

Olivier BEAUMONT, LaBRI

Arnaud LEGRAND, Yves ROBERT, ENS Lyon

LIP laboratory at Ecole Normale Supérieure de Lyon

www.ens-lyon.fr/LIP

Static scheduling strategies for heterogeneous systems

ReMaP project: jointly operated by CNRS, ENS Lyon and INRIA

Algorithm design, scheduling and load balancing for heterogeneous platforms

Recent publications

Heterogeneous linear algebra

- Matrix multiplication on heterogeneous platforms
IEEE Trans. Parallel Distributed Systems 12 (2001), 1033-1051
- A proposal for a heterogeneous cluster ScaLAPACK (dense linear solvers)
IEEE Trans. Computers 10 (2001), 1052-1070
- Dense linear algebra kernels on heterogeneous platforms: redistribution issues
Parallel Computing 28 (2002), 155-185

Master-slave tasking

- The master-slave paradigm with heterogeneous processors
Cluster'2001, IEEE Computer Society Press
- Bandwidth-centric allocation of independent tasks on heterogeneous platforms
IPDPS'2002, IEEE Computer Society Press
- Scheduling strategies for master-slave tasking on heterogeneous processor grids
PARA'02, LNCS Springer Verlag

Outline

- 1 Minimum makespan scheduling
- 2 Divisible load scheduling
- 3 Steady-state scheduling
- 4 Limitations of static scheduling

Outline

- 1 Minimum makespan scheduling
- 2 Divisible load scheduling
- 3 Steady-state scheduling
- 4 Limitations of static scheduling

Outline

- 1 Minimum makespan scheduling
- 2 Divisible load scheduling
- 3 Steady-state scheduling
- 4 Limitations of static scheduling

Outline

- 1 Minimum makespan scheduling
- 2 Divisible load scheduling
- 3 Steady-state scheduling
- 4 Limitations of static scheduling

Minimum makespan scheduling

Framework

- Efficient scheduling of application tasks critical to achieving high performance
- Scheduling and load-balancing already difficult for homogeneous resources

Questions

1. Impact of heterogeneity on the design and analysis of scheduling heuristics?
2. Limitations of static scheduling on large-scale distributed platforms?

Standard macro-dataflow model

1. Dependence graph DAG $\mathcal{G} = (V, E, w, c)$

- node-weighted: task durations $w(T), T \in V$
- edge-weighted: communication volume $c(T, T'), e : T \rightarrow T' \in E$
- tasks are atomic and cannot be preempted

2. Computing resources $\mathcal{P} = (P, t, \text{link})$

- t_i cycle-time of processor P_i
- $\text{link}(i, j)$ speed of link between P_i and P_j
- computation-communication overlap
- 😞 multi-port: no bound on the number of simultaneous communications involving a processor at a given time-step

Scheduling objective

Minimize makespan, i.e.

$$\max_{v \in V} (\sigma(v) + w(v) \times t_{\text{alloc}(v)}).$$

NP-complete problem, even with

- simple fork-join dependence graph,
- infinite number of identical resources ($t_i = 1$),
- fully homogeneous communication network ($\text{link}(i, j) = 1$).

Heuristics

Several heuristics targeted to heterogeneous resources

Among others PCT, BIL, CPOP, GDL.

HEFT, a natural extension of list heuristics:

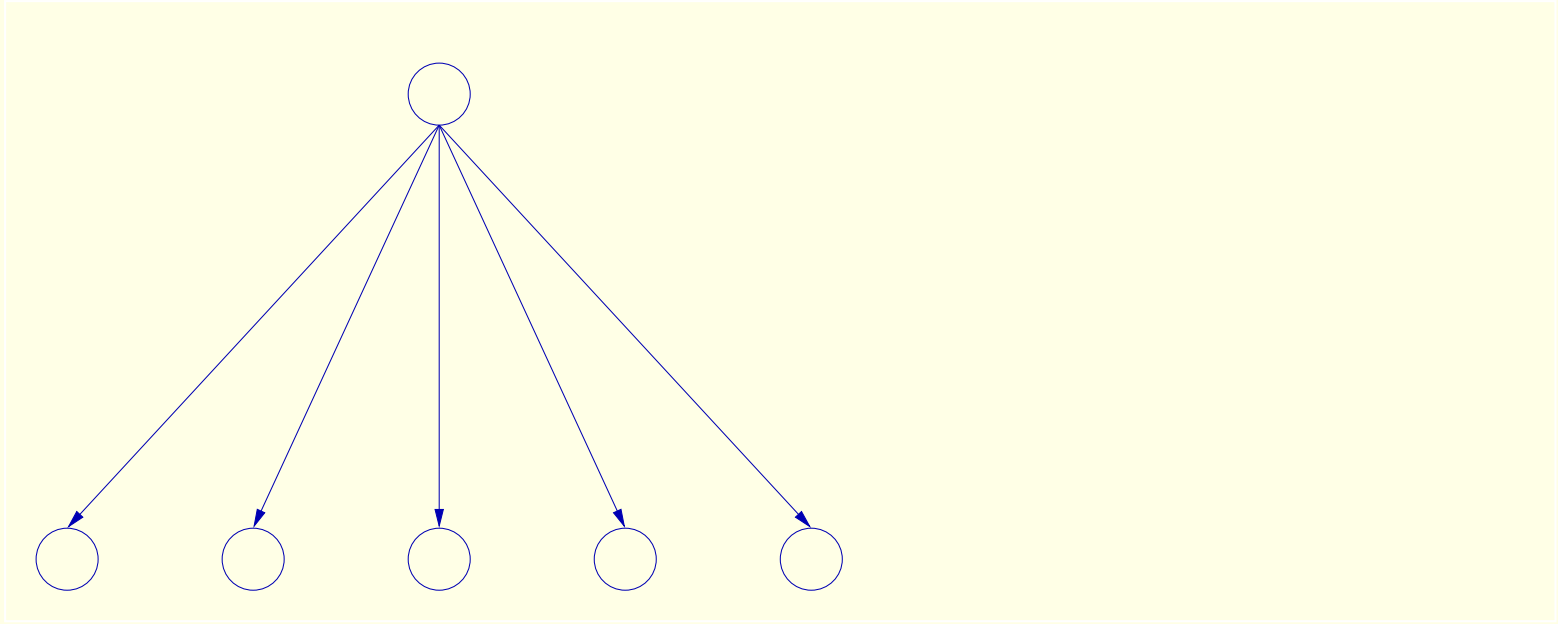
- Compute bottom level for each task (longest path to final node)
- Assign priorities according to non-increasing bottom levels (intuitively, the longer the path, the more urgent the task)
- Select ready task of maximum priority
- Assign to processor which allows for earliest termination

