Overview of Scheduling 1/2

Loris MARCHAL

(with the help of Olivier BEAUMONT, Henri CASANOVA, Arnaud LEGRAND, Yves ROBERT and Frédéric VIVIEN)

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

loris.marchal@ens-lyon.fr

ACIS Laboratory
September 27, 2006

Evolution of parallel machines

An ever-increasing demand of computing power

Parallelism is an attempt to answer

Parallel algorithm design and scheduling were already difficult tasks with homogeneous machines

On heterogeneous platforms, it gets worse

Evolution of parallel machines

An ever-increasing demand of computing power
Parallelism is an attempt to answer

Parallel algorithm design and scheduling were already difficult tasks with homogeneous machines
On heterogeneous platforms, it gets worse

Evolution of parallel machines

An ever-increasing demand of computing power
Parallelism is an attempt to answer

Parallel algorithm design and scheduling were already difficult tasks with homogeneous machines
On heterogeneous platforms, it gets worse

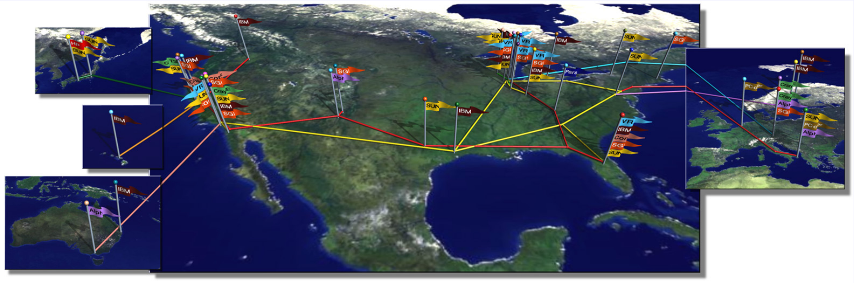
Evolution of parallel machines

An ever-increasing demand of computing power
Parallelism is an attempt to answer

Parallel algorithm design and scheduling were already difficult tasks with homogeneous machines
On heterogeneous platforms, it gets worse

An ever-increasing demand of computing power

Parallelism is an attempt to answer

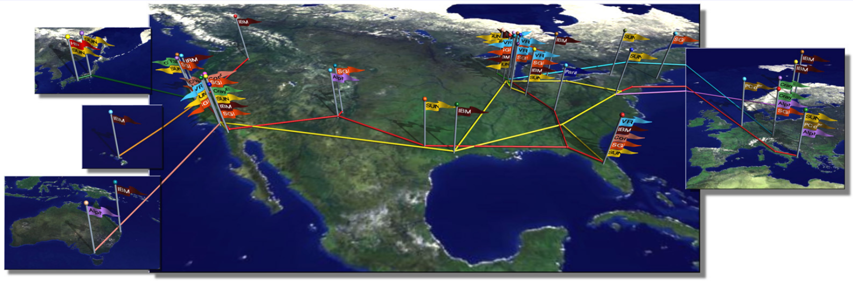


Parallel algorithm design and scheduling were already difficult tasks with homogeneous machines

On heterogeneous platforms, it gets worse

An ever-increasing demand of computing power

Parallelism is an attempt to answer

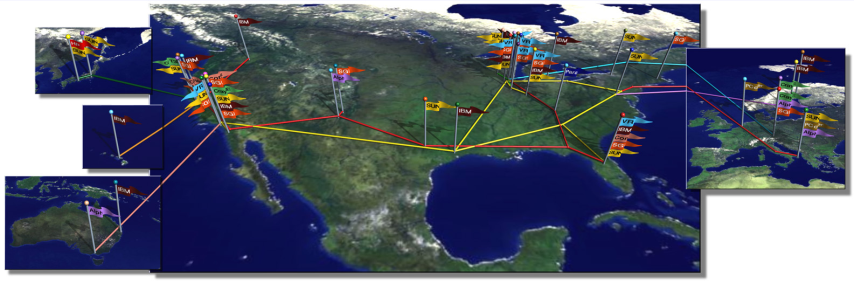


Parallel algorithm design and scheduling were already difficult tasks with homogeneous machines

On heterogeneous platforms, it gets worse

Evolution of parallel machines

An ever-increasing demand of computing power
Parallelism is an attempt to answer



Parallel algorithm design and scheduling were already difficult tasks with homogeneous machines
On heterogeneous platforms, it gets worse

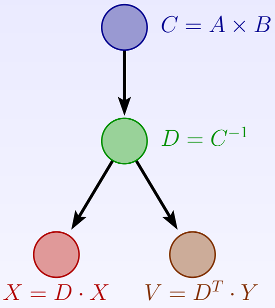
Outline

1. Background on traditional scheduling
2. Divisible Load Scheduling (or changing the task model)
3. Simulation for Grid Computing (next week)
4. Steady-State Scheduling (next week)

Outline

Traditional scheduling – Framework

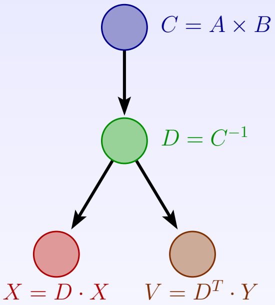
- Application = DAG $G = (\mathcal{T}, E, w)$
  - $\mathcal{T}$ = set of tasks
  - E = dependence constraints
  - $w(T)$ = computational cost of task T (execution time)
  - $c(T, T')$ = communication cost (data sent from T to T')



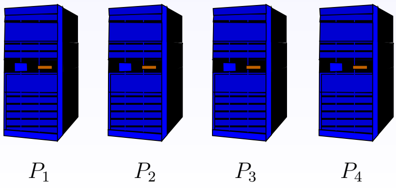
- Platform
  - Set of p identical processors

Traditional scheduling – Framework

- Application = DAG $G = (\mathcal{T}, E, w)$
  - $\mathcal{T}$ = set of tasks
  - E = dependence constraints
  - $w(T)$ = computational cost of task T (execution time)
  - $c(T, T')$ = communication cost (data sent from T to T')

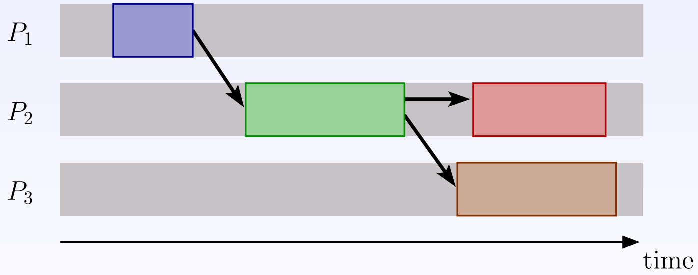


- Platform
  - Set of p identical processors



Traditional scheduling – Framework

- Schedule
  - $\sigma(T)$ = date to begin execution of task T
  - alloc(T) = processor assigned to it



Traditional scheduling – Constraints

- Data dependences If $(T, T') \in E$ then
  - if alloc(T) = alloc(T') then
    $$\sigma(T) + w(T) \leq \sigma(T')$$
  - if alloc(T) $\neq$ alloc(T') then
    $$\sigma(T) + w(T) + c(T, T') \leq \sigma(T')$$
- Resource constraints
  $$\text{alloc}(T) = \text{alloc}(T') \Rightarrow \\ [\sigma(T), \sigma(T) + w(T)[\bigcap [\sigma(T'), \sigma(T') + w(T')[= \emptyset$$

Traditional scheduling – Constraints

- Data dependences If $(T, T') \in E$ then
  - if alloc(T) = alloc(T') then
    $$\sigma(T) + w(T) \leq \sigma(T')$$
  - if alloc$(T) \neq$ alloc(T') then
    $$\sigma(T) + w(T) + c(T, T') \leq \sigma(T')$$
- Resource constraints

$$\text{alloc}(T) = \text{alloc}(T') \Rightarrow$$
$$[\sigma(T), \sigma(T) + w(T)[\bigcap[\sigma(T'), \sigma(T') + w(T')[= \emptyset$$

Traditional scheduling – Constraints

- Data dependences If $(T, T') \in E$ then
  - if alloc(T) = alloc(T') then
    $$\sigma(T) + w(T) \leq \sigma(T')$$
  - if alloc$(T) \neq$ alloc(T') then
    $$\sigma(T) + w(T) + c(T, T') \leq \sigma(T')$$
- Resource constraints
  $$\text{alloc}(T) = \text{alloc}(T') \Rightarrow [\sigma(T), \sigma(T) + w(T)[\bigcap[\sigma(T'), \sigma(T') + w(T')[= \emptyset$$

Traditional scheduling – Constraints

- **Data dependences** If $(T, T') \in E$ then
  - if alloc(T) = alloc(T') then

$$\sigma(T) + w(T) \leq \sigma(T')$$

  - if alloc$(T) \neq$ alloc(T') then

$$\sigma(T) + w(T) + c(T, T') \leq \sigma(T')$$

- **Resource constraints**

$$\text{alloc}(T) = \text{alloc}(T') \Rightarrow \\ [\sigma(T), \sigma(T) + w(T)[\bigcap [\sigma(T'), \sigma(T') + w(T')[= \emptyset$$

Traditional scheduling – Objective functions

- Makespan or total execution time

$$MS(\sigma) = \max_{T \in \mathcal{T}} \left(\sigma(T) + w(T)\right)$$

- Other classical objectives:
  - Sum of completion times
  - With arrival times: maximum flow (response time), or sum flow
  - More fairness with maximum stretch, or sum stretch

Traditional scheduling – Objective functions

- Makespan or total execution time

$$MS(\sigma) = \max_{T \in \mathcal{T}} \left(\sigma(T) + w(T)\right)$$

- Other classical objectives:
  - Sum of completion times
  - With arrival times: maximum flow (response time), or sum flow
  - More fairness with maximum stretch, or sum stretch

