On the Impact of Platform Models

EPIT 2007

Arnaud Legrand

CNRS/INRIA, LIG laboratory

June 6, 2007

Motivation

- Scientific computing : large needs in computation or storage resources.
- Need to use systems with “several processors”:
  - Parallel computers with shared/distributed memory
  - Clusters
  - Heterogeneous clusters
  - Clusters of clusters
  - Network of workstations
  - The Grid
  - Desktop Grids
- When modeling platform, communications modeling seems to be the most controversial part.
- Two kinds of people produce communication models: those who are concerned with scheduling and those who are concerned with performance evaluation.
- All these models are imperfect and intractable.

Outline

Part I Platform Model

Part II Scheduling Case Study

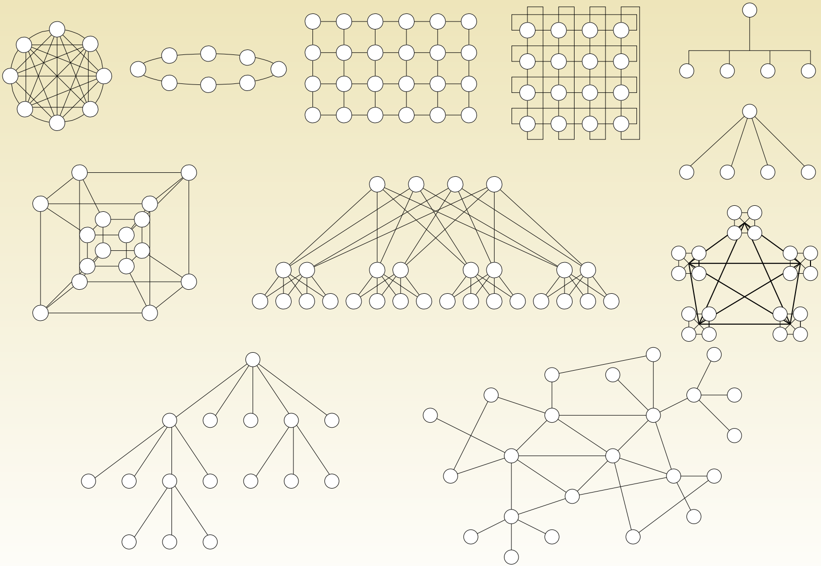
Part I

Platform Model

Outline

1. Topology
2. Point to Point Communication Models
3. Modeling Concurrency
4. Remind This is a Model, Hence Imperfect

Various Topologies Used in the Litterature



Outline

1. Topology
2. Point to Point Communication Models
   - Hockney
   - LogP and Friends
   - TCP
3. Modeling Concurrency
4. Remind This is a Model, Hence Imperfect

UET-UCT

Hem. . . This one is mainly used by scheduling theoreticians to prove that their problem is hard and to know whether there is some hope to prove some clever result or not.

"Hockney" Model

Hockney [Hoc94] proposed the following model for performance evaluation of the Paragon. A message of size m from P_i to P_j requires:

$$t_{i,j}(m) = L_{i,j} + m/B_{i,j}$$

In scheduling, there are three types of "corresponding" models:

- Communications are not "splitable" and each communication k is associated to a communication time t_k (accounting for message size, latency, bandwidth, middleware, ...).
- Communications are "splitable" but latency is considered to be negligible (linear divisible model):

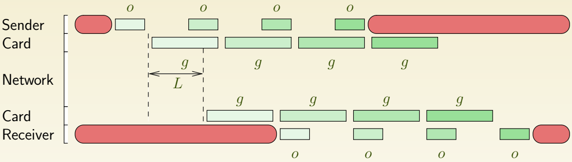
$$t_{i,j}(m) = m/B_{i,j}$$

- Communications are "splitable" and latency cannot be neglected (linear divisible model):

$$t_{i,j}(m) = L_{i,j} + m/B_{i,j}$$

The LogP model [CKP+96] is defined by 4 parameters:

- L is the network latency
- o is the middleware overhead (message splitting and packing, buffer management, connection, ...) for a message of size w
- g is the gap (the minimum time between two packets communication) between two messages of size w
- P is the number of processors/modules

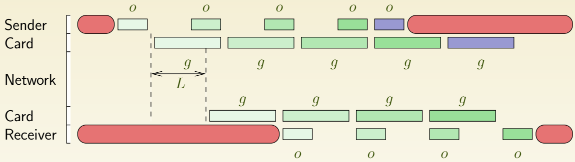


- Sending m bytes with packets of size w:
$$2o + L + \left\lceil \frac{m}{w} \right\rceil \cdot \max(o, g)$$
- Occupation on the sender and on the receiver:
$$o + L + \left(\left\lceil \frac{m}{w} \right\rceil - 1\right) \cdot \max(o, g)$$

LogP

The LogP model [CKP+96] is defined by 4 parameters:

- L is the network latency
- o is the middleware overhead (message splitting and packing, buffer management, connection, ...) for a message of size w
- g is the gap (the minimum time between two packets communication) between two messages of size w
- P is the number of processors/modules



- Sending m bytes with packets of size w:

$$2o + L + \left\lceil \frac{m}{w} \right\rceil \cdot \max(o, g)$$

- Occupation on the sender and on the receiver:

$$o + L + \left(\left\lceil \frac{m}{w} \right\rceil - 1 \right) \cdot \max(o, g)$$

LogGP & pLogP

The previous model works fine for short messages. However, many parallel machines have special support for long messages, hence a higher bandwidth. LogGP [AISS97] is an extension of LogP:

G captures the bandwidth for long messages:

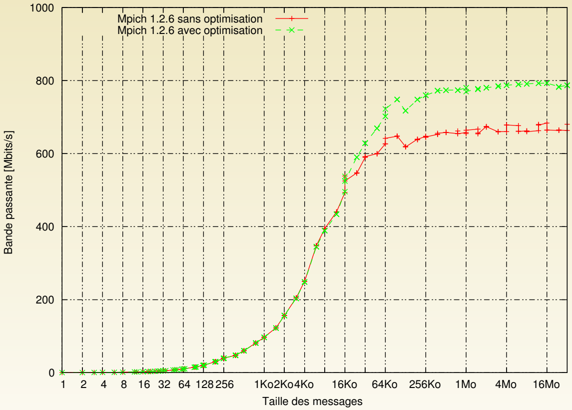
short messages $2o + L + \left\lceil \frac{m}{w} \right\rceil \cdot \max(o, g)$

long messages $2o + L + (m-1)G$

There is no fundamental difference...

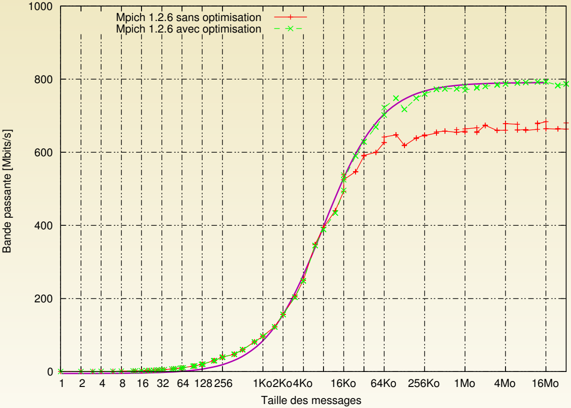
OK, it works for small and large messages. Does it work for average-size messages ? pLogP [KBV00] is an extension of LogP when L, o and g depends on the message size m. They also have introduced a distinction between o_s and o_r. This is more and more precise but concurency is still not taken into account.

With the Hockney model: $\frac{m}{L+m/B}$.



MPICH, TCP with Gigabit Ethernet

With the Hockney model: $\frac{m}{L+m/B}$.



MPICH, TCP with Gigabit Ethernet

What About TCP-based Networks?

The previous models work fine for parallel machines. Most networks use TCP that has fancy flow-control mechanism and slow start. Is it valid to use affine model for such networks?

The answer seems to be yes but latency and bandwidth parameters have to be carefully measured [LQDB05].

- Probing for $m = 1$b and $m = 1$Mb leads to bad results.
- The whole middleware layers should be benchmarked (theoretical latency is useless because of middleware, theoretical bandwidth is useless because of middleware and latency).

The slow-start does not seem to be too harmful.

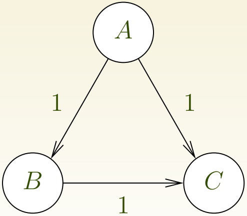
Most people forget that the round-trip time has a huge impact on the bandwidth.

Outline

1. Topology
2. Point to Point Communication Models
3. Modeling Concurrency
   - Multi-port
   - Single-port (Pure and Full Duplex)
   - Flows
4. Remind This is a Model, Hence Imperfect

Multi-ports

- A given processor can communicate with as many other processors as he wishes without any degradation.
- This model is widely used by scheduling theoreticians (think about all DAG with commmunications scheduling problems) to prove that their problem is hard and to know whether there is some hope to prove some clever result or not.
  Some theoreticians feel like this model is borderline, especially when allowing duplication or when trying to design algorithms with super tight approximation ratios [Yves Robert 01–??].
  Frankly, such a model is totally unrealistic.
- Using MPI and synchronous communications, it may not be an issue. However, with multi-core, multi-processor machines, it cannot be ignored...

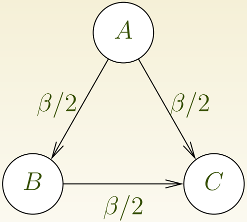


Multi-port

Bounded Multi-port

- Assume now that we have threads or multi-core processors.

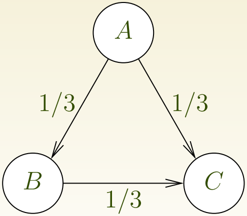
  We can write that sum of the throughputs of all communications (incomming and outgoing). Such a model is OK for wide-area communications [HP04].
- Remember, the bounds due to the round-trip-time must not be forgotten!



Multi-port (β)

Single-port (Pure)

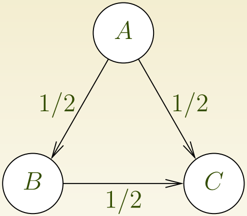
- A process can communicate with only one other process at a time. This constraint is generally written as a constraint on the sum of communication times and is thus rather easy to use in a scheduling context (even though it complexifies problems).
- This model makes sense when using non-threaded versions of communication libraries (e.g., MPI). As soon as you're allowed to use threads, bounded-multiport seems a more reasonnable option (both for performance and scheduling complexity).



1-port (pure)

Single-port (Full-Duplex)

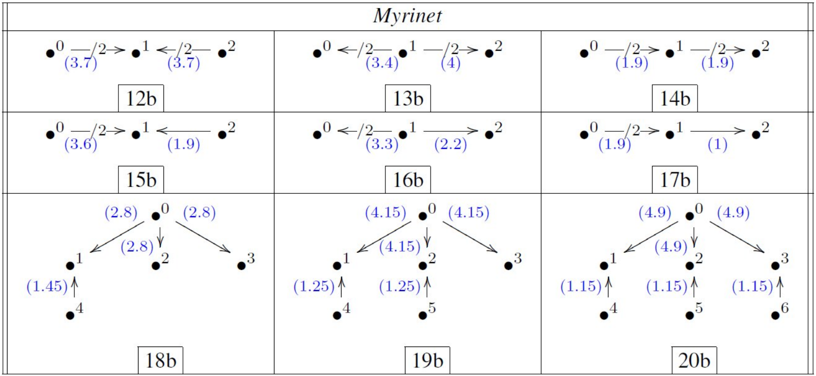
At a given time, a process can be engaged in at most one emission and one reception. This constraint is generally written as two constraints: one on the sum of incomming communication times and one on the sum of outgoing communication times.



1-port (full duplex)

Single-port (Full-Duplex)

This model somehow makes sense when using networks like Myrinet that have few multiplexing units and with protocols without control flow [Mar07].

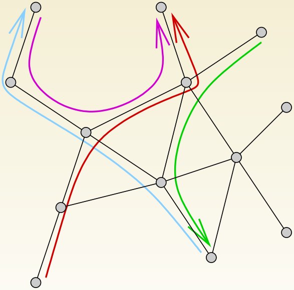


Even if it does not model well complex situations, such a model is not harmfull.

