer time of X units of load to P_i : $C_i + c_i \times X$

complexity of the 1-round (or multi-rounds) affine divisible load distribution is still open!

Affine Multiple rounds

- ▶ allow for **multiple rounds** \Rightarrow better results
- ▶ flaw of modeling: infinite number of rounds (with infinitely small chunks) gives optimal results



introduce **latencies**: \rightarrow **affine** model

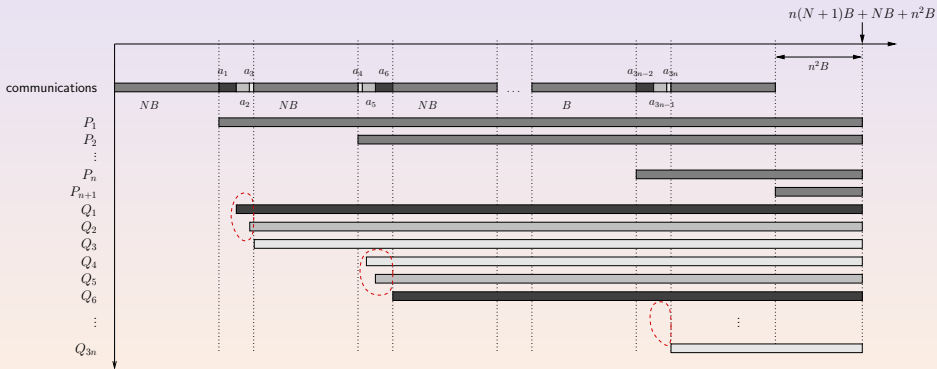
- ▶ Transfer time of X units of load to P_i : $C_i + c_i \times X$

complexity of the 1-round (or multi-rounds) affine divisible load distribution is still open!

Adding memory constraints

once again, adding memory constraints \Rightarrow same problem as for independent tasks:

The problem gets **strongly NP-hard** again, whatever the number of distribution may be.



Outline

Master-Slave Paradigm to Schedule Independent Tasks

Introducing Memory Constraints - Complexity Results

Approximation Algorithms and Heuristics

Variations

- Throughput Maximization

- Divisible Load

Conclusion

Conclusion

- ▶ On a theoretical point of view, memory constraints make all simple problems much harder. Even “classical” relaxation techniques fail to make the problem more tractable.
- ▶ On a practical point of view, if these constraints are not too tight (sufficient to overlap a little bit communications and computations), very efficient simple heuristics can be used.
- ▶ Classical list-based scheduling heuristics that aim at greedily minimizing the completion time of each task are outperformed by the simplest heuristic that consists in delegating data to the available processor that has the smallest communication time, regardless of its computation power.
- ▶ The different models we have been dealing with tend to confirm this trend on a theoretical point of view.

On an heterogeneous collection of workstations, network is the critical resource and it should be handled with care using bandwidth centric approaches.

Conclusion

- ▶ On a theoretical point of view, memory constraints make all simple problems much harder. Even “classical” relaxation techniques fail to make the problem more tractable.
- ▶ On a practical point of view, if these constraints are not too tight (sufficient to overlap a little bit communications and computations), very efficient simple heuristics can be used.
- ▶ Classical list-based scheduling heuristics that aim at greedily minimizing the completion time of each task are outperformed by the simplest heuristic that consists in delegating data to the available processor that has the smallest communication time, regardless of its computation power.
- ▶ The different models we have been dealing with tend to confirm this trend on a theoretical point of view.

On an heterogeneous collection of workstations, network is the critical resource and it should be handled with care using bandwidth centric approaches.

Conclusion

- ▶ On a theoretical point of view, memory constraints make all simple problems much harder. Even “classical” relaxation techniques fail to make the problem more tractable.
- ▶ On a practical point of view, if these constraints are not too tight (sufficient to overlap a little bit communications and computations), very efficient simple heuristics can be used.
- ▶ Classical list-based scheduling heuristics that aim at greedily minimizing the completion time of each task are outperformed by the simplest heuristic that consists in delegating data to the available processor that has the smallest communication time, regardless of its computation power.
- ▶ The different models we have been dealing with tend to confirm this trend on a theoretical point of view.


On an heterogeneous collection of workstations, network is the critical resource and it should be handled with care using bandwidth centric approaches.

Conclusion

- ▶ On a theoretical point of view, memory constraints make all simple problems much harder. Even “classical” relaxation techniques fail to make the problem more tractable.
- ▶ On a practical point of view, if these constraints are not too tight (sufficient to overlap a little bit communications and computations), very efficient simple heuristics can be used.
- ▶ Classical list-based scheduling heuristics that aim at greedily minimizing the completion time of each task are outperformed by the simplest heuristic that consists in delegating data to the available processor that has the smallest communication time, regardless of its computation power.
- ▶ The different models we have been dealing with tend to confirm this trend on a theoretical point of view.

On an heterogeneous collection of workstations, network is the critical resource and it should be handled with care using bandwidth centric approaches.

Bibliography

-  O. Beaumont, A. Legrand, L. Marchal, and Y. Robert.
Scheduling strategies for mixed data and task parallelism on heterogeneous clusters.
Parallel Processing Letters, 13(2), 2003.
-  Olivier Beaumont, Arnaud Legrand, and Yves Robert.
A polynomial-time algorithm for allocating independent tasks on heterogeneous fork-graphs.
In *ISCIS XVII, Seventeenth International Symposium On Computer and Information Sciences*, pages 115–119. CRC Press, 2002.
-  L. Carter, H. Casanova, J. Ferrante, and B. Kreaseck.
Autonomous protocols for bandwidth-centric scheduling of independent-task applications.
In *International Parallel and Distributed Processing Symposium IPDPS'2003*. IEEE Computer Society Press, 2003.
-  Pierre-François Dutot.
Complexity of master-slave tasking on heterogeneous trees.
European Journal of Operational Research, 2003.
Special issue on the Dagstuhl meeting on Scheduling for Computing and Manufacturing systems (to appear).
-  Pierre-François Dutot.
Master-slave tasking on heterogeneous processors.
In *International Parallel and Distributed Processing Symposium IPDPS'2003*. IEEE Computer Society Press, 2003.