Traditional scheduling – Objective functions

- Makespan or total execution time

$$MS(\sigma) = \max_{T \in \mathcal{T}} \left(\sigma(T) + w(T)\right)$$

- Other classical objectives:
  - Sum of completion times
  - With arrival times: maximum flow (response time), or sum flow
  - More fairness with maximum stretch, or sum stretch

Traditional scheduling – Objective functions

- Makespan or total execution time

$$MS(\sigma) = \max_{T \in \mathcal{T}} (\sigma(T) + w(T))$$

- Other classical objectives:
  - Sum of completion times
  - With arrival times: maximum flow (response time), or sum flow
  - More fairness with maximum stretch, or sum stretch

Traditional scheduling – Objective functions

- Makespan or total execution time

$$MS(\sigma) = \max_{T \in \mathcal{T}} \left(\sigma(T) + w(T)\right)$$

- Other classical objectives:
  - Sum of completion times
  - With arrival times: maximum flow (response time), or sum flow
  - More fairness with maximum stretch, or sum stretch

Traditional scheduling – About the model

- Simple but OK for computational resources
  - No CPU sharing, even in models with preemption
  - At most one task running per processor at any time-step
- Very crude for network resources
  - Unbounded bandwidth on any link
  - Unlimited number of simultaneous sends/receives per processor
  - Fully connected interconnection graph (clique)

In fact, model assumes infinite network capacity

Traditional scheduling – About the model

- Simple but OK for computational resources
  - No CPU sharing, even in models with preemption
  - At most one task running per processor at any time-step
- Very crude for network resources
  - Unbounded bandwidth on any link
  - Unlimited number of simultaneous sends/receives per processor
  - Fully connected interconnection graph (clique)

In fact, model assumes infinite network capacity

Traditional scheduling – About the model

- Simple but OK for computational resources
  - No CPU sharing, even in models with preemption
  - At most one task running per processor at any time-step
- Very crude for network resources
  - Unbounded bandwidth on any link
  - Unlimited number of simultaneous sends/receives per processor
  - Fully connected interconnection graph (clique)

In fact, model assumes infinite network capacity

Traditional scheduling – About the model

- Simple but OK for computational resources
  - No CPU sharing, even in models with preemption
  - At most one task running per processor at any time-step
- Very crude for network resources
  - Unbounded bandwidth on any link
  - Unlimited number of simultaneous sends/receives per processor
  - Fully connected interconnection graph (clique)

In fact, model assumes infinite network capacity

Traditional scheduling – About the model

- Simple but OK for computational resources
  - No CPU sharing, even in models with preemption
  - At most one task running per processor at any time-step
- Very crude for network resources
  - Unbounded bandwidth on any link
  - Unlimited number of simultaneous sends/receives per processor
  - Fully connected interconnection graph (clique)

In fact, model assumes infinite network capacity

Traditional scheduling – About the model

- Simple but OK for computational resources
  - No CPU sharing, even in models with preemption
  - At most one task running per processor at any time-step
- Very crude for network resources
  - Unbounded bandwidth on any link
  - Unlimited number of simultaneous sends/receives per processor
  - Fully connected interconnection graph (clique)

In fact, model assumes infinite network capacity

Traditional scheduling – About the model

- Simple but OK for computational resources
  - No CPU sharing, even in models with preemption
  - At most one task running per processor at any time-step
- Very crude for network resources
  - Unbounded bandwidth on any link
  - Unlimited number of simultaneous sends/receives per processor
  - Fully connected interconnection graph (clique)

In fact, model assumes infinite network capacity

Traditional scheduling – About the model

- Simple but OK for computational resources
  - No CPU sharing, even in models with preemption
  - At most one task running per processor at any time-step
- Very crude for network resources
  - Unbounded bandwidth on any link
  - Unlimited number of simultaneous sends/receives per processor
  - Fully connected interconnection graph (clique)

In fact, model assumes infinite network capacity

Makespan minimization

- NP-hardness
  - $Pb(p)$ NP-complete for independent tasks and no communications ($E = \emptyset$, $p = 2$ and $c = \overline{0}$)
  - $Pb(p)$ NP-complete for UET-UCT graphs ($w = c = \overline{1}$)
- Approximation algorithms
  - Without communications, list scheduling is a $(2 - \frac{1}{p})$-approximation
  - With communications, result extends to coarse-grain graphs
  - With communications, no λ-approximation in general

Makespan minimization

- NP-hardness
  - $Pb(p)$ NP-complete for independent tasks and no communications ($E = \emptyset$, $p = 2$ and $c = \overline{0}$)
  - $Pb(p)$ NP-complete for UET-UCT graphs ($w = c = \overline{1}$)
- Approximation algorithms
  - Without communications, list scheduling is a $(2 - \frac{1}{p})$-approximation
  - With communications, result extends to coarse-grain graphs
  - With communications, no λ-approximation in general

Makespan minimization

- NP-hardness
  - $Pb(p)$ NP-complete for independent tasks and no communications ($E = \emptyset$, $p = 2$ and $c = \overline{0}$)
  - $Pb(p)$ NP-complete for UET-UCT graphs ($w = c = \overline{1}$)
- Approximation algorithms
  - Without communications, list scheduling is a $(2 - \frac{1}{p})$-approximation
  - With communications, result extends to coarse-grain graphs
  - With communications, no λ-approximation in general

Makespan minimization

- NP-hardness
  - $Pb(p)$ NP-complete for independent tasks and no communications ($E = \emptyset$, $p = 2$ and $c = \overline{0}$)
  - $Pb(p)$ NP-complete for UET-UCT graphs ($w = c = \overline{1}$)

- Approximation algorithms
  - Without communications, list scheduling is a $(2 - \frac{1}{p})$-approximation
  - With communications, result extends to coarse-grain graphs
  - With communications, no λ-approximation in general

Makespan minimization

- NP-hardness
  - $Pb(p)$ NP-complete for independent tasks and no communications ($E = \emptyset$, $p = 2$ and $c = \overline{0}$)
  - $Pb(p)$ NP-complete for UET-UCT graphs ($w = c = \overline{1}$)

- Approximation algorithms
  - Without communications, list scheduling is a $(2 - \frac{1}{p})$-approximation
  - With communications, result extends to coarse-grain graphs
  - With communications, no λ-approximation in general

Makespan minimization

- NP-hardness
  - $Pb(p)$ NP-complete for independent tasks and no communications ($E = \emptyset$, $p = 2$ and $c = \overline{0}$)
  - $Pb(p)$ NP-complete for UET-UCT graphs ($w = c = \overline{1}$)

- Approximation algorithms
  - Without communications, list scheduling is a $(2 - \frac{1}{p})$-approximation
  - With communications, result extends to coarse-grain graphs
  - With communications, no λ-approximation in general

Makespan minimization

- NP-hardness
  - $Pb(p)$ NP-complete for independent tasks and no communications ($E = \emptyset$, $p = 2$ and $c = \overline{0}$)
  - $Pb(p)$ NP-complete for UET-UCT graphs ($w = c = \overline{1}$)

- Approximation algorithms
  - Without communications, list scheduling is a $(2 - \frac{1}{p})$-approximation
  - With communications, result extends to coarse-grain graphs
  - With communications, no λ-approximation in general

List scheduling – Without communications (1/2)

- *Initialization:*
  1. Compute priority level of all tasks
  2. Priority queue = list of free tasks (tasks without predecessors) sorted by priority
  3. t is the current time step: $t = 0$.
- *While there remain tasks to execute*:
  1. Add new free tasks, if any, to the queue. If the execution of a task terminates at time step t, suppress this task from the predecessor list of all its successors. Add those tasks whose predecessor list has become empty.
  2. If there are q available processors and r tasks in the queue, remove first $\min(q, r)$ tasks from the queue and execute them; if T is one of these tasks, let $\sigma(T) = t$.
  3. Increment t.

List scheduling – Without communications (2/2)

- Priority level
  - Use critical path: longest path from the task to an exit node
  - Computed recursively by a bottom-up traversal of the graph
- Implementation details
  - Cannot iterate from $t = 0$ to $t = MS(\sigma)$ (exponential in problem size)
  - Use a heap for free tasks valued by priority level
  - Use a heap for processors valued by termination time
  - Complexity $O(|V| \log |V| + |E|)$

List scheduling – With communications (1/2)

- Priority level
  - Use pessimistic critical path: include all edge costs in the weight
  - Computed recursively by a bottom-up traversal of the graph
- MCP Modified Critical Path
  - Assign free task with highest priority to best processor
  - Best processor = finishes execution first, given already taken scheduling decisions
  - Free tasks may not be ready for execution (communication delays)
  - May explore inserting the task in empty slots of schedule
  - Complexity $O(|V| \log |V| + (|E| + |V|)p)$

List scheduling – With communications (2/2)

- ETF Earliest Finish Time
  - Dynamically recompute priorities of free tasks
  - Select free task that finishes execution first (on best processor), given already taken scheduling decisions
  - Higher complexity $O(|V|^3 p)$
  - May miss "urgent" tasks on critical path
- Other approaches
  - Two-step: clustering + load balancing
    - DSC Dominant Sequence Clustering $O((|V| + |E|) \log |V|)$
    - LLB List-based Load Balancing $O(C \log C + |V|)$ (C number of clusters generated by DSC)
  - Low-cost: FCP Fast Critical Path
    - Maintain constant-size sorted list of free tasks:
    - Best processor = first idle or the one sending last message
    - Low complexity $O(|V| \log p + |E|)$