Fluid Modeling

When using TCP-based networks, it is generally reasonnable to use flows to model bandwidth sharing [MR99, Low03].

$$\forall l \in \mathcal{L}, \sum_{r \in \mathcal{R} \text{ s.t. } l \in r} \rho_r \leqslant c_l$$



Income Maximization maximize $\sum_{r \in \mathcal{R}} \rho_r$

Max-Min Fairness maximize $\min_{r \in \mathcal{R}} \rho_r$

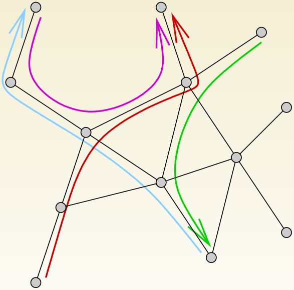
Proportional Fairness maximize $\sum_{r \in \mathcal{R}} \log(\rho_r)$

Potential Delay Minimization minimize $\sum_{r \in \mathcal{R}} \frac{1}{\rho_r}$

Some weird function minimize $\sum_{r \in \mathcal{R}} \arctan(\rho_r)$

When using TCP-based networks, it is generally reasonnable to use flows to model bandwidth sharing [MR99, Low03].

$$\forall l \in \mathcal{L}, \sum_{r \in \mathcal{R} \text{ s.t. } l \in r} \rho_r \leqslant c_l$$



Income Maximization maximize $\sum_{r \in \mathcal{R}} \rho_r$

Max-Min Fairness maximize $\min_{r \in \mathcal{R}} \rho_r$ ATM

Proportional Fairness maximize $\sum_{r \in \mathcal{R}} \log(\rho_r)$

TCP Vegas

Potential Delay Minimization minimize $\sum_{r \in \mathcal{R}} \frac{1}{\rho_r}$

Some weird function minimize $\sum_{r \in \mathcal{R}} \arctan(\rho_r)$

TCP Reno

Flows Extensions

- Note that this model is a multi-port model with capacity-constraints (like in the previous bounded multi-port).
- When latencies are large, using multiple connections enables to get more bandwidth. As a matter of fact, there is very few to loose in using multiple connections...
- Therefore many people enforce a sometimes artificial (but less intrusive) bound on the maximum number of connections per link [Wag05, MYCR06].

Outline

1. Topology
2. Point to Point Communication Models
3. Modeling Concurrency
4. Remind This is a Model, Hence Imperfect

Remind This is a Model, Hence Imperfect

- The previous sharing models are nice but you generally do not know other flows...
- Communications use the memory bus and hence interfere with computations. Taking such interferences into account may become more and more important with multi-core architectures.
- Interference between communications are sometimes...surprising.

Modeling is an art. You have to know your platform and your application to know what is negligeable and what is important. Even if your model is imperfect, you may still derive interesting results.

Part II

Scheduling Case Study

Outline

5. Scheduling Divisible Workload
   - Star-like Network Under the Multi-port Model
   - Bus-like Network
   - Star-like Network Under the One-Port Model
   - Multi-round algorithms

6. Iterative Algorithms

7. Data Redistribution

Context: Distributed Heterogeneous Platforms

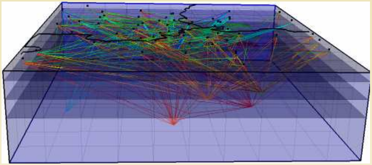
Scheduling divisible load on various architectures [Rob, BGMR96, Bea05, Yan07].

Sources of problems

- Point to point communication model (homogeneous/heterogeneous, with or without latency,. . .)
- Concurrency impact.

Seismic Tomography of the Earth

- Model of the inner structure of the Earth



- The model is validated by comparing the propagation time of a seismic wave in the model to the actual propagation time.
- Set of all seismic events of the year 1999: 817101
- Original program written for a parallel computer:

```
if (rank = ROOT)
   raydata ← read n lines from data file;
MPI_Scatter(raydata, n/P, ..., rbuff, ...,
            ROOT, MPI_COMM_WORLD);
compute_work(rbuff);
```

Applications Covered by The Divisible Loads Model

Applications made of a very (very) large number of fine grain computations.

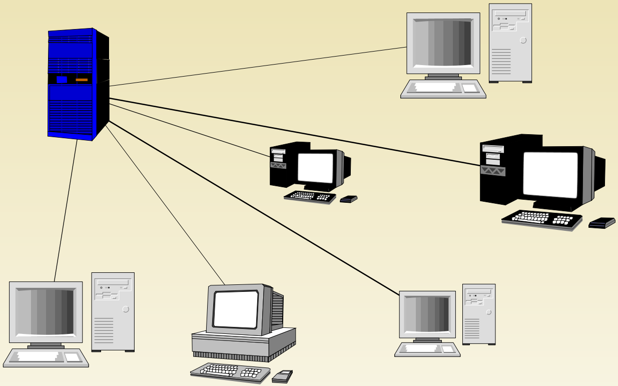
Computation time proportional to the size of the data to be processed.

Independent computations: neither synchronizations nor communications.

Outline

5. Scheduling Divisible Workload
   - Star-like Network Under the Multi-port Model
   - Bus-like Network
   - Star-like Network Under the One-Port Model
   - Multi-round algorithms
6. Iterative Algorithms
7. Data Redistribution

Star-like Network



- The links between the master and the workers have different characteristics.
- The workers have different computational power.
- Communications from the master to the workers can be done in parallel.

Notations

- A set P_1, ..., P_p of processors
- P_1 is the master processor: initially, it holds all the data.
- The overall amount of work: W_{total}.
- Processor P_i receives an amount of work $\alpha_i W_{\text{total}}$ with $\alpha_i \in \mathbb{Q}$ and $\sum_i \alpha_i = 1$.
  Length of a unit-size work on processor P_i: w_i.
  Computation time on P_i: $\alpha_i W_{\text{total}} w_i$.
- Time needed to send a unit-message from P_1 to P_i: c_i.
  Communication time on P_i: $\alpha_i W_{\text{total}} c_i$.
  Multi-port model: P_1 can send messages in parallel to all workers.

"Optimization" Problem

If all communications start in parallel at time 0, the completion time T_i of processor P_i is equal to:

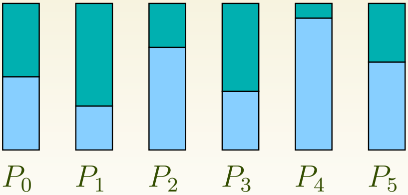
$$T_i = \alpha_i W_{\text{total}} c_i + \alpha_i W_{\text{total}} w_i$$

The makespan T of a load distribution is thus equal to:

$$\max_i \alpha_i W_{\text{total}} (c_i + w_i) = T$$

Therefore this problem is really trivial as we just need to note that $\alpha_i = T/(W_{\text{total}}(c_i + w_i))$ and $\sum_i \alpha_j = 1$ to get T.
Hence, we minimize the makespan by setting:

$$\alpha_i = \frac{1}{\sum_j \frac{c_i + w_i}{c_j + w_j}}$$

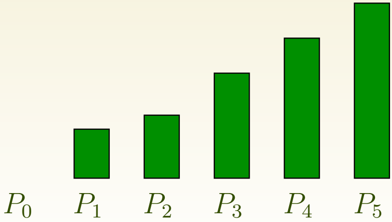


Latencies: just for fun

Let's assume that the time needed to send a message of size α_i from P_1 to P_i is now equal to:

$$L_i + c_i \times \alpha_i$$

Therefore in the optimal solution: forall i such that $\alpha_i > 0$, $L_i + \alpha_i W_{\text{total}} \times (c_i + w_i) = T$.
So just sort the processor by increasing latency and "fill" the W_{total} units of fluid load (the "density" of one unit of load on P_i being equal to $c_i + w_i$).

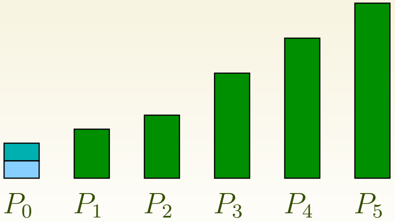


Latencies: just for fun

Let's assume that the time needed to send a message of size α_i from P_1 to P_i is now equal to:

$$L_i + c_i \times \alpha_i$$

Therefore in the optimal solution: forall i such that $\alpha_i > 0$, $L_i + \alpha_i W_{\text{total}} \times (c_i + w_i) = T$.
So just sort the processor by increasing latency and "fill" the W_{total} units of fluid load (the "density" of one unit of load on P_i being equal to $c_i + w_i$).

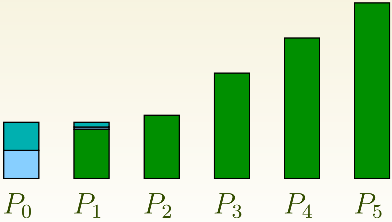


Latencies: just for fun

Let's assume that the time needed to send a message of size α_i from P_1 to P_i is now equal to:

$$L_i + c_i \times \alpha_i$$

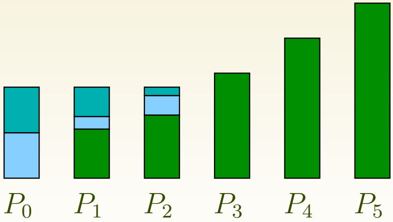
Therefore in the optimal solution: forall i such that $\alpha_i > 0$, $L_i + \alpha_i W_{\text{total}} \times (c_i + w_i) = T$.
So just sort the processor by increasing latency and "fill" the W_{total} units of fluid load (the "density" of one unit of load on P_i being equal to $c_i + w_i$).



Let's assume that the time needed to send a message of size α_i from P_1 to P_i is now equal to:

$$L_i + c_i \times \alpha_i$$

Therefore in the optimal solution: forall i such that $\alpha_i > 0$, $L_i + \alpha_i W_{\text{total}} \times (c_i + w_i) = T$.
So just sort the processor by increasing latency and "fill" the W_{total} units of fluid load (the "density" of one unit of load on P_i being equal to $c_i + w_i$).

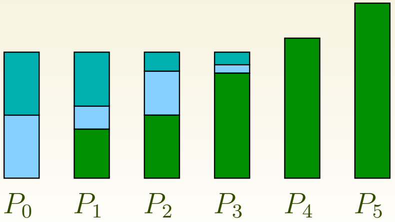


Latencies: just for fun

Let's assume that the time needed to send a message of size α_i from P_1 to P_i is now equal to:

$$L_i + c_i \times \alpha_i$$

Therefore in the optimal solution: forall i such that $\alpha_i > 0$, $L_i + \alpha_i W_{\text{total}} \times (c_i + w_i) = T$.
So just sort the processor by increasing latency and "fill" the W_{total} units of fluid load (the "density" of one unit of load on P_i being equal to $c_i + w_i$).



Outline

5 Scheduling Divisible Workload
- Star-like Network Under the Multi-port Model
- Bus-like Network
- Star-like Network Under the One-Port Model
- Multi-round algorithms

6 Iterative Algorithms

7 Data Redistribution

Notations

- A set P_1, ..., P_p of processors
- P_1 is the master processor: initially, it holds all the data.
- The overall amount of work: W_{total}.
- **Processor P_i receives an amount of work $\alpha_i W_{\text{total}}$ with $\alpha_i W_{\text{total}} \in \mathbb{Q}$ and $\sum_i \alpha_i = 1$.**
  Length of a unit-size work on processor P_i: w_i.
  Computation time on P_i: $n_i w_i$.
- Time needed to send a unit-message from P_1 to P_i: c.
  One-port model: P_1 sends a single message at a time, all processors communicate at the same speed with the master.

For processor P_i (with $c_1 = 0$ and $c_j = c$ otherwise):

$$T_i = \sum_{j=1}^{i} \alpha_j W_{\mathsf{total}}.c_j + \alpha_i W_{\mathsf{total}}.w_i$$

$$T = \max_{1 \leqslant i \leqslant p} \left(\sum_{j=1}^{i} \alpha_j W_{\mathsf{total}}.c_j + \alpha_i W_{\mathsf{total}}.w_i \right)$$

We look for a data distribution α_1, ..., α_p which minimizes T.

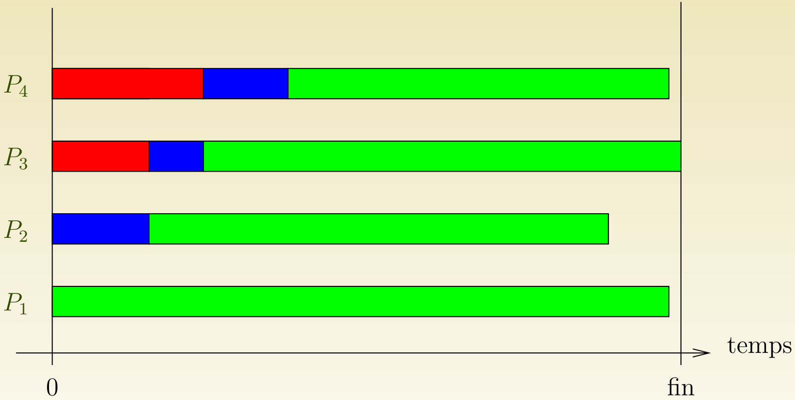
Properties of Load-Balancing

Lemma 1.

In an optimal solution, all processors end their processing at the same time.