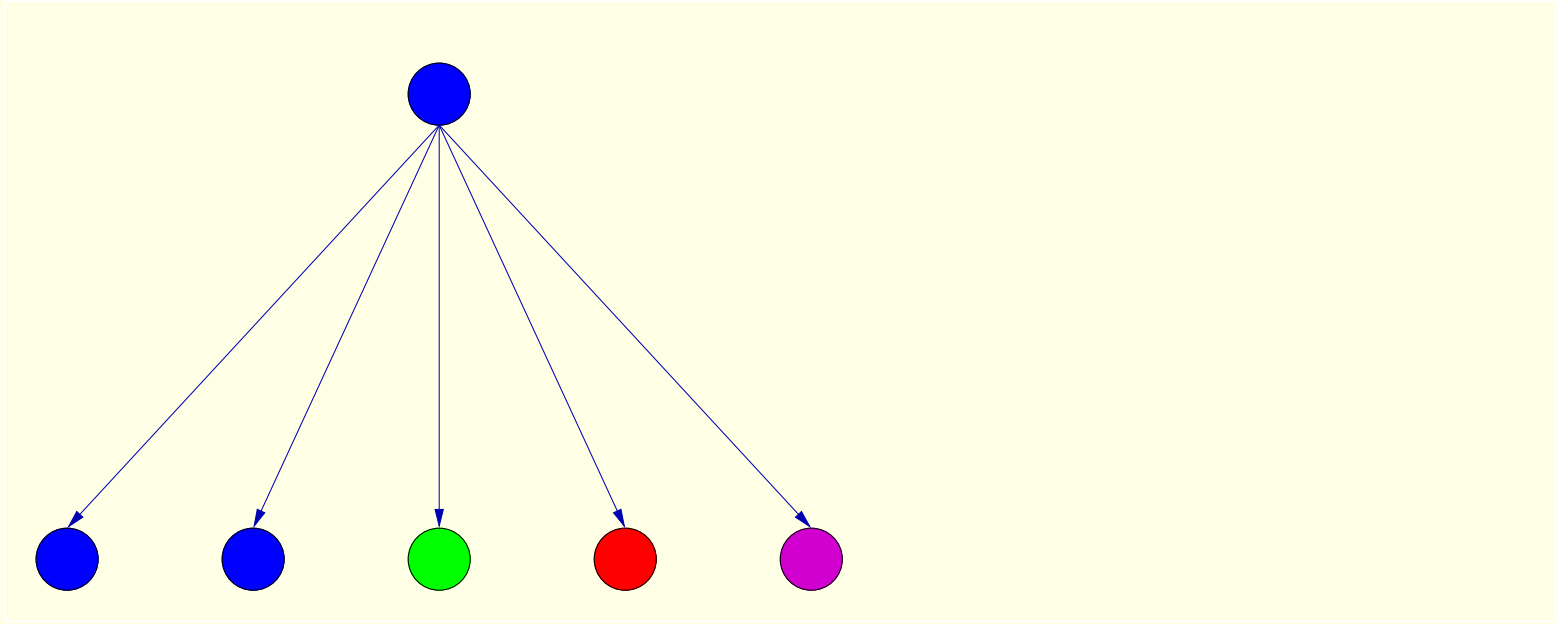
One-port model

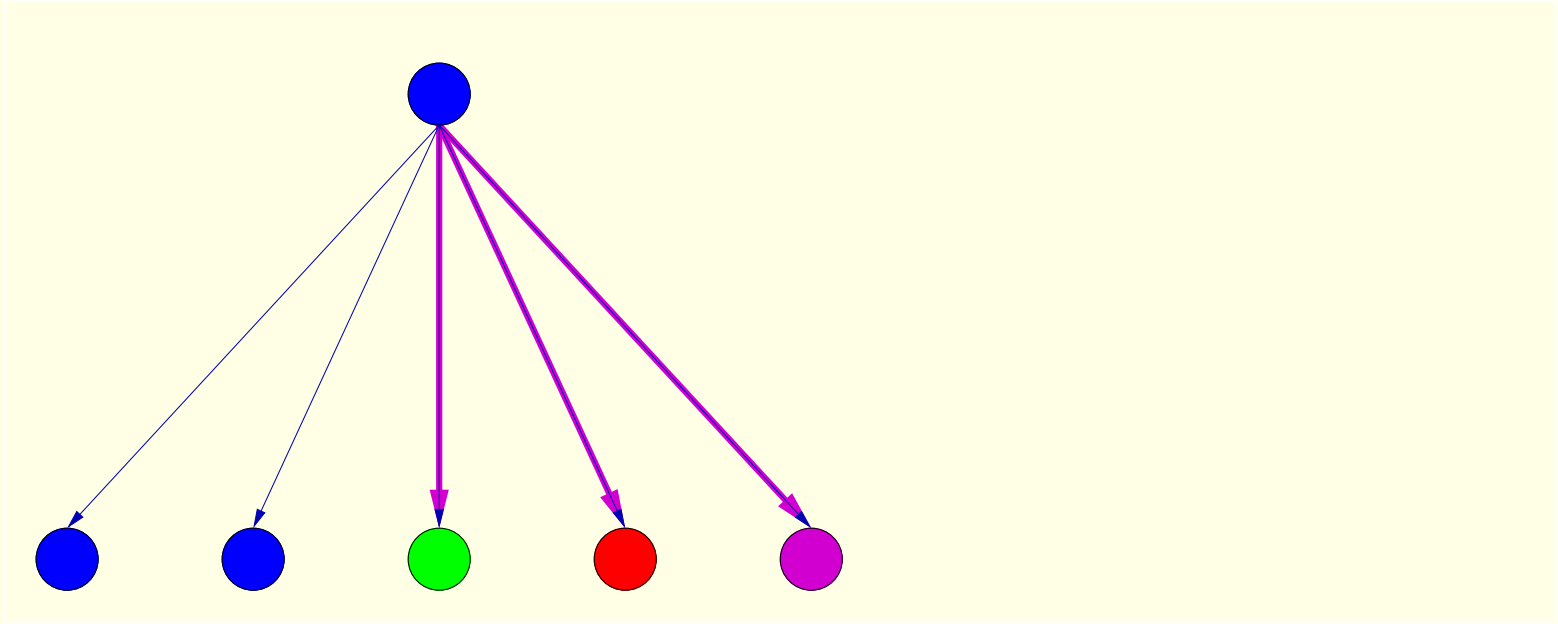
At a given time-step, a processor can communicate (send and/or receive) with at most another processor