Extending the model to heterogeneous clusters

- Task graph with n tasks $T_1, \ldots, T_n$.
- Platform with p heterogeneous processors $P_1, \ldots, P_p$.
- Computation costs:
  - w_{iq} = execution time of T_i on P_q
  - $\overline{w_i} = \frac{\sum_{q=1}^{p} w_{iq}}{p}$ average execution time of T_i
  - particular case: consistent tasks $w_{iq} = w_i \times \gamma_q$
- Communication costs:
  - data(i, j): data volume for edge $e_{ij} : T_i \rightarrow T_j$
  - v_{qr}: communication time for unit-size message from P_q to P_r (zero if $q = r$)
  - com(i, j, q, r) = data$(i, j) \times v_{qr}$ communication time from T_i executed on P_q to P_j executed on P_r
  - $\overline{\text{com}_{ij}}$ = data$(i, j) \times \frac{\sum_{1 \leq q, r \leq p, q \neq r} v_{qr}}{p(p-1)}$ average communication cost for edge $e_{ij} : T_i \rightarrow T_j$

Rewriting constraints

Dependences For $e_{ij} : T_i \to T_j$, $q = \mathsf{alloc}(T_i)$ and $r = \mathsf{alloc}(T_j)$:

$$\sigma(T_i) + w_{iq} + \mathsf{com}(i, j, q, r) \leq \sigma(T_j)$$

Resources If $q = \mathsf{alloc}(T_i) = \mathsf{alloc}(T_j)$, then

$$(\sigma(T_i) + w_{iq} \leq \sigma(T_j)) \text{ or } (\sigma(T_j) + w_{jq} \leq \sigma(T_i))$$

Makespan

$$\max_{1 \leq i \leq n} \left(\sigma(T_i) + w_{i,\mathsf{alloc}(T_i)}\right)$$

HEFT: Heterogeneous Earliest Finishing Time

1. Priority level:
   - $\text{rank}(T_i) = \overline{w_i} + \max_{T_j \in \text{Succ}(T_i)}(\overline{\text{com}_{ij}} + \text{rank}(T_j))$, where $\text{Succ}(T)$ is the set of successors of T
   - Recursive computation by bottom-up traversal of the graph
2. Allocation
   - For current task T_i, determine best processor P_q: minimize $\sigma(T_i) + w_{iq}$
   - Enforce constraints related to communication costs
   - Insertion scheduling: look for $t = \sigma(T_i)$ s.t. P_q is available during interval $[t, t + w_{iq}[$
3. Complexity: similar to MCP

New platforms, new problems, new solutions

Target platforms: Large-scale heterogeneous platforms
(networks of workstations, clusters, collections of clusters, grids, ...)

New problems

- Heterogeneity of processors (CPU power, memory)
- Heterogeneity of communication links
- Irregularity of interconnection network
- Non-dedicated platforms

Need to adapt algorithms and scheduling strategies: new objective functions, new models

New platforms, new problems, new solutions

Target platforms: Large-scale heterogeneous platforms
(networks of workstations, clusters, collections of clusters, grids, ...)

New problems

- Heterogeneity of processors (CPU power, memory)
- Heterogeneity of communication links
- Irregularity of interconnection network
- Non-dedicated platforms

Need to adapt algorithms and scheduling strategies: new objective functions, new models

New approaches

1. Divisible load model Assume work can be arbitrarily divided into sub-tasks
2. Steady-state throughput Rather than optimizing execution from the very beginning to the very end, optimize execution core instead

Both approaches are relaxation methods: simplifying model allows to tackle more complex problems

Bibliography

- Introductory book:
  Distributed and parallel computing, H. El-Rewini and T. G. Lewis, Manning 1997
- FCP:
  On the complexity of list scheduling algorithms for distributed-memory systems, A. Radulescu and A.J.C. van Gemund, 13th ACM Int Conf. Supercomputing (1999), 68-75
- HEFT:
  Performance-effective and low-complexity task scheduling for heterogeneous computing, H. Topcuoglu and S. Hariri and M.-Y. Wu, IEEE TPDS 13, 3 (2002), 260-274

Outline

1. Background on traditional scheduling

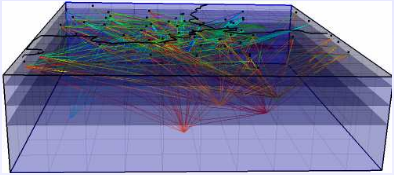
2. Divisible Load Scheduling (or changing the task model)
   - Bus network – Classical approach
   - Bus network – Divisible Load Approach
   - Star network

3. Simulation for Grid Computing (next week)

4. Steady-State Scheduling (next week)

Earth seismic tomography

- Modeling the internal structure of earth



- Validation by comparing expected propagation time of a wave in a model against collected experimental data
- Set of seismic events during 1999: 817101 events
- Original code written for a parallel machine:

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

Divisible load applications

- Divisible load applications can be divided into any number of independent pieces
- Perfectly parallel job: any sub-task can itself be processed in parallel, and on any number of workers
- Model = good approximation for applications that consist of very (very) large numbers of identical, low-granularity computations
- Large spectrum of scientific problems

Divisible load applications

- Divisible load applications can be divided into any number of independent pieces
- Perfectly parallel job: any sub-task can itself be processed in parallel, and on any number of workers
- Model = good approximation for applications that consist of very (very) large numbers of identical, low-granularity computations
- Large spectrum of scientific problems

Divisible load applications

- Divisible load applications can be divided into any number of independent pieces
- Perfectly parallel job: any sub-task can itself be processed in parallel, and on any number of workers
- Model = good approximation for applications that consist of very (very) large numbers of identical, low-granularity computations
- Large spectrum of scientific problems

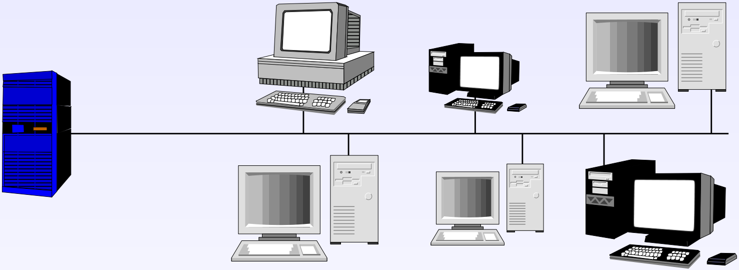
Divisible load applications

- Divisible load applications can be divided into any number of independent pieces
- Perfectly parallel job: any sub-task can itself be processed in parallel, and on any number of workers
- Model = good approximation for applications that consist of very (very) large numbers of identical, low-granularity computations
- Large spectrum of scientific problems

Divisible Load Scheduling (DLS)

- Applications composed of a very (very) large number of low-granularity computations
- Computation time proportional to data volume processed: linear cost model
- Independent computations: neither synchronizations nor communications

Bus network



- Identical links between master and slaves
- Slaves have different computing power

Notations

- Set P_1, ..., P_p of processors
- P_1 is the master, initially holds all data
- Total amount of work: W_{total}
- Processor P_i receives $n_i \in \mathbb{N}$ load units, where $\sum_i n_i = W_{\text{total}}$
  Time for one load unit on P_i: w_i
  Execution time on P_i: $n_i w_i$
- Communication time of one load-unit from P_1 to P_i: c
  One-port model: P_1 serially sends one message to each slave

Notations

- Set P_1, ..., P_p of processors
- P_1 is the master, initially holds all data
- Total amount of work: W_{total}
- Processor P_i receives $n_i \in \mathbb{N}$ load units, where $\sum_i n_i = W_{\text{total}}$
  Time for one load unit on P_i: w_i
  Execution time on P_i: $n_i w_i$
- Communication time of one load-unit from P_1 to P_i: c
  One-port model: P_1 serially sends one message to each slave

Notations

- Set P_1, ..., P_p of processors
- P_1 is the master, initially holds all data
- Total amount of work: W_{total}
- Processor P_i receives $n_i \in \mathbb{N}$ load units, where $\sum_i n_i = W_{\text{total}}$
  Time for one load unit on P_i: w_i
  Execution time on P_i: $n_i w_i$
- Communication time of one load-unit from P_1 to P_i: c
  One-port model: P_1 serially sends one message to each slave

Notations

- Set P_1, ..., P_p of processors
- P_1 is the master, initially holds all data
- Total amount of work: W_{total}
- Processor P_i receives $n_i \in \mathbb{N}$ load units, where $\sum_i n_i = W_{\text{total}}$
  Time for one load unit on P_i: w_i
  Execution time on P_i: $n_i w_i$
- Communication time of one load-unit from P_1 to P_i: c
  One-port model: P_1 serially sends one message to each slave

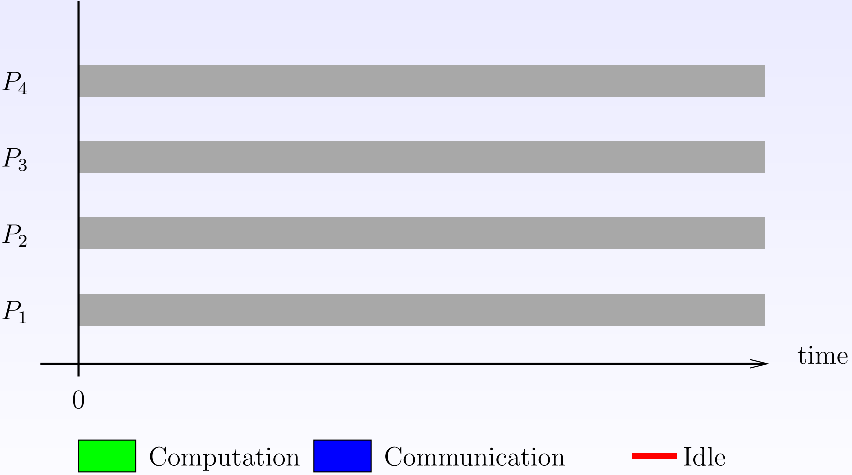
Notations

- Set P_1, ..., P_p of processors
- P_1 is the master, initially holds all data
- Total amount of work: W_{total}
- Processor P_i receives $n_i \in \mathbb{N}$ load units, where $\sum_i n_i = W_{\text{total}}$
  Time for one load unit on P_i: w_i
  Execution time on P_i: $n_i w_i$
- Communication time of one load-unit from P_1 to P_i: c
  One-port model: P_1 serially sends one message to each slave