Demonstration of Lemma 1

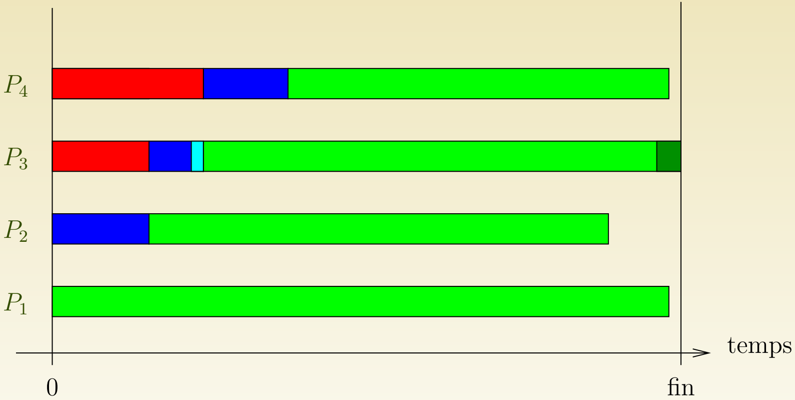
Two workers i and $i+1$ with $T_i < T_{i+1}$.



We decrease α_{i+1} by ε.

Demonstration of Lemma 1

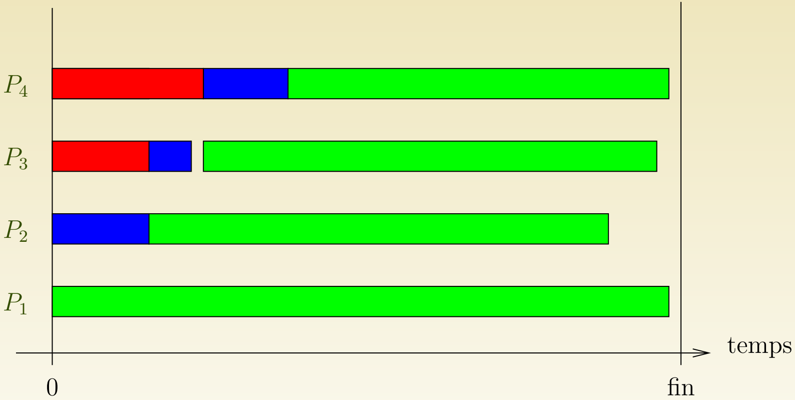
Two workers i and $i+1$ with $T_i < T_{i+1}$.



We decrease α_{i+1} by ε.

Demonstration of Lemma 1

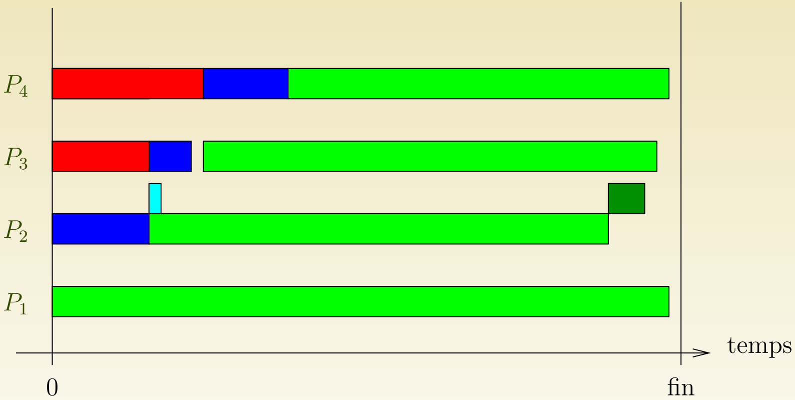
Two workers i and $i+1$ with $T_i < T_{i+1}$.



We decrease α_{i+1} by ε.

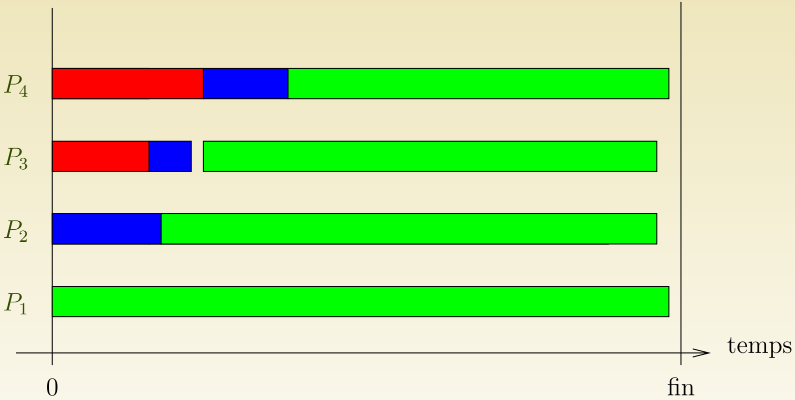
Demonstration of Lemma 1

Two workers i and $i+1$ with $T_i < T_{i+1}$.



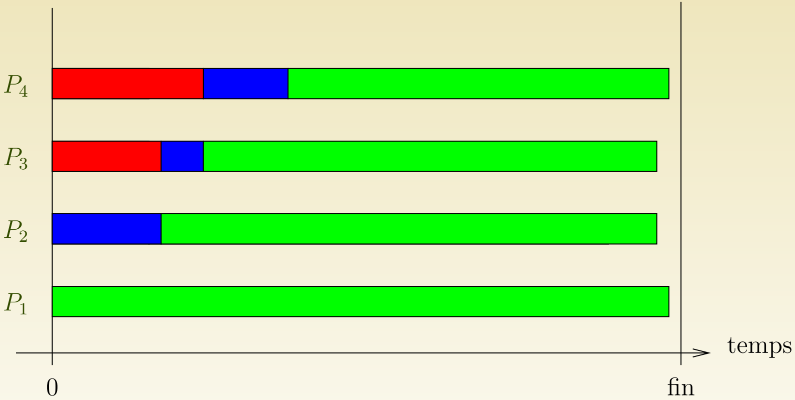
We increase α_i by ε.

Two workers i and $i+1$ with $T_i < T_{i+1}$.



We increase α_i by ε.

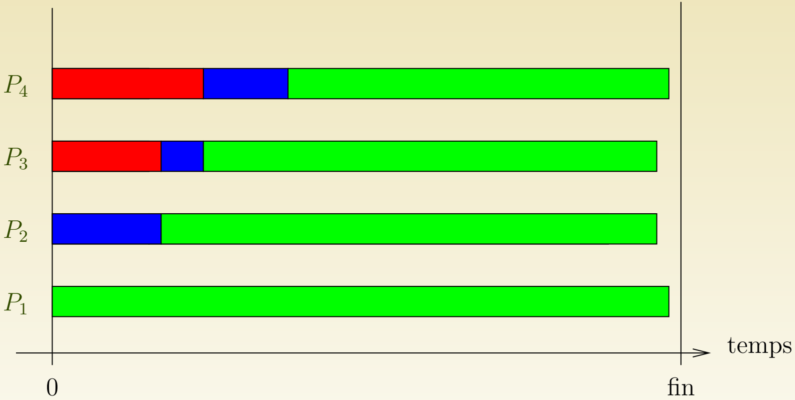
Two workers i and $i + 1$ with $T_i < T_{i+1}$.



The communication time for the following processors is unchanged.

Demonstration of Lemma 1

Two workers i and $i+1$ with $T_i < T_{i+1}$.



We end up with a better solution !

Property for the Resource Selection

Lemma 2.

In an optimal solution all processors work.

Lemma 2.

In an optimal solution all processors work.

Demonstration: this is just a corollary of lemma 1...

Resolution

$T = \alpha_1 W_{\mathsf{total}} w_1.$

Resolution

$T = \alpha_1 W_{\text{total}} w_1$.

$T = \alpha_2 (c + w_2) W_{\text{total}}$. Therefore $\alpha_2 = \frac{w_1}{c+w_2}\alpha_1$.

Resolution

$T = \alpha_1 W_{\text{total}} w_1$.

$T = \alpha_2 (c + w_2) W_{\text{total}}$. Therefore $\alpha_2 = \frac{w_1}{c+w_2} \alpha_1$.

$T = (\alpha_2 c + \alpha_3 (c + w_3)) W_{\text{total}}$. Therefore $\alpha_3 = \frac{w_2}{c+w_3} \alpha_2$.

Resolution

$T = \alpha_1 W_{\text{total}} w_1$.

$T = \alpha_2 (c + w_2) W_{\text{total}}$. Therefore $\alpha_2 = \frac{w_1}{c+w_2} \alpha_1$.

$T = (\alpha_2 c + \alpha_3 (c + w_3)) W_{\text{total}}$. Therefore $\alpha_3 = \frac{w_2}{c+w_3} \alpha_2$.

$\alpha_i = \frac{w_{i-1}}{c+w_i} \alpha_{i-1}$ for $i \geqslant 2$.

Resolution

$T = \alpha_1 W_{\text{total}} w_1$.

$T = \alpha_2(c + w_2)W_{\text{total}}$. Therefore $\alpha_2 = \frac{w_1}{c+w_2}\alpha_1$.

$T = (\alpha_2 c + \alpha_3(c + w_3))W_{\text{total}}$. Therefore $\alpha_3 = \frac{w_2}{c+w_3}\alpha_2$.

$\alpha_i = \frac{w_{i-1}}{c+w_i}\alpha_{i-1}$ for $i \geqslant 2$.

$\sum_{i=1}^{n} \alpha_i = 1$.

Resolution

$T = \alpha_1 W_{\text{total}} w_1$.

$T = \alpha_2(c + w_2)W_{\text{total}}$. Therefore $\alpha_2 = \frac{w_1}{c+w_2}\alpha_1$.

$T = (\alpha_2 c + \alpha_3(c + w_3))W_{\text{total}}$. Therefore $\alpha_3 = \frac{w_2}{c+w_3}\alpha_2$.

$\alpha_i = \frac{w_{i-1}}{c+w_i}\alpha_{i-1}$ for $i \geqslant 2$.

$\sum_{i=1}^{n} \alpha_i = 1$.

$$\alpha_1 \left(1 + \frac{w_1}{c + w_2} + \ldots + \prod_{k=2}^{j} \frac{w_{k-1}}{c + w_k} + \ldots \right) = 1$$

How important is the influence of the ordering of the processor on the solution ?

?

No Impact of The Order of the Communications

Volume processed by processors P_i and P_{i+1} during a time T.

Processor P_i**:** $\alpha_i(c + w_i)W_{\text{total}} = T$. Therefore $\alpha_i = \frac{1}{c+w_i}\frac{T}{W_{\text{total}}}$.

Processor P_{i+1}**:** $\alpha_i c W_{\text{total}} + \alpha_{i+1}(c + w_{i+1})W_{\text{total}} = T$.
Thus $\alpha_{i+1} = \frac{1}{c+w_{i+1}}(\frac{T}{W_{\text{total}}} - \alpha_i c) = \frac{w_i}{(c+w_i)(c+w_{i+1})}\frac{T}{W_{\text{total}}}$.

Processors P_i **and** P_{i+1}**:**

$$\alpha_i + \alpha_{i+1} = \frac{c + w_i + w_{i+1}}{(c + w_i)(c + w_{i+1})}$$

Choice of the Master Processor

We compare processors P_1 and P_2.

Choice of the Master Processor

We compare processors P_1 and P_2.

Processor P_1: $\alpha_1 w_1 W_{\text{total}} = T$. Then, $\alpha_1 = \frac{1}{w_1} \frac{T}{W_{\text{total}}}$.

We compare processors P_1 and P_2.

Processor P_1: $\alpha_1 w_1 W_{\text{total}} = T$. Then, $\alpha_1 = \frac{1}{w_1} \frac{T}{W_{\text{total}}}$.

Processor P_2: $\alpha_2 (c + w_2) W_{\text{total}} = T$. Thus, $\alpha_2 = \frac{1}{c+w_2} \frac{T}{W_{\text{total}}}$.

We compare processors P_1 and P_2.

Processor P_1**:** $\alpha_1 w_1 W_{\text{total}} = T$. Then, $\alpha_1 = \frac{1}{w_1}\frac{T}{W_{\text{total}}}$.

Processor P_2**:** $\alpha_2 (c + w_2) W_{\text{total}} = T$. Thus, $\alpha_2 = \frac{1}{c+w_2}\frac{T}{W_{\text{total}}}$.

Total volume processed:

$$\alpha_1 + \alpha_2 = \frac{c + w_1 + w_2}{w_1(c + w_2)} = \frac{c + w_1 + w_2}{cw_1 + w_1 w_2}$$

Choice of the Master Processor

We compare processors P_1 and P_2.

Processor P_1**:** $\alpha_1 w_1 W_{\text{total}} = T$. Then, $\alpha_1 = \frac{1}{w_1}\frac{T}{W_{\text{total}}}$.

Processor P_2**:** $\alpha_2(c + w_2) W_{\text{total}} = T$. Thus, $\alpha_2 = \frac{1}{c+w_2}\frac{T}{W_{\text{total}}}$.

Total volume processed:

$$\alpha_1 + \alpha_2 = \frac{c + w_1 + w_2}{w_1(c + w_2)} = \frac{c + w_1 + w_2}{cw_1 + w_1 w_2}$$

Minimal when $w_1 < w_2$.
Master $=$ the most powerfull processor (for computations).