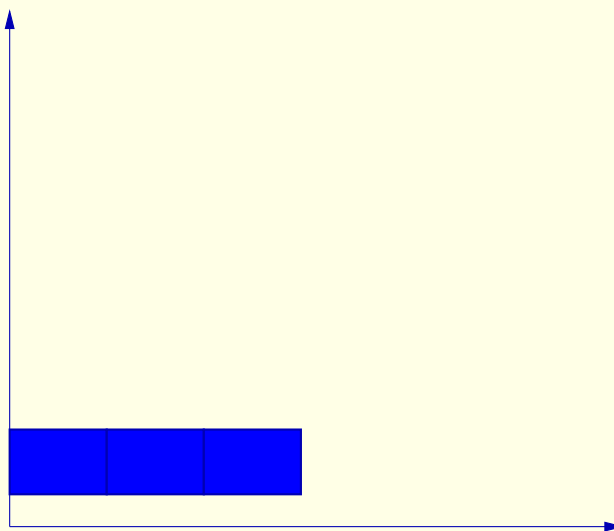
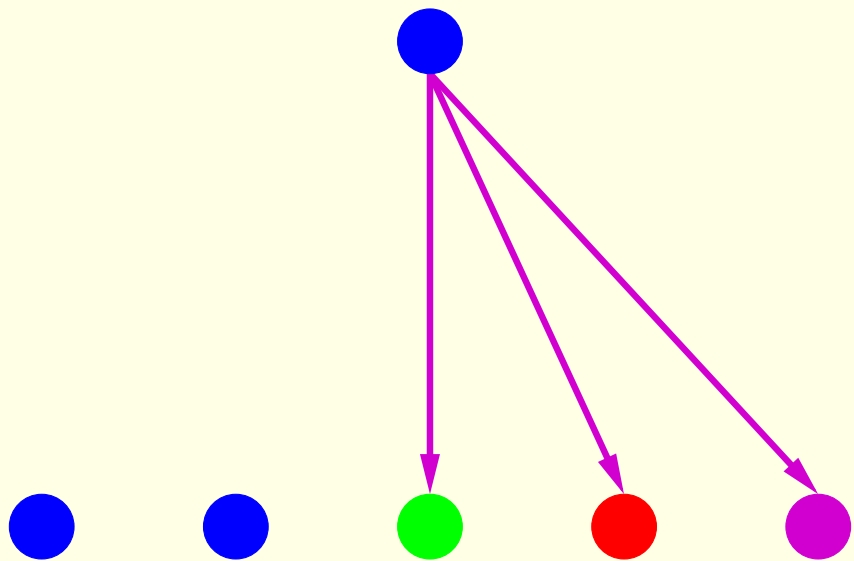
At a given time-step, any communication permutation may be implemented, but no more