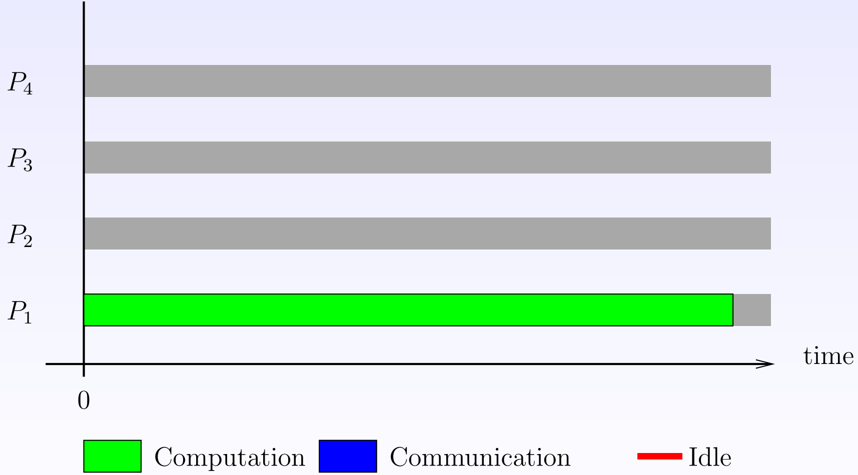
Assessing the model

- One-port hypothesis is realistic ☺
- Linear cost model is simple ☹ but acceptable ☺ for
  - large problems
  - one-round scenarios