Conclusion

- Closed-form expressions for the execution time and the distribution of data.
- Choice of the master.
- The ordering of the processors has no impact.
- All processors take part in the work.

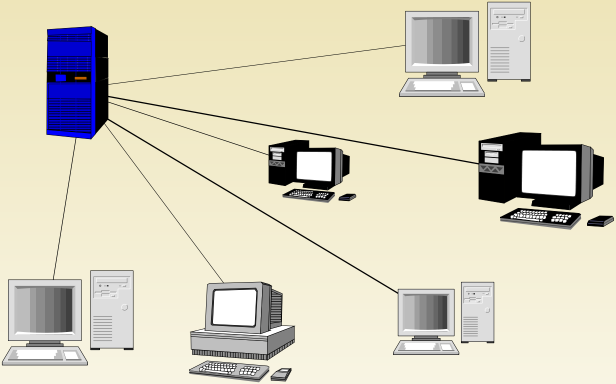
Outline

5 Scheduling Divisible Workload
- Star-like Network Under the Multi-port Model
- Bus-like Network
- Star-like Network Under the One-Port Model
- Multi-round algorithms

6 Iterative Algorithms

7 Data Redistribution

Star-like Network



- The links between the master and the workers have different characteristics.
- The workers have different computational power.

Notations

- A set P_1, ..., P_p of processors
- P_1 is the master processor: initially, it holds all the data.
- The overall amount of work: W_{total}.
- Processor P_i receives an amount of work $\alpha_i W_{\text{total}}$ with $\sum_i n_i = W_{\text{total}}$ with $\alpha_i W_{\text{total}} \in \mathbb{Q}$ and $\sum_i \alpha_i = 1$.
  Length of a unit-size work on processor P_i: w_i.
  Computation time on P_i: $n_i w_i$.
- **Time needed to send a unit-message from P_1 to P_i: c_i.**
  One-port model: P_1 sends a single message at a time.

Goal : maximize the number of processed tasks within a time-bound T_f : $\sum \alpha_i$.

Star Network and Linear Cost Model

Goal : maximize the number of processed tasks within a time-bound T_f : $\sum \alpha_i$.

Lemma 3.

In any optimal solution of the StarLinear problem, all workers participate in the computation, and all processors finish computing simultaneously.

Star Network and Linear Cost Model

Goal : maximize the number of processed tasks within a time-bound T_f : $\sum \alpha_i$.

Lemma 3.

In any optimal solution of the StarLinear problem, all workers participate in the computation, and all processors finish computing simultaneously.

Lemma 4.

An optimal ordering for the StarLinear problem is obtained by serving the workers in the ordering of non decreasing link capacities c_i.

Sketch of the Proof of Lemma 3

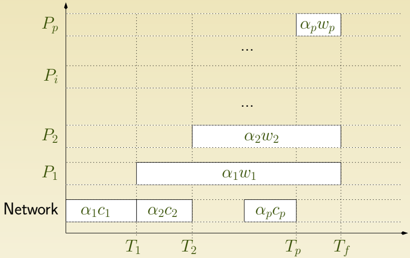
Two steps :

- All workers participate in the computation. . .

Sketch of the Proof of Lemma 3

Two steps :

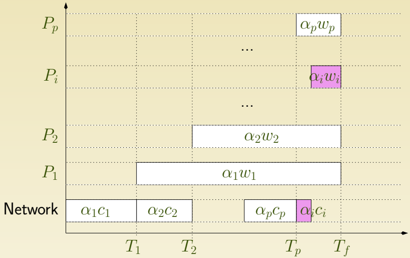
- All workers participate in the computation... otherwise it would not be optimal.



Sketch of the Proof of Lemma 3

Two steps :

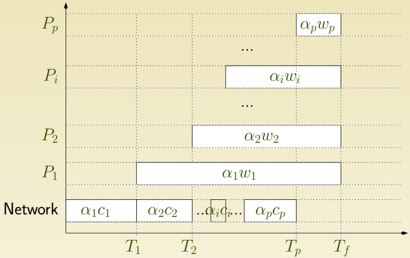
- All workers participate in the computation. . . otherwise it would not be optimal.
- All processors finish their work at the same time.



Sketch of the Proof of Lemma 3

Two steps :

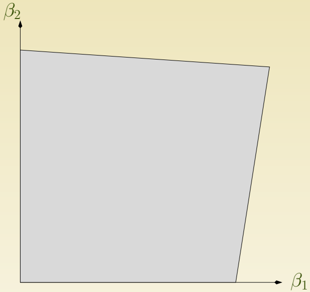
- All workers participate in the computation. . . otherwise it would not be optimal.
- All processors finish their work at the same time.



$$
\begin{array}{l}
\textsc{Maximize } \sum \beta_i, \\
\textsc{subject to} \\
\left\{\begin{array}{lll}
\text{LB}(i) & \forall i, & \beta_i \geqslant 0 \\
\text{UB}(i) & \forall i, & \sum_{k=1}^{i} \beta_k c_k + \beta_i w_i \leqslant T_f
\end{array}\right.
\end{array}
$$

Two steps :

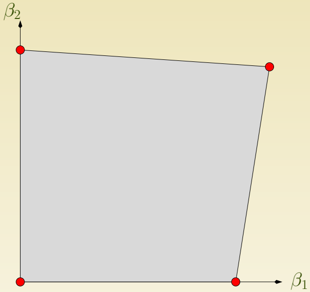
- All workers participate in the computation. . . otherwise it would not be optimal.
- All processors finish their work at the same time.



$$\begin{array}{l}\textsc{Maximize } \sum \beta_i, \\ \textsc{subject to} \\ \left\{\begin{array}{lll} \mathsf{LB}(i) & \forall i, & \beta_i \geqslant 0 \\ \mathsf{UB}(i) & \forall i, & \sum_{k=1}^{i} \beta_k c_k + \beta_i w_i \leqslant T_f \end{array}\right.\end{array}$$

Two steps :

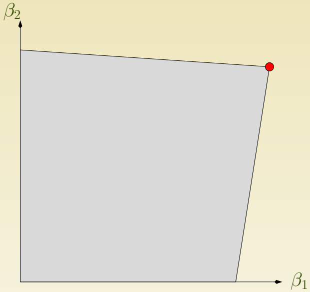
- All workers participate in the computation... otherwise it would not be optimal.
- All processors finish their work at the same time.



$$
\begin{array}{l}
\textsc{Maximize} \sum \beta_i, \\
\textsc{subject to} \\
\begin{cases}
\mathsf{LB}(i) & \forall i, \quad \beta_i \geqslant 0 \\
\mathsf{UB}(i) & \forall i, \quad \sum_{k=1}^{i} \beta_k c_k + \beta_i w_i \leqslant T_f
\end{cases}
\end{array}
$$

Two steps :

- All workers participate in the computation. . . otherwise it would not be optimal.
- All processors finish their work at the same time.

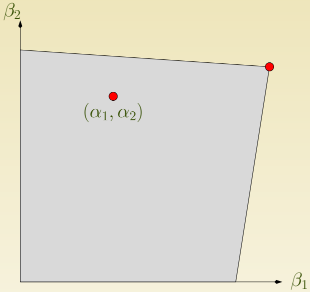


$$\textsc{Maximize} \sum \beta_i,$$
$$\textsc{subject to}$$
$$\begin{cases} \mathsf{LB}(i) & \forall i, \quad \beta_i \geqslant 0 \\ \mathsf{UB}(i) & \forall i, \quad \sum_{k=1}^{i} \beta_k c_k + \beta_i w_i \leqslant T_f \end{cases}$$

Sketch of the Proof of Lemma 3

Two steps :

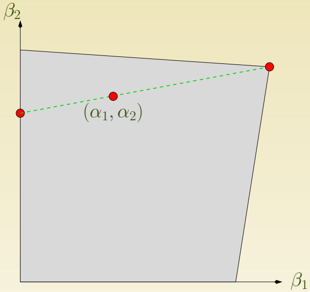
- All workers participate in the computation... otherwise it would not be optimal.
- All processors finish their work at the same time.



$$
\begin{array}{l}
\textsc{Maximize } \sum \beta_i, \\
\textsc{subject to} \\
\left\{ \begin{array}{lll}
\text{LB}(i) & \forall i, & \beta_i \geqslant 0 \\
\text{UB}(i) & \forall i, & \sum_{k=1}^{i} \beta_k c_k + \beta_i w_i \leqslant T_f
\end{array} \right.
\end{array}
$$

Two steps :

- All workers participate in the computation…otherwise it would not be optimal.
- All processors finish their work at the same time.



$$\textsc{Maximize} \sum \beta_i,$$

SUBJECT TO

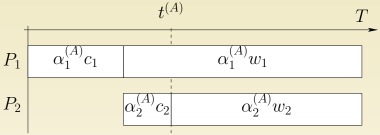
$$\begin{cases} \mathsf{LB}(i) & \forall i, \quad \beta_i \geqslant 0 \\ \mathsf{UB}(i) & \forall i, \quad \sum_{k=1}^{i} \beta_k c_k + \beta_i w_i \leqslant T_f \end{cases}$$

Sketch of the Proof of Lemma 4

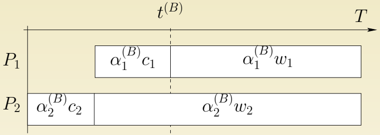
The proof is based on the comparison of the amount of work that is performed by the first two workers, and then proceeds by induction.

Sketch of the Proof of Lemma 4

The proof is based on the comparison of the amount of work that is performed by the first two workers, and then proceeds by induction.

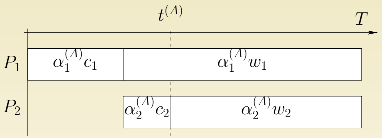


(A) P_1 starts before P_2

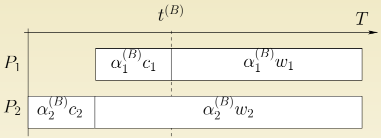


(B) P_2 starts before P_1

The proof is based on the comparison of the amount of work that is performed by the first two workers, and then proceeds by induction.



(A) P_1 starts before P_2

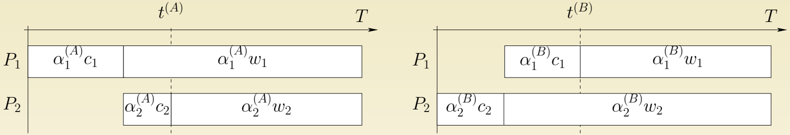


(B) P_2 starts before P_1

$$t^{(A)} = t^{(B)}, \tag{1}$$

Sketch of the Proof of Lemma 4

The proof is based on the comparison of the amount of work that is performed by the first two workers, and then proceeds by induction.



(A) P_1 starts before P_2 (B) P_2 starts before P_1

$$t^{(A)} = t^{(B)}, \tag{1}$$

and

$$(\alpha_1^{(A)} + \alpha_2^{(A)}) - (\alpha_1^{(B)} + \alpha_2^{(B)}) = \frac{T(c_2 - c_1)}{(c_1 + w_1)(c_2 + w_2)}. \tag{2}$$

Conclusion

- The processors must be ordered by decreasing bandwidths
- All processors are working
- All processors end their work at the same time
- Formulas for the execution time and the distribution of data

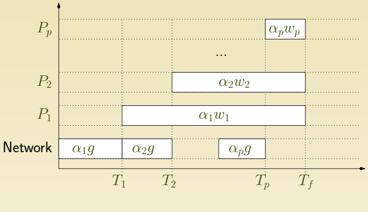
Outline

5 Scheduling Divisible Workload
- Star-like Network Under the Multi-port Model
- Bus-like Network
- Star-like Network Under the One-Port Model
- Multi-round algorithms

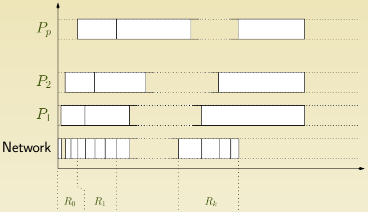
6 Iterative Algorithms

7 Data Redistribution

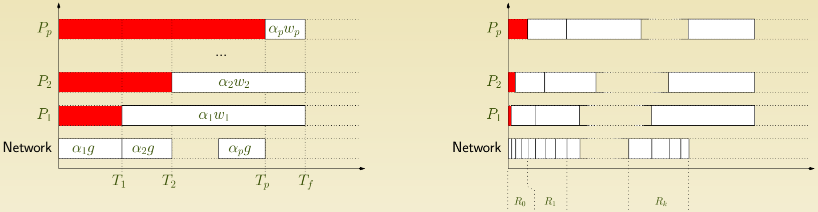
One-round vs. Multi-round



One round



Multi-round

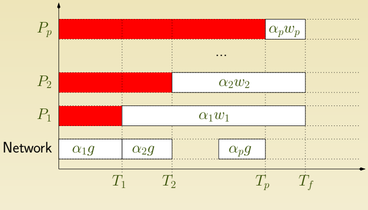


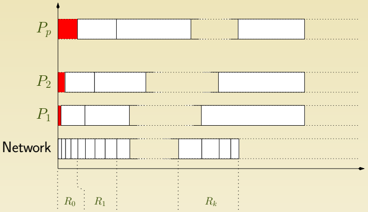
One round	Multi-round
$\rightsquigarrow$ long idle-times	Efficient when W_{total} large

Intuition: start with small rounds, then increase chunks.