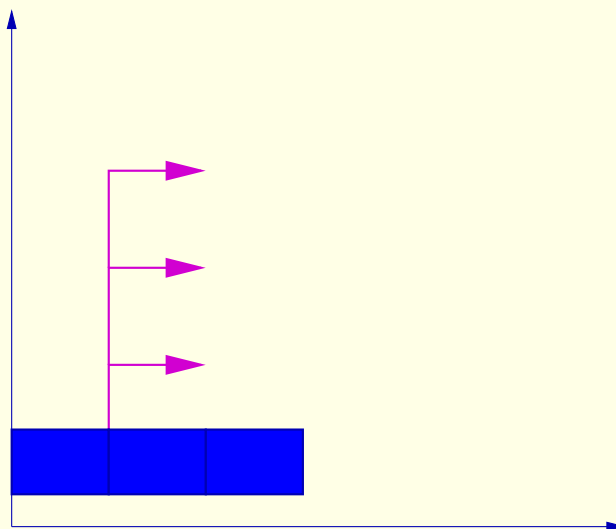
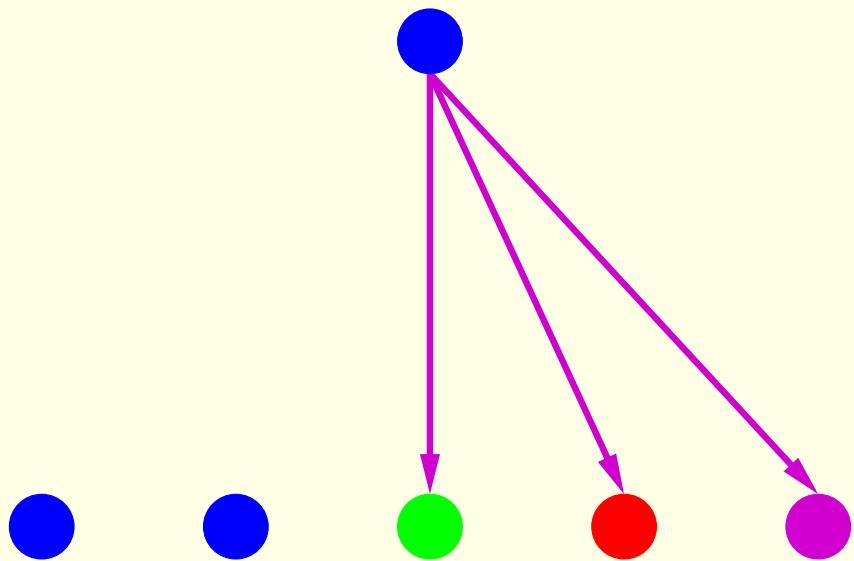
- Nicely models switches like Myrinet or multiplexed bus architectures
- Bi-directional one-port model close to actual capabilities of modern processors
- Communication-aware scheduling from the literature: Sinnen and Sousa, Hollermann et al., Hsu et al., Tan et al., Orduna et al., and Roig et al

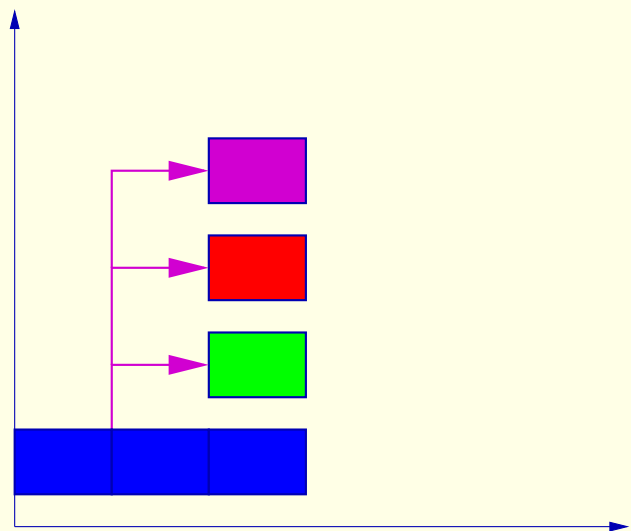
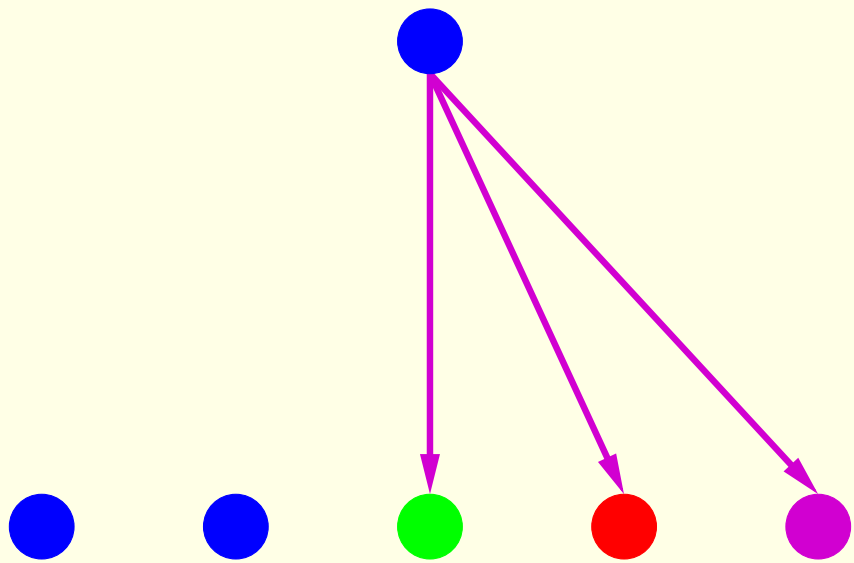