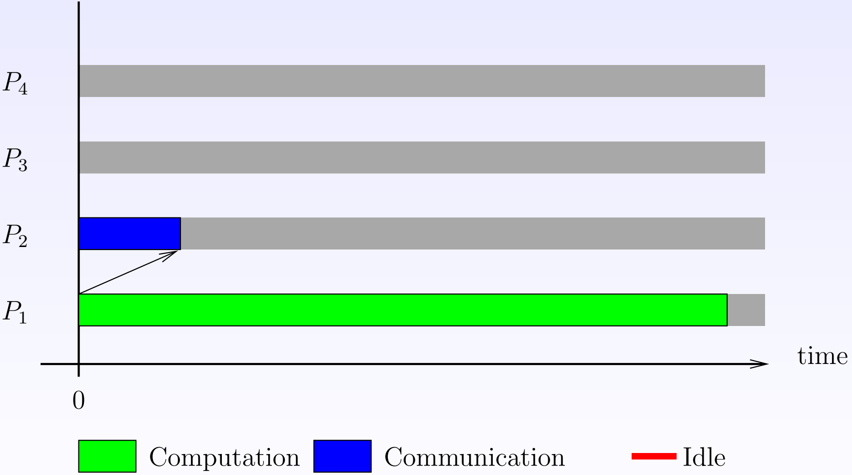
Scheduling example



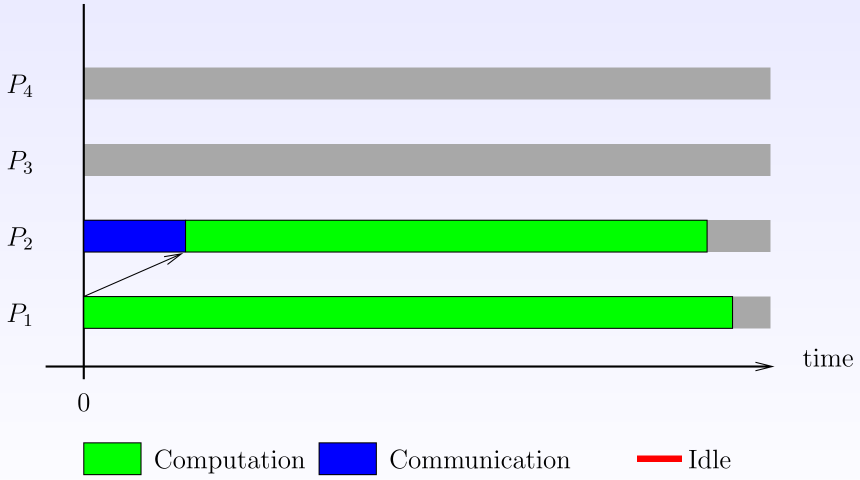
Scheduling example



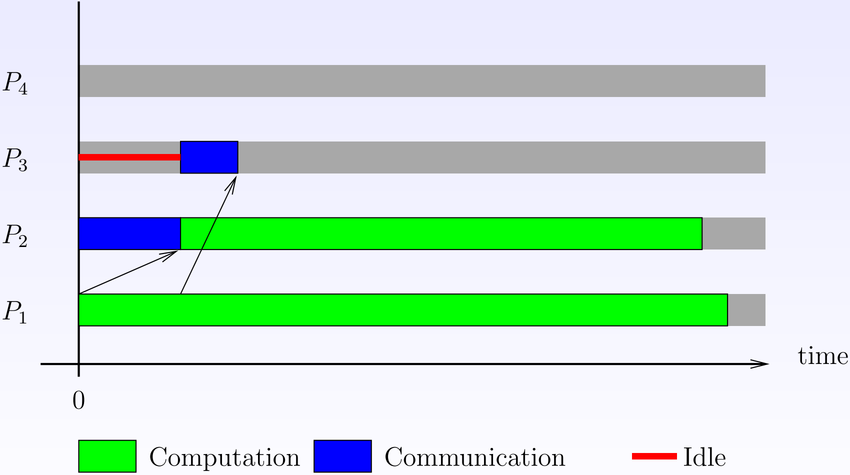
Scheduling example



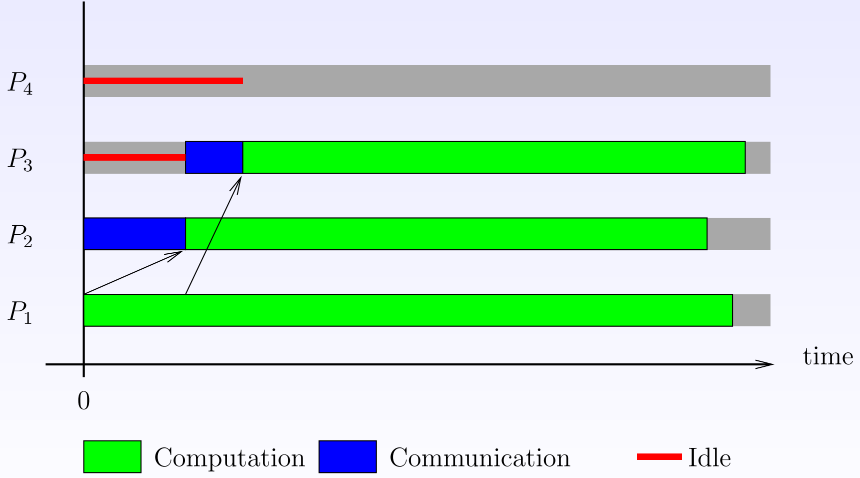
Scheduling example



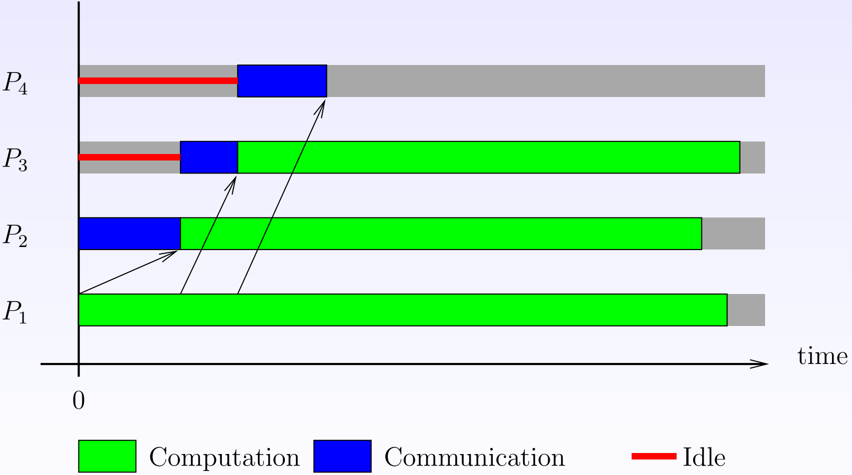
Scheduling example



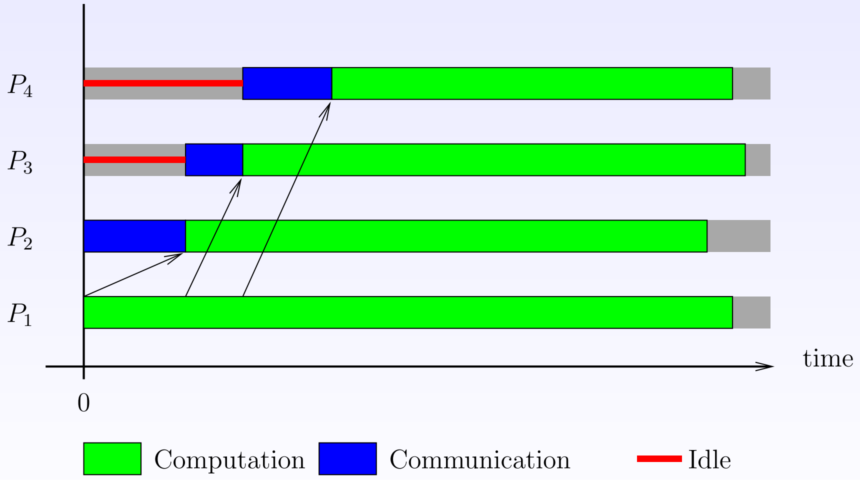
Scheduling example



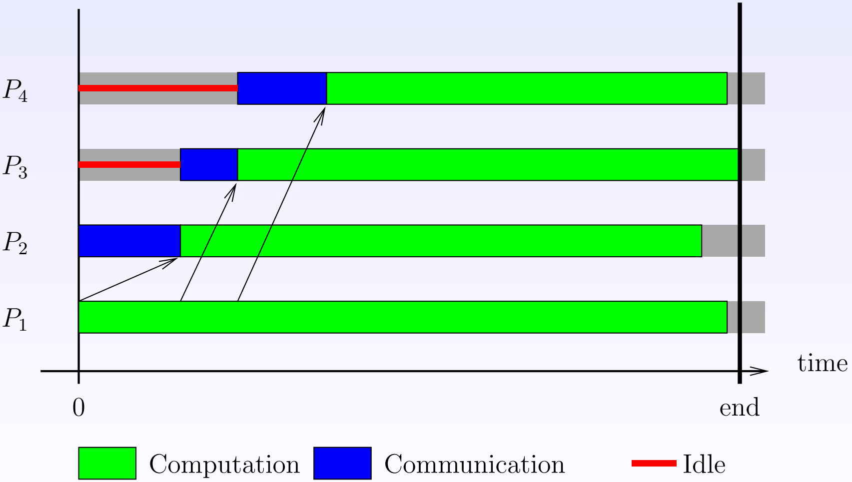
Scheduling example



Scheduling example



Scheduling example



Problem specification

- Master sends n_i data items to P_i within a single message
- Master sequentially sends messages to slaves in the order

$$P_2, P_3, ..., P_p$$

- Simultaneously, master processes its own n_1 data items
- A slave cannot start computing before reception of its message is complete
- Question: compute best allocation and corresponding schedule

Problem specification

- Master sends n_i data items to P_i within a single message
- Master sequentially sends messages to slaves in the order

$$P_2, P_3, ..., P_p$$

- Simultaneously, master processes its own n_1 data items
- A slave cannot start computing before reception of its message is complete
- Question: compute best allocation and corresponding schedule

Problem specification

- Master sends n_i data items to P_i within a single message
- Master sequentially sends messages to slaves in the order

$$P_2, P_3, ..., P_p$$

- Simultaneously, master processes its own n_1 data items
- A slave cannot start computing before reception of its message is complete
- Question: compute best allocation and corresponding schedule

Problem specification

- Master sends n_i data items to P_i within a single message
- Master sequentially sends messages to slaves in the order

$$P_2, P_3, ..., P_p$$

- Simultaneously, master processes its own n_1 data items
- A slave cannot start computing before reception of its message is complete
- Question: compute best allocation and corresponding schedule

Problem specification

- Master sends n_i data items to P_i within a single message
- Master sequentially sends messages to slaves in the order

$$P_2, P_3, ..., P_p$$

- Simultaneously, master processes its own n_1 data items
- A slave cannot start computing before reception of its message is complete
- Question: compute best allocation and corresponding schedule

Equations

- P_1: $T_1 = n_1.w_1$
- P_2: $T_2 = n_2.c + n_2.w_2$
- P_3: $T_3 = (n_2.c + n_3.c) + n_3.w_3$
- P_i: $T_i = \sum_{j=2}^{i} n_j.c + n_i.w_i$ for $i \geq 2$
- P_i: $T_i = \sum_{j=1}^{i} n_j.c_j + n_i.w_i$ for $i \geq 1$
  where $c_1 = 0$ and $c_j = c$ if $j \neq 1$

Equations

- P_1: $T_1 = n_1.w_1$
- P_2: $T_2 = n_2.c + n_2.w_2$
- P_3: $T_3 = (n_2.c + n_3.c) + n_3.w_3$
- P_i: $T_i = \sum_{j=2}^{i} n_j.c + n_i.w_i$ for $i \geq 2$
- P_i: $T_i = \sum_{j=1}^{i} n_j.c_j + n_i.w_i$ for $i \geq 1$
  where $c_1 = 0$ and $c_j = c$ if $j \neq 1$

Equations

- P_1: $T_1 = n_1.w_1$
- P_2: $T_2 = n_2.c + n_2.w_2$
- P_3: $T_3 = (n_2.c + n_3.c) + n_3.w_3$
- P_i: $T_i = \sum_{j=2}^{i} n_j.c + n_i.w_i$ for $i \geq 2$
- P_i: $T_i = \sum_{j=1}^{i} n_j.c_j + n_i.w_i$ for $i \geq 1$
  where $c_1 = 0$ and $c_j = c$ if $j \neq 1$

Equations

- P_1: $T_1 = n_1.w_1$
- P_2: $T_2 = n_2.c + n_2.w_2$
- P_3: $T_3 = (n_2.c + n_3.c) + n_3.w_3$
- P_i: $T_i = \sum_{j=2}^{i} n_j.c + n_i.w_i$ for $i \geq 2$
- P_i: $T_i = \sum_{j=1}^{i} n_j.c_j + n_i.w_i$ for $i \geq 1$
  where $c_1 = 0$ and $c_j = c$ if $j \neq 1$

Equations

- P_1: $T_1 = n_1.w_1$
- P_2: $T_2 = n_2.c + n_2.w_2$
- P_3: $T_3 = (n_2.c + n_3.c) + n_3.w_3$
- P_i: $T_i = \sum_{j=2}^{i} n_j.c + n_i.w_i$ for $i \geq 2$
- P_i: $T_i = \sum_{j=1}^{i} n_j.c_j + n_i.w_i$ for $i \geq 1$
  where $c_1 = 0$ and $c_j = c$ if $j \neq 1$

Execution time (1/2)

$$T = \max_{1 \le i \le p} \left(\sum_{j=1}^{i} n_j.c_j + n_i.w_i \right)$$

Goal: find distribution n_1, ..., n_p to minimize T

Execution time (2/2)

$$T = \max\left(n_1.c_1 + n_1.w_1, \max_{2\leq i\leq p}\left(\sum_{j=1}^{i} n_j.c_j + n_i.w_i\right)\right)$$

$$T = n_1.c_1 + \max\left(n_1.w_1, \max_{2\leq i\leq p}\left(\sum_{j=2}^{i} n_j.c_j + n_i.w_i\right)\right)$$

Optimal distribution of W_{total} data among p processors:
- assign n_1 data items to P_1
- use optimal distribution of $W_{\mathsf{total}} - n_1$ data items among $p-1$ remaining processors

Dynamic Programming Algorithm

solution$[0, p] \leftarrow \mathrm{cons}(0, NIL)$; *cost*$[0, p] \leftarrow 0$
for $d \leftarrow 1$ **to** W_{total} **do**
    solution$[d, p] \leftarrow \mathrm{cons}(d, NIL)$
    cost$[d, p] \leftarrow d \cdot c_p + d \cdot w_p$
for $i \leftarrow p - 1$ **downto** 1 **do**
    solution$[0, i] \leftarrow \mathrm{cons}(0,$ *solution*$[0, i + 1])$
    cost$[0, i] \leftarrow 0$
    for $d \leftarrow 1$ **to** W_{total} **do**
        $(sol, min) \leftarrow (0,$ *cost*$[d, i + 1])$
        for $e \leftarrow 1$ **to** d **do**
            $m \leftarrow e \cdot c_i + \max(e \cdot w_i,$ *cost*$[d - e, i + 1])$
            if $m < min$ **then**
                $(sol, min) \leftarrow (e, m)$
        solution$[d, i] \leftarrow \mathrm{cons}(sol,$ *solution*$[d - sol, i + 1])$
        cost$[d, i] \leftarrow min$
return (*solution*$[W_{\text{total}}, 1]$, *cost*$[W_{\text{total}}, 1]$)

Complexity

- **Theoretical**

$$O(W_{\text{total}}^2 \cdot p)$$

- **In practice**
  With $W_{\text{total}} = 817101$ and $p = 16$, using a 933MHz Pentium III: over two days ...
  (Optimized version: 6 minutes)

Do we really need such an exact integer solution?
No! ☺

Complexity

- **Theoretical**

$$O(W_{\text{total}}^2 \cdot p)$$

- **In practice**

  With $W_{\text{total}} = 817101$ and $p = 16$, using a 933MHz Pentium III: over two days ...
  (Optimized version: 6 minutes)

Do we really need such an exact integer solution?