Problems:

One-round vs. Multi-round





One round
$\rightsquigarrow$ long idle-times

Multi-round
Efficient when W_{total} large

Intuition: start with small rounds, then increase chunks.
Problems:

- linear communication model leads to absurd solution
- resource selection
- number of rounds
- size of each round

Notations

- A set P_1, ..., P_p of processors
- P_1 is the master processor: initially, it holds all the data.
- The overall amount of work: W_{total}.
- Processor P_i receives an amount of work $\alpha_i W_{\text{total}}$ with $\sum_i n_i = W_{\text{total}}$ with $\alpha_i W_{\text{total}} \in \mathbb{Q}$ and $\sum_i \alpha_i = 1$.
  Length of a unit-size work on processor P_i: w_i.
  Computation time on P_i: $n_i w_i$.
- **Time needed to send a message of size α_i P_1 to P_i: $L_i + c_i \times \alpha_i$.**
  One-port model: P_1 sends and receives a single message at a time.

Complexity

Definition: One round, $\forall i, c_i = 0$.

Given W_{total}, p workers, $(P_i)_{1 \leqslant i \leqslant p}$, $(L_i)_{1 \leqslant i \leqslant p}$, and a rational number $T \geqslant 0$, and assuming that bandwidths are infinite, is it possible to compute all W_{total} load units within T time units?

Theorem 1.

The problem with one-round and infinite bandwidths is NP-complete.

Definition: One round, $\forall i, c_i = 0$.

Given W_{total}, p workers, $(P_i)_{1 \leqslant i \leqslant p}$, $(L_i)_{1 \leqslant i \leqslant p}$, and a rational number $T \geqslant 0$, and assuming that bandwidths are infinite, is it possible to compute all W_{total} load units within T time units?

Theorem 1.

The problem with one-round and infinite bandwidths is NP-complete.

What is the complexity of the general problem with finite bandwidths and several rounds ?

Complexity

Definition: One round, $\forall i, c_i = 0$.

Given W_{total}, p workers, $(P_i)_{1 \leqslant i \leqslant p}$, $(L_i)_{1 \leqslant i \leqslant p}$, and a rational number $T \geqslant 0$, and assuming that bandwidths are infinite, is it possible to compute all W_{total} load units within T time units?

Theorem 1.

The problem with one-round and infinite bandwidths is NP-complete.

What is the complexity of the general problem with finite bandwidths and several rounds ?

The general problem is NP-hard, but does not appear to be in NP (no polynomial bound on the number of activations).

Fixed activation sequence

Hypotheses

1. Number of activations : N_{act};
2. Whether P_i is **the** processor used during activation j : $\chi_i^{(j)}$

$$
\begin{aligned}
&\textsc{Minimize } T,\\
&\textsc{under the constraints}\\
&\begin{cases}
\displaystyle\sum_{j=1}^{N_{\text{act}}}\sum_{i=1}^{p}\chi_i^{(j)}\alpha_i^{(j)} = W_{\text{total}}\\
\displaystyle\forall k \leqslant N_{\text{act}}, \forall l : \left(\sum_{j=1}^{k}\sum_{i=1}^{p}\chi_i^{(j)}(L_i+\alpha_i^{(j)}c_i)\right) + \sum_{j=k}^{N_{\text{act}}}\chi_l^{(j)}\alpha_l^{(j)}w_l \leqslant T\\
\forall i,j : \alpha_i^{(j)} \geqslant 0
\end{cases}
\end{aligned}
\tag{3}
$$

Can be solved in polynomial time.

Fixed number of activations

MINIMIZE T,

UNDER THE CONSTRAINTS

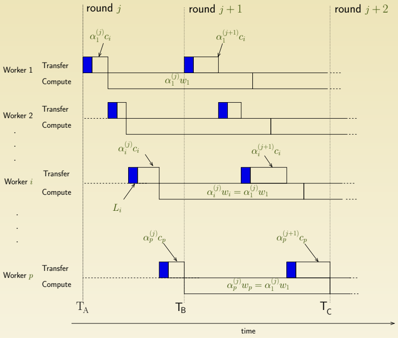
$$
\begin{cases}
\displaystyle\sum_{j=1}^{N_{\text{act}}}\sum_{i=1}^{p} \chi_i^{(j)} \alpha_i^{(j)} = W_{\text{total}} \\
\displaystyle\forall k \leqslant N_{\text{act}}, \forall l : \left(\sum_{j=1}^{k}\sum_{i=1}^{p} \chi_i^{(j)} (L_i + \alpha_i^{(j)} c_i) \right) + \sum_{j=k}^{N_{\text{act}}} \chi_l^{(j)} \alpha_l^{(j)} w_l \leqslant T \\
\displaystyle\forall k \leqslant N_{\text{act}} : \sum_{i=1}^{p} \chi_i^{(k)} \leqslant 1 \\
\forall i,j : \chi_i^{(j)} \in \{0,1\} \\
\forall i,j : \alpha_i^{(j)} \geqslant 0
\end{cases}
\tag{4}
$$

Exact but exponential (branch-and-bound algorithms).

In a round: all workers have same computation time

Geometrical increase of rounds size

No idle time in communications:



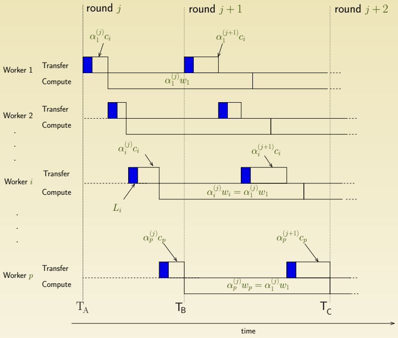
$$\alpha_i^{(j)} w_i = \sum_{k=1}^{p} (L_k + \alpha_k^{(j+1)} c_k).$$

Heuristic processor selection: by decreasing bandwidths

In a round: all workers have same computation time

Geometrical increase of rounds size

No idle time in communications:

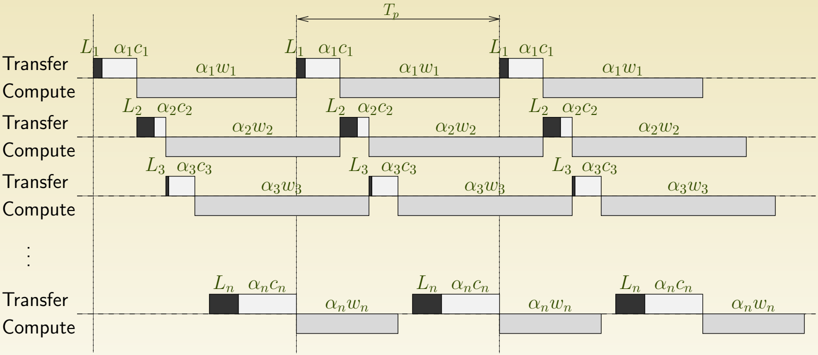


$$\alpha_i^{(j)} w_i = \sum_{k=1}^{p} (L_k + \alpha_k^{(j+1)} c_k).$$

Heuristic processor selection: by decreasing bandwidths

No guarantee...

Periodic Schedule



How to choose T_p? Which resources to select?

With no Overlap (1/4)

Equations

- Divide total execution time T into k periods of duration T_p.
- $\mathcal{I} \subset \{1, \ldots, p\}$ participating processors.
- Bandwidth limitation:

$$\sum_{i \in \mathcal{I}} (L_i + \alpha_i c_i) \leqslant T_p.$$

- No overlap:

$$\forall i \in \mathcal{I}, \qquad L_i + \alpha_i (c_i + w_i) \leqslant T_p.$$

Normalization

- β_i average number of tasks processed by P_i during one time unit.

Normalization

- β_i average number of tasks processed by P_i during one time unit.
- Linear program:
$$\textsc{Maximize} \sum_{i=1}^{p} \beta_i$$
$$\begin{cases} \forall i \in \mathcal{I}, \quad \beta_i(c_i + w_i) \leqslant 1 - \frac{L_i}{T_p} \\ \sum_{i \in \mathcal{I}} \beta_i c_i \leqslant 1 - \frac{\sum_{i \in \mathcal{I}} L_i}{T_p} \end{cases} .$$

With no Overlap (2/4)

Normalization

- β_i average number of tasks processed by P_i during one time unit.
- Linear program: $$\left\{\begin{array}{l}\textsc{Maximize} \sum_{i=1}^{p} \beta_i \\ \forall i \in \mathcal{I}, \quad \beta_i(c_i + w_i) \leqslant 1 - \frac{L_i}{T_p} \\ \sum_{i \in \mathcal{I}} \beta_i c_i \leqslant 1 - \frac{\sum_{i \in \mathcal{I}} L_i}{T_p}\end{array}\right. .$$

Relaxed version

$$\left\{\begin{array}{l}\textsc{Maximize} \sum_{i=1}^{p} x_i \\ \forall 1 \leqslant i \leqslant p, \quad x_i(c_i + w_i) \leqslant 1 - \frac{L_i}{T_p} \\ \sum_{i=1}^{p} x_i c_i \leqslant 1 - \frac{\sum_{i=1}^{p} L_i}{T_p}\end{array}\right.$$

With no Overlap (2/4)

Normalization

- β_i average number of tasks processed by P_i during one time unit.
- Linear program: $$\begin{array}{l}\textsc{Maximize} \sum_{i=1}^{p} \beta_i \\ \left\{\begin{array}{ll} \forall i \in \mathcal{I}, & \beta_i(c_i + w_i) \leqslant 1 - \frac{L_i}{T_p} \\ \sum_{i \in \mathcal{I}} \beta_i c_i \leqslant 1 - \frac{\sum_{i \in \mathcal{I}} L_i}{T_p} \end{array}\right. \end{array}.$$

Relaxed version

$$\begin{array}{l}\textsc{Maximize} \sum_{i=1}^{p} x_i \\ \left\{\begin{array}{ll} \forall 1 \leqslant i \leqslant p, & x_i(c_i + w_i) \leqslant 1 - \frac{\sum_{i=1}^{p} L_i}{T_p} \\ \sum_{i=1}^{p} x_i c_i \leqslant 1 - \frac{\sum_{i=1}^{p} L_i}{T_p} \end{array}\right. \end{array}$$

Bandwidth-centric solution

- Sort: $c_1 \leqslant c_2 \leqslant \ldots \leqslant c_p$.
- Let q be the largest index so that $\sum_{i=1}^{q} \frac{c_i}{c_i+w_i} \leqslant 1$.
- If $q < p$, $\varepsilon = 1 - \sum_{i=1}^{q} \frac{c_i}{c_i+w_i}$.
- Optimal solution to relaxed program:

$$\forall 1 \leqslant i \leqslant q, \quad x_i = \frac{1 - \frac{\sum_{i=1}^{p} L_i}{T_p}}{c_i + w_i}$$

and (if $q < p$):

$$x_{q+1} = \left(1 - \frac{\sum_{i=1}^{p} L_i}{T_p}\right)\left(\frac{\varepsilon}{c_{q+1}}\right),$$

and $x_{q+2} = x_{q+3} = \ldots = x_p = 0$.

With no Overlap (4/4)

Asymptotic optimality

- Let $T_p = \sqrt{T^*_{\max}}$ and $\alpha_i = x_i T_p$ for all i.
- Then $T \leqslant T^*_{\max} + O(\sqrt{T^*_{\max}})$.
- Closed-form expressions for resource selection and task assignment provided by the algorithm.

With Overlap

Key points

- Still sort resources according to the c_i.
- Greedily select resources until the sum of the ratios $\frac{c_i}{w_i}$ $\left(\text{instead of } \frac{c_i}{c_i+w_i}\right)$ exceeds 1.

Conclusion

- NP-hardness comes from the one-port model and latencies.
- The problem is however rather easy to approximate. Rough solutions are way enough.
- Communications are much more important than computations.

Outline

5 Scheduling Divisible Workload

6 Iterative Algorithms
- The Problem
- Fully Homogeneous Network
- Heterogeneous Network (Complete)
- Heterogeneous Network (General Case)

7 Data Redistribution

Outline

5 Scheduling Divisible Workload

6 Iterative Algorithms
- The Problem
- Fully Homogeneous Network
- Heterogeneous Network (Complete)
- Heterogeneous Network (General Case)

7 Data Redistribution

How to embed a ring in a complex network [LRRV04].

Sources of problems

- Heterogeneity of processors (computational power, memory, etc.)
- Heterogeneity of communications links.
- Irregularity of interconnection network.