No! ☺

Complexity

- **Theoretical**

$$O(W_{\text{total}}^2 \cdot p)$$

- **In practice**

  With $W_{\text{total}} = 817101$ and $p = 16$, using a 933MHz Pentium III: over two days ...
  (Optimized version: 6 minutes)

Do we really need such an exact integer solution?
No! ☺

Notations

- Set P_1, ..., P_p of processors
- P_1 is the master, initially holds all data
- Total amount of work: W_{total}
- **Processor P_i receives $\alpha_i W_{\text{total}}$ load units where $\alpha_i W_{\text{total}} \in \mathbb{Q}$ and $\sum_i \alpha_i = 1$**
  Time for one load unit on P_i: w_i
  Execution time on P_i: $\alpha_i w_i$
- Communication time of one load-unit from P_1 to P_i: c
  One-port model: P_1 serially sends one message to each slave

Notations

- Set P_1, ..., P_p of processors
- P_1 is the master, initially holds all data
- Total amount of work: W_{total}
- **Processor P_i receives $\alpha_i W_{\text{total}}$ load units where $\alpha_i W_{\text{total}} \in \mathbb{Q}$ and $\sum_i \alpha_i = 1$**
  Time for one load unit on P_i: w_i
  Execution time on P_i: $\alpha_i w_i$
- Communication time of one load-unit from P_1 to P_i: c
  One-port model: P_1 serially sends one message to each slave

Notations

- Set P_1, ..., P_p of processors
- P_1 is the master, initially holds all data
- Total amount of work: W_{total}
- **Processor P_i receives $\alpha_i W_{\text{total}}$ load units**
  where $\alpha_i W_{\text{total}} \in \mathbb{Q}$ and $\sum_i \alpha_i = 1$
  Time for one load unit on P_i: w_i
  Execution time on P_i: $\alpha_i w_i$
- Communication time of one load-unit from P_1 to P_i: c
  One-port model: P_1 serially sends one message to each slave

Notations

- Set P_1, ..., P_p of processors
- P_1 is the master, initially holds all data
- Total amount of work: W_{total}
- **Processor P_i receives $\alpha_i W_{\textbf{total}}$ load units**
  where $\alpha_i W_{\textbf{total}} \in \mathbb{Q}$ and $\sum_i \alpha_i = 1$
  Time for one load unit on P_i: w_i
  Execution time on P_i: $\alpha_i w_i$
- Communication time of one load-unit from P_1 to P_i: c
  One-port model: P_1 serially sends one message to each slave

Notations

- Set P_1, ..., P_p of processors
- P_1 is the master, initially holds all data
- Total amount of work: W_{total}
- **Processor P_i receives $\alpha_i W_{\textbf{total}}$ load units**
  where $\alpha_i W_{\textbf{total}} \in \mathbb{Q}$ and $\sum_i \alpha_i = 1$
  Time for one load unit on P_i: w_i
  Execution time on P_i: $\alpha_i w_i$
- Communication time of one load-unit from P_1 to P_i: c
  One-port model: P_1 serially sends one message to each slave

Equations

For processor P_i (where $c_1 = 0$ et $c_j = c$ if $j \neq 1$):

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

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

Goal: find distribution α_1, ..., α_p to minimize T

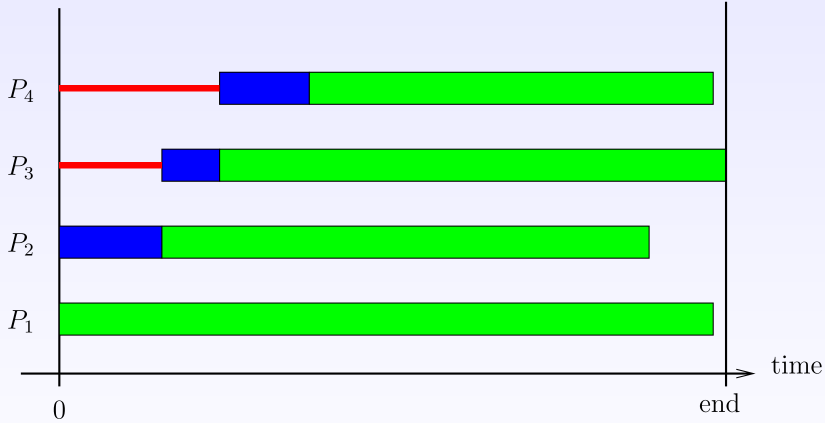
Load balancing property

Lemma

In any optimal solution, all processors terminate execution simultaneously

Proof of Lemma (1/2)

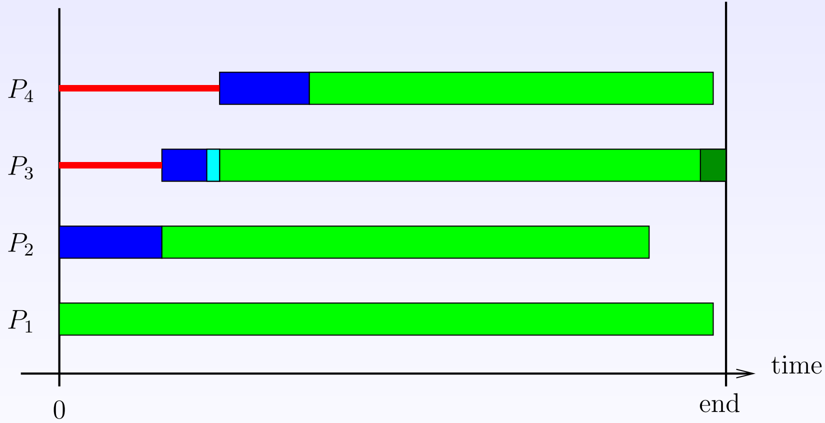
Two slaves i and $i+1$ with $T_i < T_{i+1}$



Decrease α_{i+1} by ε

Proof of Lemma (1/2)

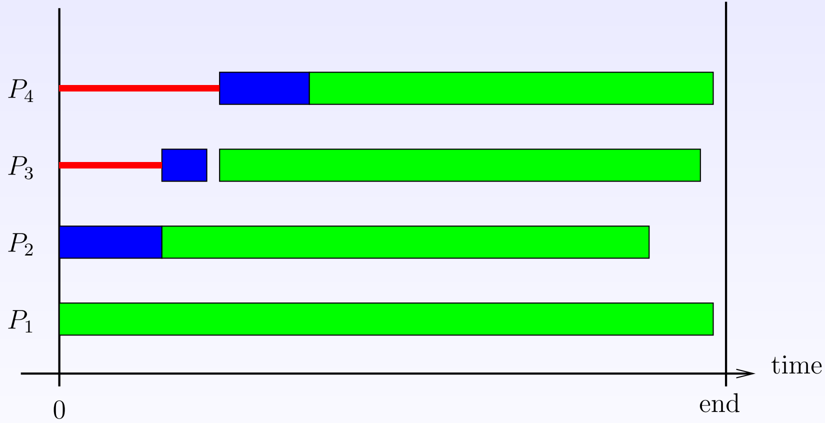
Two slaves i and $i+1$ with $T_i < T_{i+1}$



Decrease α_{i+1} by ε

Proof of Lemma (1/2)

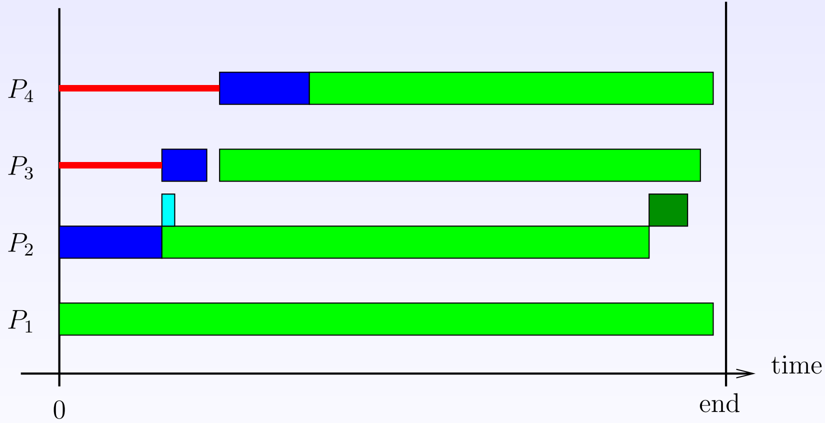
Two slaves i and $i+1$ with $T_i < T_{i+1}$



Decrease α_{i+1} by ε

Proof of Lemma (1/2)

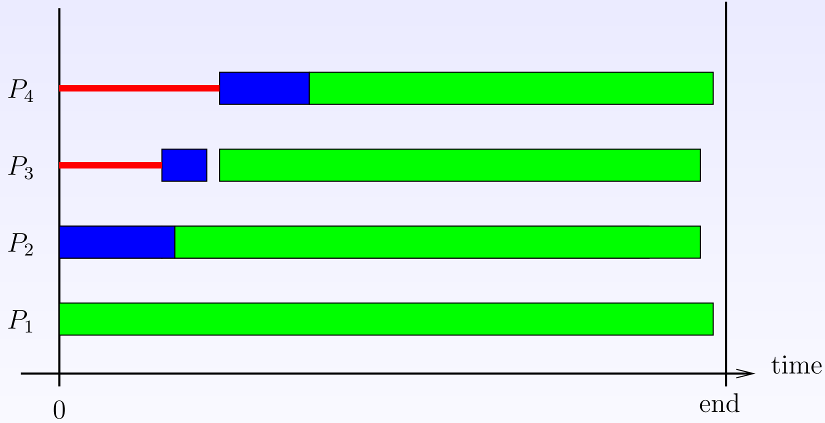
Two slaves i and $i+1$ with $T_i < T_{i+1}$



Increase α_i by ε

Proof of Lemma (1/2)

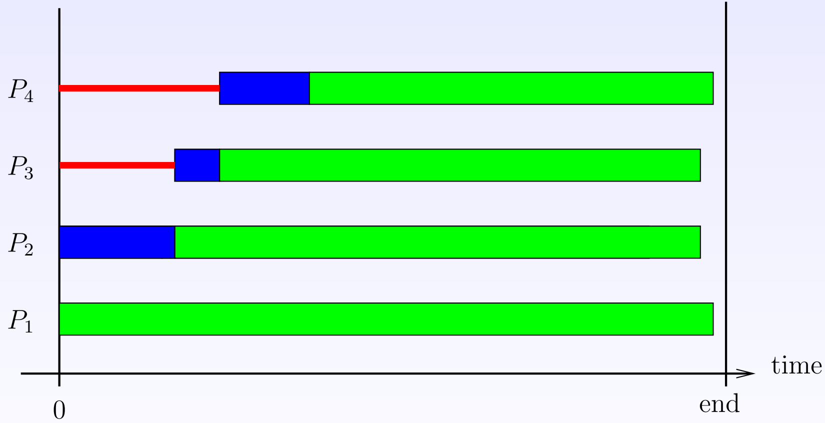
Two slaves i and $i+1$ with $T_i < T_{i+1}$



Increase α_i by ε

Proof of Lemma (1/2)

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



Communication time for following processors does not change

Proof of Lemma (1/2)

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

Get better solution!

Proof of Lemma (2/2)

- Ideally: $T'_i = T'_{i+1}$
  Choose ε such that:
  $(\alpha_i+\varepsilon)W_{\mathsf{total}}(c+w_i) = (\alpha_i+\varepsilon)W_{\mathsf{total}}c+(\alpha_{i+1}-\varepsilon)W_{\mathsf{total}}(c+w_{i+1})$
- If master finishes earlier: absurd
- If master finishes later: similar, decrease its load by ε

Resource selection

Lemma

In any optimal solution, all processors are enrolled

Proof: simple follow-up of previous Lemma

Resource selection

Lemma

In any optimal solution, all processors are enrolled

Proof: simple follow-up of previous Lemma

Compute the allocation

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

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

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

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

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

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

Compute the allocation

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

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

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

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

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

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

Compute the allocation

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

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

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

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

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

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

Compute the allocation

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

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

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

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

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

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

Compute the allocation

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

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

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

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

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

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

Compute the allocation

$T = \alpha_1 W_{\textsf{total}} w_1$

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

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

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

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

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

Impact of communication ordering

?

Impact of communication ordering ?

Load processed by P_i and P_{i+1} within time T

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

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

Hence $\alpha_{i+1} = \frac{1}{c+w_{i+1}}(\frac{T}{W_{\text{total}}} - \alpha_i c) = \frac{w_i}{(c+w_i)(c+w_{i+1})}\frac{T}{W_{\text{total}}}$

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

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

No Impact!

Impact of communication ordering ?

Load processed by P_i and P_{i+1} within time T

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

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

Hence $\alpha_{i+1} = \frac{1}{c+w_{i+1}}(\frac{T}{W_{\text{total}}} - \alpha_i c) = \frac{w_i}{(c+w_i)(c+w_{i+1})}\frac{T}{W_{\text{total}}}$

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

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

No Impact!

Impact of communication ordering ?

Load processed by P_i and P_{i+1} within time T

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

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

Hence $\alpha_{i+1} = \frac{1}{c+w_{i+1}}(\frac{T}{W_{\text{total}}} - \alpha_i c) = \frac{w_i}{(c+w_i)(c+w_{i+1})}\frac{T}{W_{\text{total}}}$

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

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

No Impact!

Impact of communication ordering ?

Load processed by P_i and P_{i+1} within time T

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

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

Hence $\alpha_{i+1} = \frac{1}{c+w_{i+1}}(\frac{T}{W_{\text{total}}} - \alpha_i c) = \frac{w_i}{(c+w_i)(c+w_{i+1})}\frac{T}{W_{\text{total}}}$

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

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

No Impact!

Impact of communication ordering ?

Load processed by P_i and P_{i+1} within time T

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

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

Hence $\alpha_{i+1} = \frac{1}{c+w_{i+1}}(\frac{T}{W_{\text{total}}} - \alpha_i c) = \frac{w_i}{(c+w_i)(c+w_{i+1})}\frac{T}{W_{\text{total}}}$

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

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

No Impact!

Impact of communication ordering ?

Load processed by P_i and P_{i+1} within time T

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

Processor P_{i+1}: $\alpha_i c W_{\text{total}} + \alpha_{i+1}(c+w_{i+1})W_{\text{total}} = T$

Hence $\alpha_{i+1} = \frac{1}{c+w_{i+1}}(\frac{T}{W_{\text{total}}} - \alpha_i c) = \frac{w_i}{(c+w_i)(c+w_{i+1})}\frac{T}{W_{\text{total}}}$

Processors P_i and P_{i+1}:

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

No Impact!

Choosing master processor

Compare processors P_1 and P_2

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

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

Total load processed:

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

Load maximal when $w_1 < w_2$
Master = fastest processor

Choosing master processor

Compare processors P_1 and P_2

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

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

Total load processed:

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

Load maximal when $w_1 < w_2$

Master = fastest processor

Choosing master processor

Compare processors P_1 and P_2

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

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

Total load processed:

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

Load maximal when $w_1 < w_2$

Master = fastest processor

Choosing master processor

Compare processors P_1 and P_2

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

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

Total load processed:

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

Load maximal when $w_1 < w_2$
Master = fastest processor

Choosing master processor

Compare processors P_1 and P_2

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

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

Total load processed:

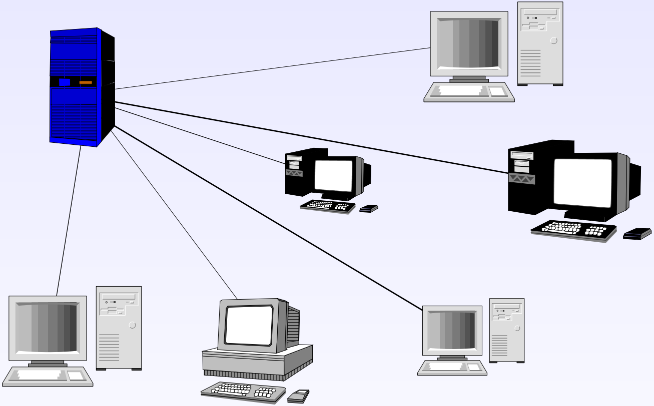
$$\alpha_1 + \alpha_2 = \frac{c + w_1 + w_2}{w_1(c + w_2)} = \frac{c + w_1 + w_2}{cw_1 + w_1 w_2}$$

Load maximal when $w_1 < w_2$
Master = fastest processor

Conclusion

- Closed-form formula for execution time and load distribution
- Choice of master
- Ordering of slaves not important
- All processors are enrolled
- Powerful approach! ☺

Star network



- Communication links between master and slaves have different bandwidths
- Slaves have different computing power

Notations

- Set P_1, ..., P_p of processors
- P_1 is the master, initially holds all data
- Total amount of work: W_{total}
- Processor P_i receives $\alpha_i W_{\text{total}}$ load units,
  where $\alpha_i W_{\text{total}} \in \mathbb{Q}$ and $\sum_i \alpha_i = 1$
  Time for one load unit on P_i: w_i
  Execution time on P_i: $\alpha_i w_i$
- **Communication time of one load-unit from P_1 to P_i: c_i**
  One-port model: P_1 serially sends one message to each slave

Notations

- Set P_1, ..., P_p of processors
- P_1 is the master, initially holds all data
- Total amount of work: W_{total}
- Processor P_i receives $\alpha_i W_{\text{total}}$ load units, where $\alpha_i W_{\text{total}} \in \mathbb{Q}$ and $\sum_i \alpha_i = 1$
  Time for one load unit on P_i: w_i
  Execution time on P_i: $\alpha_i w_i$
- **Communication time of one load-unit from P_1 to P_i: c_i**
  One-port model: P_1 serially sends one message to each slave

Notations

- Set P_1, ..., P_p of processors
- P_1 is the master, initially holds all data
- Total amount of work: W_{total}
- Processor P_i receives $\alpha_i W_{\text{total}}$ load units,
  where $\alpha_i W_{\text{total}} \in \mathbb{Q}$ and $\sum_i \alpha_i = 1$
  Time for one load unit on P_i: w_i
  Execution time on P_i: $\alpha_i w_i$
- **Communication time of one load-unit from P_1 to P_i: c_i**
  One-port model: P_1 serially sends one message to each slave

Notations

- Set P_1, ..., P_p of processors
- P_1 is the master, initially holds all data
- Total amount of work: W_{total}
- Processor P_i receives $\alpha_i W_{\text{total}}$ load units,
  where $\alpha_i W_{\text{total}} \in \mathbb{Q}$ and $\sum_i \alpha_i = 1$
  Time for one load unit on P_i: w_i
  Execution time on P_i: $\alpha_i w_i$
- **Communication time of one load-unit from P_1 to P_i: c_i**
  One-port model: P_1 serially sends one message to each slave

Notations

- Set P_1, ..., P_p of processors
- P_1 is the master, initially holds all data
- Total amount of work: W_{total}
- Processor P_i receives $\alpha_i W_{\text{total}}$ load units,
  where $\alpha_i W_{\text{total}} \in \mathbb{Q}$ and $\sum_i \alpha_i = 1$
  Time for one load unit on P_i: w_i
  Execution time on P_i: $\alpha_i w_i$
- **Communication time of one load-unit from P_1 to P_i: c_i**
  One-port model: P_1 serially sends one message to each slave

Impact of communication ordering

?