Targeted Applications: Iterative Algorithms

- A set of data (typically, a matrix)
- Structure of the algorithms:
  1. Each processor performs a computation on its chunk of data
  2. Each processor exchange the "border" of its chunk of data with its neighbor processors
  3. We go back at Step 1

Targeted Applications: Iterative Algorithms

- A set of data (typically, a matrix)
- Structure of the algorithms:
  1. Each processor performs a computation on its chunk of data
  2. Each processor exchange the "border" of its chunk of data with its neighbor processors
  3. We go back at Step 1

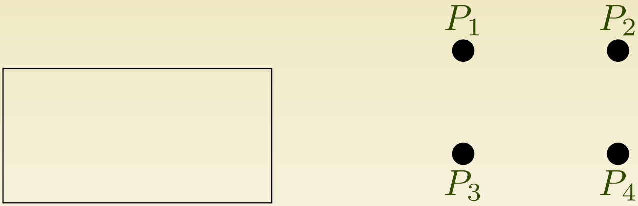
Question: how can we efficiently execute such an algorithm on such a platform?

The Questions

- Which processors should be used ?
- What amount of data should we give them ?
- How do we cut the set of data ?

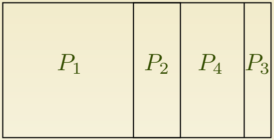
First of All, a Simplification: Slicing the Data

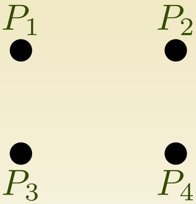
- Data: a 2-D array



First of All, a Simplification: Slicing the Data

- Data: a 2-D array

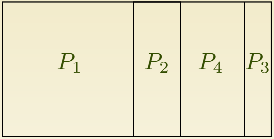


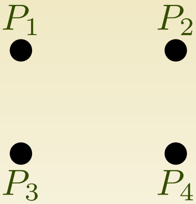


- Unidimensional cutting into vertical slices

First of All, a Simplification: Slicing the Data

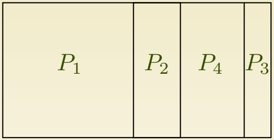
- Data: a 2-D array

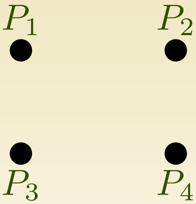




- Unidimensional cutting into vertical slices
- Consequences:

- Data: a 2-D array

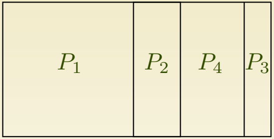


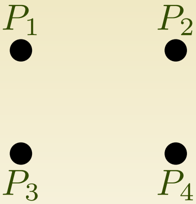


- Unidimensional cutting into vertical slices
- Consequences:
  1. Borders and neighbors are easily defined

First of All, a Simplification: Slicing the Data

- Data: a 2-D array

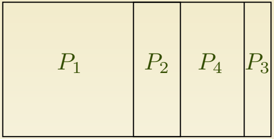


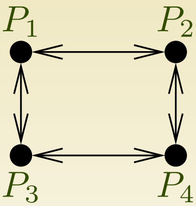


- Unidimensional cutting into vertical slices
- Consequences:
  1. Borders and neighbors are easily defined
  2. Constant volume of data exchanged between neighbors: D_c

First of All, a Simplification: Slicing the Data

- Data: a 2-D array





- Unidimensional cutting into vertical slices
- Consequences:
  1. Borders and neighbors are easily defined
  2. Constant volume of data exchanged between neighbors: D_c
  3. Processors are virtually organized into a ring

Notations

- Processors: P_1, ..., P_p
- Processor P_i executes a unit task in a time w_i
- Overall amount of work D_w;
  Share of P_i: $\alpha_i.D_w$ processed in a time $\alpha_i.D_w.w_i$
  ($\alpha_i \geqslant 0$, $\sum_j \alpha_j = 1$)
- Cost of a unit-size communication from P_i to P_j: $c_{i,j}$
- Cost of a sending from P_i to its successor in the ring: $D_c.c_{i,\mathsf{succ}(i)}$

Communications: 1-Port Model (Full-Duplex)

A processor can:

- send at most one message at any time;
- receive at most one message at any time;
- send and receive a message simultaneously.

Objective

1. Select q processors among p

Objective

1. Select q processors among p
2. Order them into a ring

Objective

1. Select q processors among p
2. Order them into a ring
3. Distribute the data among them

Objective

1. Select q processors among p
2. Order them into a ring
3. Distribute the data among them

So as to minimize:

$$\max_{1 \leqslant i \leqslant p} \mathbb{I}\{i\}[\alpha_i.D_w.w_i + D_c.(c_{i,\mathsf{pred}(i)} + c_{i,\mathsf{succ}(i)})]$$

Where $\mathbb{I}\{i\}[x] = x$ if P_i participates in the computation, and 0 otherwise

Outline

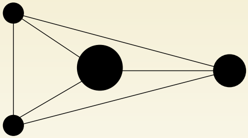
5 Scheduling Divisible Workload

6 Iterative Algorithms
- The Problem
- Fully Homogeneous Network
- Heterogeneous Network (Complete)
- Heterogeneous Network (General Case)

7 Data Redistribution

Special Hypotheses

1. There exists a communication link between any two processors
2. All links have the same capacity
   $(\exists c, \forall i, j\ c_{i,j} = c)$



Consequences

- Either the most powerful processor performs all the work, or all the processors participate

Consequences

- Either the most powerful processor performs all the work, or all the processors participate
- If all processors participate, all end their share of work simultaneously

Consequences

- Either the most powerful processor performs all the work, or all the processors participate
- If all processors participate, all end their share of work simultaneously($\exists \tau, \quad \alpha_i.D_w.w_i = \tau$, so $1 = \sum_i \frac{\tau}{D_w.w_i}$)

Consequences

- Either the most powerful processor performs all the work, or all the processors participate
- If all processors participate, all end their share of work simultaneously($\exists \tau, \quad \alpha_i.D_w.w_i = \tau$, so $1 = \sum_i \frac{\tau}{D_w.w_i}$)
- Time of the optimal solution:

$$T_{\mathsf{step}} = \min \left\{ D_w.w_{\min}, D_w.\frac{1}{\sum_i \frac{1}{w_i}} + 2.D_c.c \right\}$$

Outline

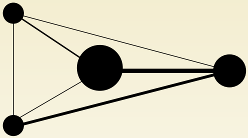
5 Scheduling Divisible Workload

6 Iterative Algorithms
- The Problem
- Fully Homogeneous Network
- Heterogeneous Network (Complete)
- Heterogeneous Network (General Case)

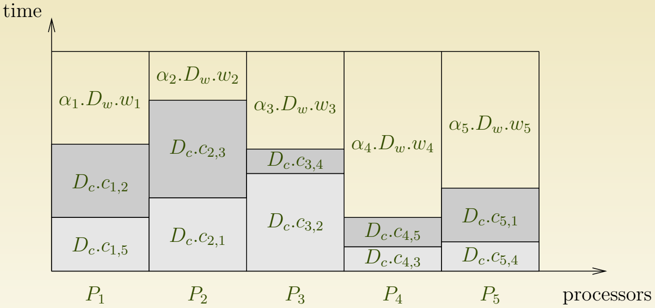
7 Data Redistribution

Special hypothesis

1. There exists a communication link between any two processors



All the Processors Participate: Study (1)



All processors end simultaneously

All the Processors Participate: Study (2)

- All processors end simultaneously

$$T_{\mathsf{step}} = \alpha_i.D_w.w_i + D_c.(c_{i,\mathsf{succ}(i)} + c_{i,\mathsf{pred}(i)})$$

All the Processors Participate: Study (2)

- All processors end simultaneously

$$T_{\text{step}} = \alpha_i.D_w.w_i + D_c.(c_{i,\text{succ}(i)} + c_{i,\text{pred}(i)})$$

- $\displaystyle\sum_{i=1}^{p} \alpha_i = 1 \;\Rightarrow\; \sum_{i=1}^{p} \frac{T_{\text{step}} - D_c.(c_{i,\text{succ}(i)} + c_{i,\text{pred}(i)})}{D_w.w_i} = 1.$ Thus

$$\frac{T_{\text{step}}}{D_w.w_{\text{cumul}}} = 1 + \frac{D_c}{D_w}\sum_{i=1}^{p}\frac{c_{i,\text{succ}(i)} + c_{i,\text{pred}(i)}}{w_i}$$

where $w_{\text{cumul}} = \frac{1}{\sum_i \frac{1}{w_i}}$

All the Processors Participate: Interpretation

$$\frac{T_{\mathsf{step}}}{D_w.w_{\mathsf{cumul}}} = 1 + \frac{D_c}{D_w} \sum_{i=1}^{p} \frac{c_{i,\mathsf{succ}(i)} + c_{i,\mathsf{pred}(i)}}{w_i}$$

All the Processors Participate: Interpretation

$$\frac{T_{\text{step}}}{D_w . w_{\text{cumul}}} = 1 + \frac{D_c}{D_w} \sum_{i=1}^{p} \frac{c_{i,\text{succ}(i)} + c_{i,\text{pred}(i)}}{w_i}$$

T_{step} is minimal when $\displaystyle\sum_{i=1}^{p} \frac{c_{i,\text{succ}(i)} + c_{i,\text{pred}(i)}}{w_i}$ is minimal

All the Processors Participate: Interpretation

$$\frac{T_{\text{step}}}{D_w . w_{\text{cumul}}} = 1 + \frac{D_c}{D_w} \sum_{i=1}^{p} \frac{c_{i,\text{succ}(i)} + c_{i,\text{pred}(i)}}{w_i}$$

T_{step} is minimal when $\displaystyle\sum_{i=1}^{p} \frac{c_{i,\text{succ}(i)} + c_{i,\text{pred}(i)}}{w_i}$ is minimal

Look for an hamiltonian cycle of minimal weight in a graph where the edge from P_i to P_j has a weight of $d_{i,j} = \frac{c_{i,j}}{w_i} + \frac{c_{j,i}}{w_j}$

All the Processors Participate: Interpretation

$$\frac{T_{\textsf{step}}}{D_w . w_{\textsf{cumul}}} = 1 + \frac{D_c}{D_w} \sum_{i=1}^{p} \frac{c_{i,\textsf{succ}(i)} + c_{i,\textsf{pred}(i)}}{w_i}$$

$T_{\textsf{step}}$ is minimal when $\displaystyle\sum_{i=1}^{p} \frac{c_{i,\textsf{succ}(i)} + c_{i,\textsf{pred}(i)}}{w_i}$ is minimal

Look for an hamiltonian cycle of minimal weight in a graph where the edge from P_i to P_j has a weight of $d_{i,j} = \frac{c_{i,j}}{w_i} + \frac{c_{j,i}}{w_j}$

NP-complete problem

All the Processors Participate: Linear Program

$$\textsc{Minimize} \ \sum_{i=1}^{p} \sum_{j=1}^{p} d_{i,j}.x_{i,j},$$

$$\textsc{satisfying the (in)equations}$$

$$\left\{\begin{array}{ll}
(1) \ \sum_{j=1}^{p} x_{i,j} = 1 & 1 \leqslant i \leqslant p \\
(2) \ \sum_{i=1}^{p} x_{i,j} = 1 & 1 \leqslant j \leqslant p \\
(3) \ x_{i,j} \in \{0,1\} & 1 \leqslant i,j \leqslant p \\
(4) \ u_i - u_j + p.x_{i,j} \leqslant p-1 & 2 \leqslant i,j \leqslant p, i \neq j \\
(5) \ u_i \text{ integer}, u_i \geqslant 0 & 2 \leqslant i \leqslant p
\end{array}\right.$$

$x_{i,j} = 1$ if, and only if, the edge from P_i to P_j is used

Best ring made of q processors

MINIMIZE T SATISFYING THE (IN)EQUATIONS

$$
\left\{
\begin{array}{ll}
(1)\ x_{i,j} \in \{0,1\} & 1 \leqslant i,j \leqslant p \\
(2)\ \sum_{i=1}^{p} x_{i,j} \leqslant 1 & 1 \leqslant j \leqslant p \\
(3)\ \sum_{i=1}^{p} \sum_{j=1}^{p} x_{i,j} = q & \\
(4)\ \sum_{i=1}^{p} x_{i,j} = \sum_{i=1}^{p} x_{j,i} & 1 \leqslant j \leqslant p \\
 & \\
(5)\ \sum_{i=1}^{p} \alpha_i = 1 & \\
(6)\ \alpha_i \leqslant \sum_{j=1}^{p} x_{i,j} & 1 \leqslant i \leqslant p \\
(7)\ \alpha_i.w_i + \frac{D_c}{D_w} \sum_{j=1}^{p} (x_{i,j}c_{i,j} + x_{j,i}c_{j,i}) \leqslant T & 1 \leqslant i \leqslant p \\
 & \\
(8)\ \sum_{i=1}^{p} y_i = 1 & \\
(9)\ -p.y_i - p.y_j + u_i - u_j + q.x_{i,j} \leqslant q-1 & 1 \leqslant i,j \leqslant p, i \neq j \\
(10)\ y_i \in \{0,1\} & 1 \leqslant i \leqslant p \\
(11)\ u_i \text{ integer}, u_i \geqslant 0 & 1 \leqslant i \leqslant p
\end{array}
\right.
$$

Linear Programming

- Problems with rational variables: can be solved in polynomial time (in the size of the problem).
- Problems with integer variables: solved in exponential time in the worst case.
- No relaxation in rationals seems possible here. . .