Impact of communication ordering

Load processed by P_i and P_{i+1} within time T

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

Processor P_{i+1}**:** $\alpha_i c_i W_{\text{total}} + \alpha_{i+1}(c_{i+1} + w_{i+1})W_{\text{total}} = T$

Hence $\alpha_{i+1} = \frac{1}{c_{i+1}+w_{i+1}}(1 - \frac{c_i}{c_i+w_i})\frac{T}{W_{\text{total}}} = \frac{w_i}{(c_i+w_i)(c_{i+1}+w_{i+1})}\frac{T}{W_{\text{total}}}$

Total load processed: $\alpha_i + \alpha_{i+1} = \frac{c_{i+1}+w_i+w_{i+1}}{(c_i+w_i)(c_{i+1}+w_{i+1})}$

Communication time: $\alpha_i c_i + \alpha_{i+1} c_{i+1} = \frac{c_i c_{i+1} + c_{i+1} w_i + c_i w_{i+1}}{(c_i+w_i)(c_{i+1}+w_{i+1})}$

$\rightsquigarrow$ **Serve processors with higher bandwidth first**

Impact of communication ordering

Load processed by P_i and P_{i+1} within time T

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

Processor P_{i+1}**:** $\alpha_i c_i W_{\text{total}} + \alpha_{i+1}(c_{i+1} + w_{i+1})W_{\text{total}} = T$

Hence $\alpha_{i+1} = \frac{1}{c_{i+1}+w_{i+1}}(1 - \frac{c_i}{c_i+w_i})\frac{T}{W_{\text{total}}} = \frac{w_i}{(c_i+w_i)(c_{i+1}+w_{i+1})}\frac{T}{W_{\text{total}}}$

Total load processed: $\alpha_i + \alpha_{i+1} = \frac{c_{i+1}+w_i+w_{i+1}}{(c_i+w_i)(c_{i+1}+w_{i+1})}$

Communication time: $\alpha_i c_i + \alpha_{i+1} c_{i+1} = \frac{c_i c_{i+1} + c_{i+1} w_i + c_i w_{i+1}}{(c_i+w_i)(c_{i+1}+w_{i+1})}$

$\leadsto$ **Serve processors with higher bandwidth first**

Impact of communication ordering

Load processed by P_i and P_{i+1} within time T

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

Processor P_{i+1}: $\alpha_i c_i W_{\text{total}} + \alpha_{i+1}(c_{i+1} + w_{i+1})W_{\text{total}} = T$

Hence $\alpha_{i+1} = \frac{1}{c_{i+1}+w_{i+1}}(1 - \frac{c_i}{c_i+w_i})\frac{T}{W_{\text{total}}} = \frac{w_i}{(c_i+w_i)(c_{i+1}+w_{i+1})}\frac{T}{W_{\text{total}}}$

Total load processed: $\alpha_i + \alpha_{i+1} = \frac{c_{i+1}+w_i+w_{i+1}}{(c_i+w_i)(c_{i+1}+w_{i+1})}$

Communication time: $\alpha_i c_i + \alpha_{i+1} c_{i+1} = \frac{c_i c_{i+1}+c_{i+1}w_i+c_i w_{i+1}}{(c_i+w_i)(c_{i+1}+w_{i+1})}$

$\rightsquigarrow$ **Serve processors with higher bandwidth first**

Impact of communication ordering

Load processed by P_i and P_{i+1} within time T

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

Processor P_{i+1}**:** $\alpha_i c_i W_{\text{total}} + \alpha_{i+1}(c_{i+1} + w_{i+1})W_{\text{total}} = T$

Hence $\alpha_{i+1} = \frac{1}{c_{i+1}+w_{i+1}}(1 - \frac{c_i}{c_i+w_i})\frac{T}{W_{\text{total}}} = \frac{w_i}{(c_i+w_i)(c_{i+1}+w_{i+1})}\frac{T}{W_{\text{total}}}$

Total load processed: $\alpha_i + \alpha_{i+1} = \frac{c_{i+1}+w_i+w_{i+1}}{(c_i+w_i)(c_{i+1}+w_{i+1})}$

Communication time: $\alpha_i c_i + \alpha_{i+1} c_{i+1} = \frac{c_i c_{i+1} + c_{i+1} w_i + c_i w_{i+1}}{(c_i+w_i)(c_{i+1}+w_{i+1})}$

$\rightsquigarrow$ **Serve processors with higher bandwidth first**

Impact of communication ordering

Load processed by P_i and P_{i+1} within time T

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

Processor P_{i+1}**:** $\alpha_i c_i W_{\text{total}} + \alpha_{i+1}(c_{i+1} + w_{i+1})W_{\text{total}} = T$

Hence $\alpha_{i+1} = \frac{1}{c_{i+1}+w_{i+1}}(1 - \frac{c_i}{c_i+w_i})\frac{T}{W_{\text{total}}} = \frac{w_i}{(c_i+w_i)(c_{i+1}+w_{i+1})}\frac{T}{W_{\text{total}}}$

Total load processed: $\alpha_i + \alpha_{i+1} = \frac{c_{i+1}+w_i+w_{i+1}}{(c_i+w_i)(c_{i+1}+w_{i+1})}$

Communication time: $\alpha_i c_i + \alpha_{i+1} c_{i+1} = \frac{c_i c_{i+1}+c_{i+1}w_i+c_i w_{i+1}}{(c_i+w_i)(c_{i+1}+w_{i+1})}$

$\leadsto$ **Serve processors with higher bandwidth first**

Impact of communication ordering

Load processed by P_i and P_{i+1} within time T

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

Processor P_{i+1}**:** $\alpha_i c_i W_{\text{total}} + \alpha_{i+1}(c_{i+1} + w_{i+1})W_{\text{total}} = T$

Hence $\alpha_{i+1} = \frac{1}{c_{i+1}+w_{i+1}}(1 - \frac{c_i}{c_i+w_i})\frac{T}{W_{\text{total}}} = \frac{w_i}{(c_i+w_i)(c_{i+1}+w_{i+1})}\frac{T}{W_{\text{total}}}$

Total load processed: $\alpha_i + \alpha_{i+1} = \frac{c_{i+1}+w_i+w_{i+1}}{(c_i+w_i)(c_{i+1}+w_{i+1})}$

Communication time: $\alpha_i c_i + \alpha_{i+1} c_{i+1} = \frac{c_i c_{i+1}+c_{i+1}w_i+c_i w_{i+1}}{(c_i+w_i)(c_{i+1}+w_{i+1})}$

$\rightsquigarrow$ **Serve processors with higher bandwidth first**

Impact of communication ordering

Load processed by P_i and P_{i+1} within time T

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

Processor P_{i+1}**:** $\alpha_i c_i W_{\text{total}} + \alpha_{i+1}(c_{i+1} + w_{i+1})W_{\text{total}} = T$

Hence $\alpha_{i+1} = \frac{1}{c_{i+1}+w_{i+1}}(1 - \frac{c_i}{c_i+w_i})\frac{T}{W_{\text{total}}} = \frac{w_i}{(c_i+w_i)(c_{i+1}+w_{i+1})}\frac{T}{W_{\text{total}}}$

Total load processed: $\alpha_i + \alpha_{i+1} = \frac{c_{i+1}+w_i+w_{i+1}}{(c_i+w_i)(c_{i+1}+w_{i+1})}$

Communication time: $\alpha_i c_i + \alpha_{i+1} c_{i+1} = \frac{c_i c_{i+1} + c_{i+1} w_i + c_i w_{i+1}}{(c_i+w_i)(c_{i+1}+w_{i+1})}$

$\rightsquigarrow$ **Serve processors with higher bandwidth first**

Resource selection

Lemma

In any optimal solution, all processors are enrolled

Proof:

- Consider an optimal solution
- Let P_k be a non-participating processor
- Enroll P_k in the end and assign load α_k s.t.
- $\alpha_k(c_k + w_k)W_{\text{total}} =$ execution time of last activated processor

Resource selection

Lemma

In any optimal solution, all processors are enrolled

Proof:

- Consider an optimal solution
- Let P_k be a non-participating processor
- Enroll P_k in the end and assign load α_k s.t.
- $\alpha_k(c_k + w_k)W_{\mathsf{total}} =$ execution time of last activated processor

Load balancing property

Lemma

In any optimal solution, all processors terminate execution simultaneously

Proof

- Most existing proofs are incorrect
- Either resort to very technical arguments (properties of solutions of linear programs) or to tedious case-by-case analysis

Load balancing property

Lemma

In any optimal solution, all processors terminate execution simultaneously

Proof

- Most existing proofs are incorrect
- Either resort to very technical arguments (properties of solutions of linear programs) or to tedious case-by-case analysis

Conclusion

- Closed-form formula for execution time and load distribution
- Serve faster-communicating processors first
- All processors terminate execution simultaneously
- All processors are enrolled
- Powerful approach! ☺

Various extensions

Good news One-round, linear model extends to other topologies (e.g. trees, linear chains)

Still open Adding return messages, although very natural, renders problem quite combinatorial

Bad news

- Becomes NP-hard when adding communication/computation latencies
- Unfortunately, adding latencies absolutely needed to deal with multi-round algorithms

Bibliography

- Pioneering book:
  Scheduling divisible loads in parallel and distributed systems, V. Bharadwaj, D. Ghose, V. Mani and T.G. Robertazzi, IEEE Computer Society Press 1996
- Recent survey:
  Ten reasons to use Divisible Load Theory, T.G. Robertazzi, Computer 36, 5 (2003), 63-68
- Bags of tasks:
  Optimal sharing of bags of tasks in heterogeneous clusters, M. Adler, Y. Gong and A.L. Rosenberg, 15th ACM SPAA (2003), 1-10
- Archive of DLS literature:
  http://www.ece.sunysb.edu/~tom/dlt.html

Outline

Outline