And, in Practice ?

All processors participate. One can use a heuristic to solve the traveling salesman problem (as Lin-Kernighan's one)

1. Exhaustive search: feasible until a dozen of processors. . .
2. Greedy heuristic: initially we take the best pair of processors; for a given ring we try to insert any unused processor in between any pair of neighbor processors in the ring. . .

And, in Practice ?

All processors participate. One can use a heuristic to solve the traveling salesman problem (as Lin-Kernighan's one)
No guarantee, but excellent results in practice.

1. Exhaustive search: feasible until a dozen of processors. . .
2. Greedy heuristic: initially we take the best pair of processors; for a given ring we try to insert any unused processor in between any pair of neighbor processors in the ring. . .

And, in Practice ?

All processors participate. One can use a heuristic to solve the traveling salesman problem (as Lin-Kernighan's one)
No guarantee, but excellent results in practice.

General case.

1. Exhaustive search: feasible until a dozen of processors...
2. Greedy heuristic: initially we take the best pair of processors; for a given ring we try to insert any unused processor in between any pair of neighbor processors in the ring...

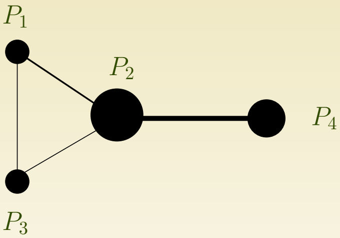
Outline

5. Scheduling Divisible Workload

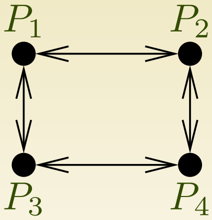
6. Iterative Algorithms
   - The Problem
   - Fully Homogeneous Network
   - Heterogeneous Network (Complete)
   - Heterogeneous Network (General Case)

7. Data Redistribution

New Difficulty: Dommunication Links Sharing

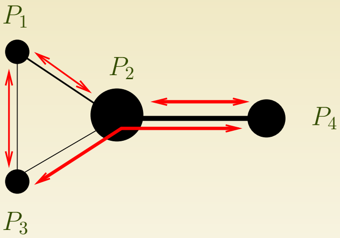


Heterogeneous platform

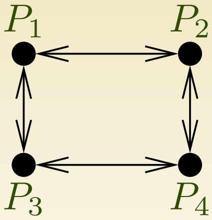


Virtual ring

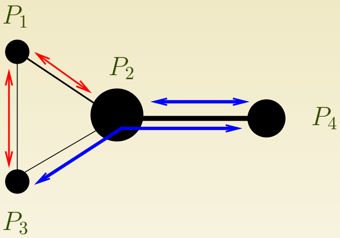
New Difficulty: Dommunication Links Sharing



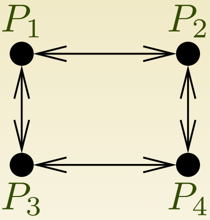
Heterogeneous platform



Virtual ring

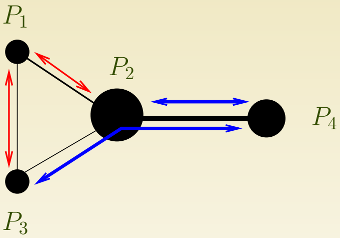


Heterogeneous platform

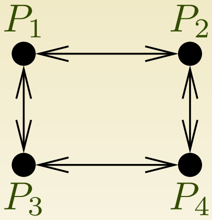


Virtual ring

New Difficulty: Dommunication Links Sharing



Heterogeneous platform

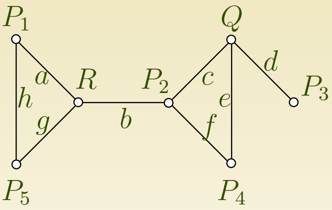


Virtual ring

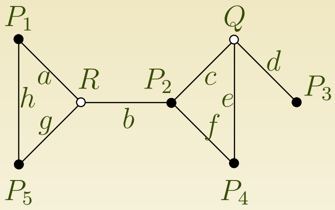
We must take communication link sharing into account.

New Notations

- A set of communications links: e_1, ..., e_n
- Bandwidth of link e_m: b_{e_m}
- There is a path $\mathcal{S}_i$ from P_i to $P_{\mathsf{succ}(i)}$ in the network
  - $\mathcal{S}_i$ uses a fraction $s_{i,m}$ of the bandwidth b_{e_m} of link e_m
  - P_i needs a time $D_c.\dfrac{1}{\min_{e_m \in \mathcal{S}_i} s_{i,m}}$ to send to its successor a message of size D_c
  - Constraints on the bandwidth of e_m: $\displaystyle\sum_{1 \leqslant i \leqslant p} s_{i,m} \leqslant b_{e_m}$
- Symmetrically, there is a path $\mathcal{P}_i$ from P_i to $P_{\mathsf{pred}(i)}$ in the network, which uses a fraction $p_{i,m}$ of the bandwidth b_{e_m} of link e_m

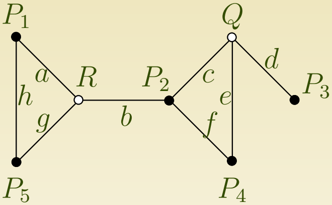


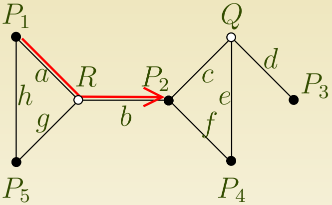
- 7 processors and 8 bidirectional communications links
- We choose a ring of 5 processors:
  $P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4 \rightarrow P_5$ (we use neither Q, nor R)



- 7 processors and 8 bidirectional communications links
- We choose a ring of 5 processors:
  $P_1 \to P_2 \to P_3 \to P_4 \to P_5$ (we use neither Q, nor R)

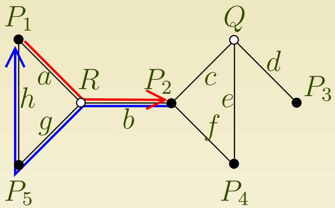
Toy Example: Choosing the Paths





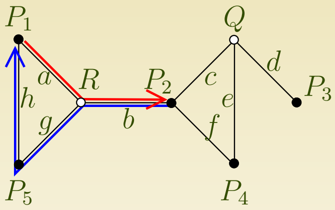
From P_1 to P_2, we use the links a and b: $\mathcal{S}_1 = \{a, b\}$.

Toy Example: Choosing the Paths



From P_1 to P_2, we use the links a and b: $\mathcal{S}_1 = \{a, b\}$.
From P_2 to P_1, we use the links b, g and h: $\mathcal{P}_2 = \{b, g, h\}$.

Toy Example: Choosing the Paths



From P_1 to P_2, we use the links a and b: $\mathcal{S}_1 = \{a, b\}$.
From P_2 to P_1, we use the links b, g and h: $\mathcal{P}_2 = \{b, g, h\}$.

From P_1: to P_2, $\mathcal{S}_1 = \{a, b\}$ and to P_5, $\mathcal{P}_1 = \{h\}$
From P_2: to P_3, $\mathcal{S}_2 = \{c, d\}$ and to P_1, $\mathcal{P}_2 = \{b, g, h\}$
From P_3: to P_4, $\mathcal{S}_3 = \{d, e\}$ and to P_2, $\mathcal{P}_3 = \{d, e, f\}$
From P_4: to P_5, $\mathcal{S}_4 = \{f, b, g\}$ and to P_3, $\mathcal{P}_4 = \{e, d\}$
From P_5: to P_1, $\mathcal{S}_5 = \{h\}$ and to P_4, $\mathcal{P}_5 = \{g, b, f\}$

From P_1 to P_2 we use links a and b: $c_{1,2} = \frac{1}{\min(s_{1,a}, s_{1,b})}$.
From P_1 to P_5 we use the link h: $c_{1,5} = \frac{1}{p_{1,h}}$.

From P_1 to P_2 we use links a and b: $c_{1,2} = \frac{1}{\min(s_{1,a}, s_{1,b})}$.
From P_1 to P_5 we use the link h: $c_{1,5} = \frac{1}{p_{1,h}}$.

Set of all sharing constraints:

Lien a: $s_{1,a} \leqslant b_a$

Lien b: $s_{1,b} + s_{4,b} + p_{2,b} + p_{5,b} \leqslant b_b$

Lien c: $s_{2,c} \leqslant b_c$

Lien d: $s_{2,d} + s_{3,d} + p_{3,d} + p_{4,d} \leqslant b_d$

Lien e: $s_{3,e} + p_{3,e} + p_{4,e} \leqslant b_e$

Lien f: $s_{4,f} + p_{3,f} + p_{5,f} \leqslant b_f$

Lien g: $s_{4,g} + p_{2,g} + p_{5,g} \leqslant b_g$

Lien h: $s_{5,h} + p_{1,h} + p_{2,h} \leqslant b_h$

Toy Example: Final Quadratic System

MINIMIZE $\max_{1 \leqslant i \leqslant 5} \left(\alpha_i.D_w.w_i + D_c.(c_{i,i-1} + c_{i,i+1})\right)$ UNDER THE CONSTRAINTS

$$
\left\{
\begin{array}{lll}
\sum_{i=1}^{5} \alpha_i = 1 & & \\
s_{1,a} \leqslant b_a & s_{1,b} + s_{4,b} + p_{2,b} + p_{5,b} \leqslant b_b & s_{2,c} \leqslant b_c \\
s_{2,d} + s_{3,d} + p_{3,d} + p_{4,d} \leqslant b_d & s_{3,e} + p_{3,e} + p_{4,e} \leqslant b_e & s_{4,f} + p_{3,f} + p_{5,f} \leqslant b_f \\
s_{4,g} + p_{2,g} + p_{5,g} \leqslant b_g & s_{5,h} + p_{1,h} + p_{2,h} \leqslant b_h & \\
s_{1,a}.c_{1,2} \geqslant 1 & s_{1,b}.c_{1,2} \geqslant 1 & p_{1,h}.c_{1,5} \geqslant 1 \\
s_{2,c}.c_{2,3} \geqslant 1 & s_{2,d}.c_{2,3} \geqslant 1 & p_{2,b}.c_{2,1} \geqslant 1 \\
p_{2,g}.c_{2,1} \geqslant 1 & p_{2,h}.c_{2,1} \geqslant 1 & s_{3,d}.c_{3,4} \geqslant 1 \\
s_{3,e}.c_{3,4} \geqslant 1 & p_{3,d}.c_{3,2} \geqslant 1 & p_{3,e}.c_{3,2} \geqslant 1 \\
p_{3,f}.c_{3,2} \geqslant 1 & s_{4,f}.c_{4,5} \geqslant 1 & s_{4,b}.c_{4,5} \geqslant 1 \\
s_{4,g}.c_{4,5} \geqslant 1 & p_{4,e}.c_{4,3} \geqslant 1 & p_{4,d}.c_{4,3} \geqslant 1 \\
s_{5,h}.c_{5,1} \geqslant 1 & p_{5,g}.c_{5,4} \geqslant 1 & p_{5,b}.c_{5,4} \geqslant 1 \\
p_{5,f}.c_{5,4} \geqslant 1 & &
\end{array}
\right.
$$

Toy Example: Conclusion

The problem sums up to a quadratic system if

1. The processors are selected;
2. The processors are ordered into a ring;
3. The communication paths between the processors are known.

In other words: a quadratic system if the ring is known.

Toy Example: Conclusion

The problem sums up to a quadratic system if

1. The processors are selected;
2. The processors are ordered into a ring;
3. The communication paths between the processors are known.

In other words: a quadratic system if the ring is known.

If the ring is known:

- Complete graph: closed-form expression;
- General graph: quadratic system.

And, in Practice ?

We adapt our greedy heuristic:

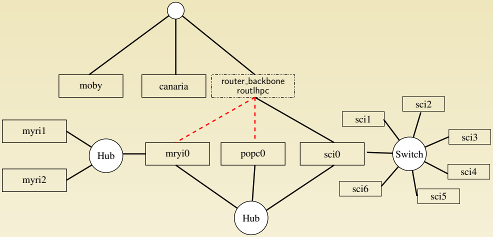
1. Initially: best pair of processors
2. For each processor P_k (not already included in the ring)
   - For each pair (P_i, P_j) of neighbors in the ring
     1. We build the graph of the unused bandwidths (Without considering the paths between P_i and P_j)
     2. We compute the shortest paths (in terms of bandwidth) between P_k and P_i and P_j
     3. We evaluate the solution
3. We keep the best solution found at step 2 and we start again

$+$ refinements (max-min fairness, quadratic solving).

Is This Meaningful ?

- No guarantee, neither theoretical, nor practical
- Simple solution:
  1. we build the complete graph whose edges are labeled with the bandwidths of the best communication paths
  2. we apply the heuristic for complete graphs
  3. we allocate the bandwidths

Example: an Actual Platform (Lyon)



Topology

P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
0.0206	0.0206	0.0206	0.0206	0.0291	0.0206	0.0087	0.0206	0.0206

P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}
0.0206	0.0206	0.0206	0.0291	0.0451	0	0	0

Processors processing times (in seconds par megaflop)

Results

First heuristic building the ring without taking link sharing into account

Second heuristic taking into account link sharing (and with quadratic programing)

Ratio D_c/D_w	H1		H2		Gain
0.64	0.008738	(1)	0.008738	(1)	0%
0.064	0.018837	(13)	0.006639	(14)	64.75%
0.0064	0.003819	(13)	0.001975	(14)	48.28%

Ratio D_c/D_w	H1		H2		Gain
0.64	0.005825	(1)	0.005825	(1)	0 %
0.064	0.027919	(8)	0.004865	(6)	82.57%
0.0064	0.007218	(13)	0.001608	(8)	77.72%

Table: T_{step}/D_w for each heuristic on Lyon's and Strasbourg's platforms (the numbers in parentheses show the size of the rings built).

Conclusion

Even though this is a very basic application, it illustrates many difficulties encountered when:

- Processors have different characteristics
- Communications links have different characteristics
- There is an irregular interconnection network with complex bandwidth sharing issues.

We need to use a realistic model of networks... Even though a more realistic model leads to a much more complicated problem, this is worth the effort as derived solutions are more efficient in practice.

Outline

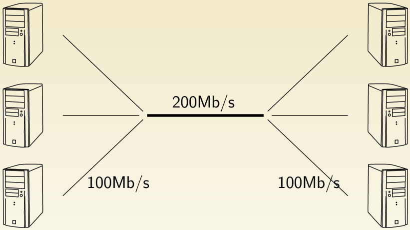
5 Scheduling Divisible Workload

6 Iterative Algorithms

7 Data Redistribution

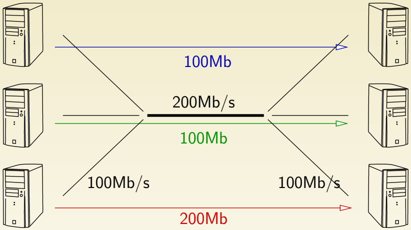
A Data Redistribution Problem

When coupling code, data often have to be redistributed from one cluster to another. Using "Brute force" is generally not a good idea [Wag05].



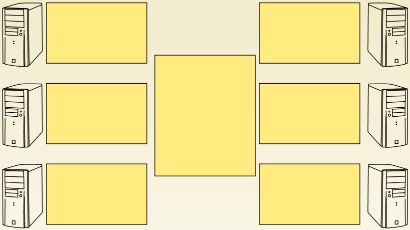
A Data Redistribution Problem

When coupling code, data often have to be redistributed from one cluster to another. Using "Brute force" is generally not a good idea [Wag05].

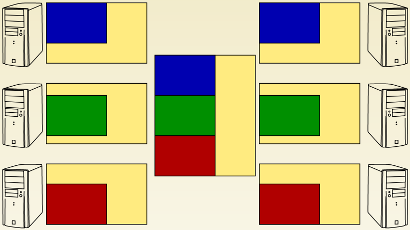


A Data Redistribution Problem

When coupling code, data often have to be redistributed from one cluster to another. Using “Brute force” is generally not a good idea [Wag05].

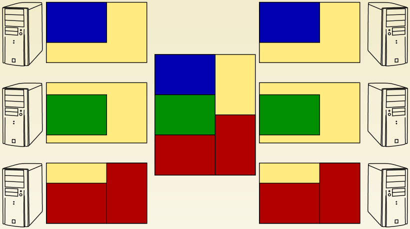


When coupling code, data often have to be redistributed from one cluster to another. Using "Brute force" is generally not a good idea [Wag05].



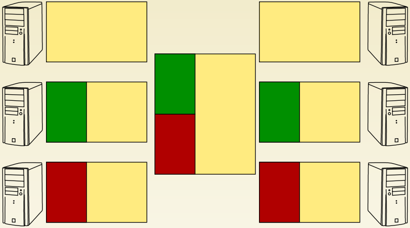
A Data Redistribution Problem

When coupling code, data often have to be redistributed from one cluster to another. Using "Brute force" is generally not a good idea [Wag05].



Non-cooperative: $C_{\max} = 2.5$

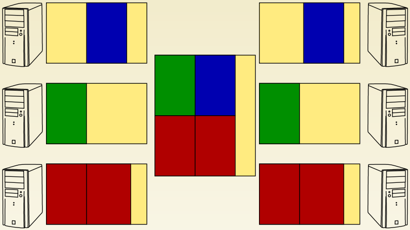
When coupling code, data often have to be redistributed from one cluster to another. Using "Brute force" is generally not a good idea [Wag05].



Non-cooperative: $C_{\max} = 2.5$

A Data Redistribution Problem

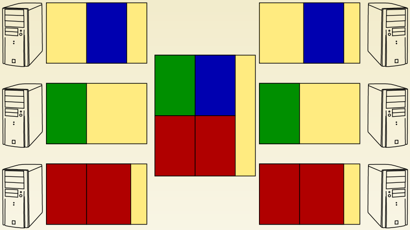
When coupling code, data often have to be redistributed from one cluster to another. Using "Brute force" is generally not a good idea [Wag05].



Non-cooperative: $C_{\max} = 2.5$
Optimal: $C_{\max} = 2$
The bottleneck moves $\leadsto$ resource waste

A Data Redistribution Problem

When coupling code, data often have to be redistributed from one cluster to another. Using "Brute force" is generally not a good idea [Wag05].



Non-cooperative: $C_{\max} = 2.5$
Optimal: $C_{\max} = 2$
The bottleneck moves $\rightsquigarrow$ resource waste

Moreover, opening dozens of connections at the same time is generally very intrusive for other users and often leads to performance degradation.

Modeling

Input

- b_1 is the bandwidth of the sending cluster
- b_2 is the bandwidth of the receiving cluster
- b_b is the bandwidth of the backbone
- β is the latency of communications
- The redistribution is modeled by a bipartite graph $G = (V_1, V_2, m, E)$. $m(v_1, v_2)$ is the amount of data to transfer from v_1 to v_2.

A given processor can communicate with at most one processor at a time. Therefore we try to decompose our redistribution as a set of synchronous communication steps.

Output We look for a set D of h matching $M_1 = (E_1, m_1), \ldots, M_h = (E_h, m_h)$ such that:

$$\forall (v_1, v_2) \in E : m(v_1, v_2) = \sum_{l=1}^{h} m_l(v_1, v_2)$$

Objective function The time needed for a communication step M_l is equal to. . .

Modeling

Objective function The time needed for a communication step M_l is equal to. . . It is unclear. It depends on the bandwidth sharing. This is why the problem has been modeled in a different way. Let's do it one more time!

Let us denote by $w(v_1, v_2)$ the minimum communication time to transfer $m(v_1, v_2)$ from v_1 to v_2.

$$w(v_1, v_2) = \frac{m(v_1, v_2)}{\min(b_1, b_2, b_b)}$$

The maximum number of flows that can be sent at full speed is bounded by:

$$k = \left\lceil \frac{b_b}{\min(b_1, b_2, b_b)} \right\rceil$$

K-Preemptive Bipartite Scheduling

Input

- β is the latency of communications
- The redistribution is modeled by a bipartite graph $G = (V_1, V_2, w, E)$. $w(v_1, v_2)$ is the time required to transfer data from v_1 to v_2.
- At most k simultaneous communications can be done.

Output We look for a set D of h matching $M_1 = (E_1, w_1), \ldots, M_h = (E_h, w_h)$ such that:

$$\forall (v_1, v_2) \in E : w(v_1, v_2) = \sum_{l=1}^{h} w_l(v_1, v_2)$$

and

$$\forall l : |E_l| \leqslant k$$

Objective function The time needed for a communication step M_l is equal to

$$c(M_l) = \max_{e \in E} w_l(e) + \beta$$

Therefore, the cost of a distribution D is

$$c(D) = \sum_{l=1}^{h} w_l(v_1, v_2) = h\beta + \sum_{l=1}^{h} \max_{e \in E} w_l(e)$$

There are two difficulties:

- The trade-off between the number of steps and the latency.
- We look for bounded-size matchings.

PBS is the exact same problem where the bound on the size of matchings is removed.

A few results on the KPBS complexity

- KPBS is strong NP-hard.
- PBS cannot be approximated with a ratio smaller than $\frac{7}{6}$.
- PBS can be approximated with a ratio $2 - \frac{1}{\beta+1}$.
- KPBS can be approximated with a ratio $\frac{8}{3}$.

Getting Rid of Annoying Constraints

- The k bound is somehow artificial but is due to the 1-port model.
- By getting rid of the latencies, you get a polynomial fractionnal matching problem (if you have understood the previous talk that used linear programing and ellipsoid, you should see why).
- With a few "standard tricks" you can even introduce release dates and optimize the maximum weighted flow instead of the makespan...
- However, taking the whole topology into account is more tricky.
  - Indeed, under a bounded multiport model, the problem is trivial.
  - However, if you want to keep the 1-port constraint, you need some matching with non-uniform bandwidth allocation, which seems to be more tricky.

Note there are also problems for which the latency is not an issue but where the hardness really comes from the bound on the number of simulataneous connections [MYCR06].

Conclusion

Ensure that all parts of your modeling are mandatory.

Maybe if k is large in practice and your latencies can be somehow overlapped, then they may not be worth being considered.

A. Alexandrov, M. Ionescu, K. Schauser, and C. Scheiman.
LogGP: Incorporating long messages into the LogP model for parallel computation.
Journal of Parallel and Distributed Computing, 44(1):71–79, 1997.

Beaumont, Olivier and Casanova, Henri and Legrand, Arnaud and Robert, Yves and Yang, Yang.
Scheduling Divisible Loads on Star and Tree Networks: Results and Open Problems.
IEEE Trans. Parallel Distributed Systems, 16(3):207–218, 2005.

V. Bharadwaj, D. Ghose, V. Mani, and T.G. Robertazzi.
Scheduling Divisible Loads in Parallel and Distributed Systems.
IEEE Computer Society Press, 1996.

D. Culler, R. Karp, D. Patterson, A. Sahay, E. Santos, K. Schauser, R. Subramonian, and T. von Eicken.
LogP: a practical model of parallel computation.

Communication of the ACM, 39(11):78–85, 1996.

R. W. Hockney.
The communication challenge for mpp : Intel paragon and meiko cs-2.
Parallel Computing, 20:389–398, 1994.

B. Hong and V.K. Prasanna.
Distributed adaptive task allocation in heterogeneous computing environments to maximize throughput.
In *International Parallel and Distributed Processing Symposium IPDPS'2004*. IEEE Computer Society Press, 2004.

T. Kielmann, H. E. Bal, and K. Verstoep.
Fast measurement of LogP parameters for message passing platforms.
In *Proceedings of the 15th IPDPS. Workshops on Parallel and Distributed Processing*, 2000.

Steven H. Low.
A duality model of TCP and queue management algorithms.

IEEE/ACM Transactions on Networking, 2003.

Dong Lu, Yi Qiao, Peter A. Dinda, and Fabián E. Bustamante.

Characterizing and predicting tcp throughput on the wide area network.
In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, 2005.

Arnaud Legrand, Hélène Renard, Yves Robert, and Frédéric Vivien.
Mapping and load-balancing iterative computations on heterogeneous clusters with shared links.
IEEE Trans. Parallel Distributed Systems, 15(6):546–558, 2004.

Maxime Martinasso.
Analyse et modélisation des communications concurrentes dans les réseaux haute performance.
PhD thesis, Université Joseph Fourier de Grenoble, 2007.

Laurent Massoulié and James Roberts.

Bandwidth sharing: Objectives and algorithms.
In *INFOCOM (3)*, pages 1395–1403, 1999.

Loris Marchal, Yang Yang, Henri Casanova, and Yves Robert.
Steady-state scheduling of multiple divisible load applications on wide-area distributed computing platforms.
Int. Journal of High Performance Computing Applications, (3), 2006.

T.G. Robertazzi.
Divisible Load Scheduling.
http://www.ece.sunysb.edu/~tom/dlt.html.

Frédéric Wagner.
Redistribution de données à travers un réseau haut débit.
PhD thesis, Université Henri Poincaré Nancy 1, 2005.

Yang, Yang and Casanova, Henri and Drozdowski, Maciej and Lawenda, Marcin and Legrand, Arnaud.
On the Complexity of Multi-Round Divisible Load Scheduling.
Research Report 6096, INRIA, 01 2007.