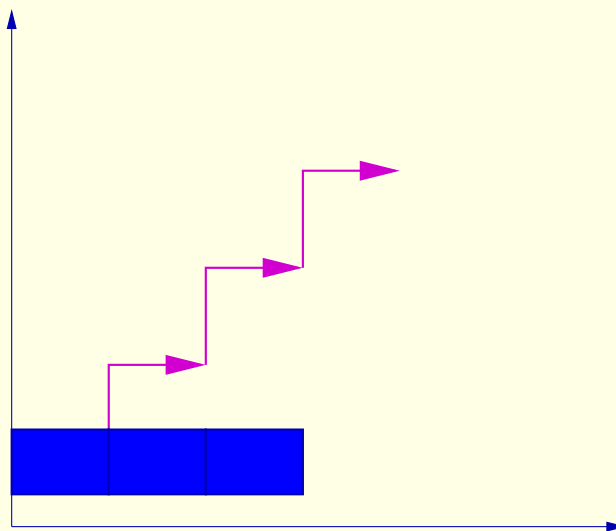
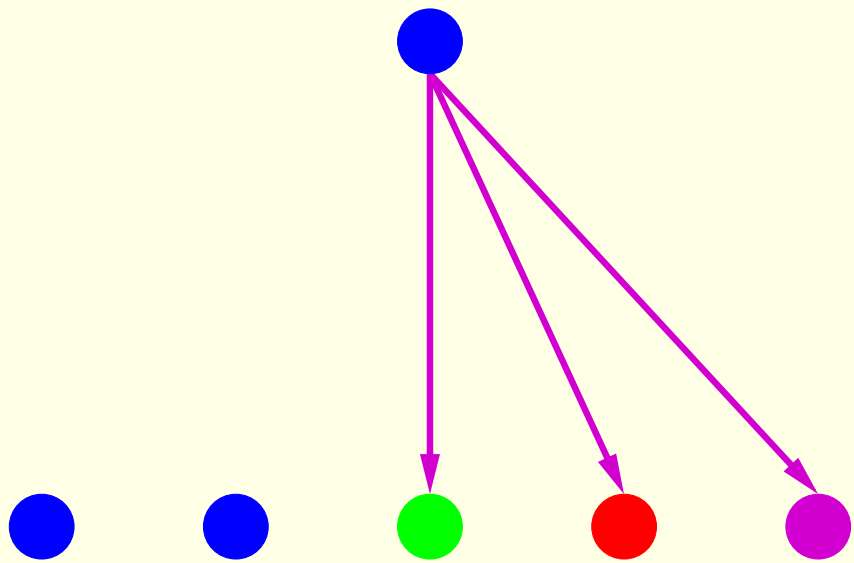


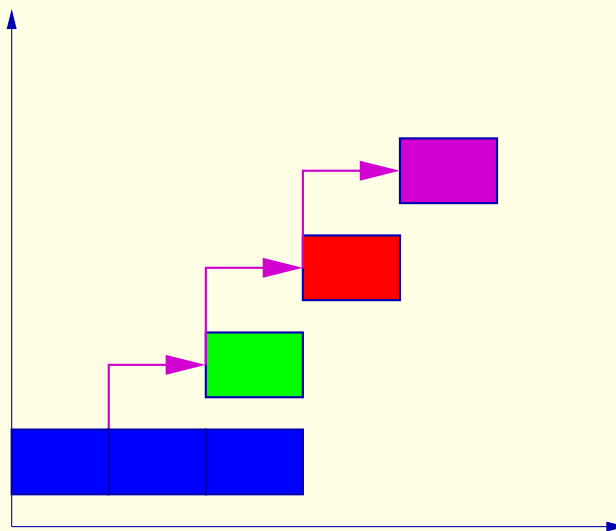
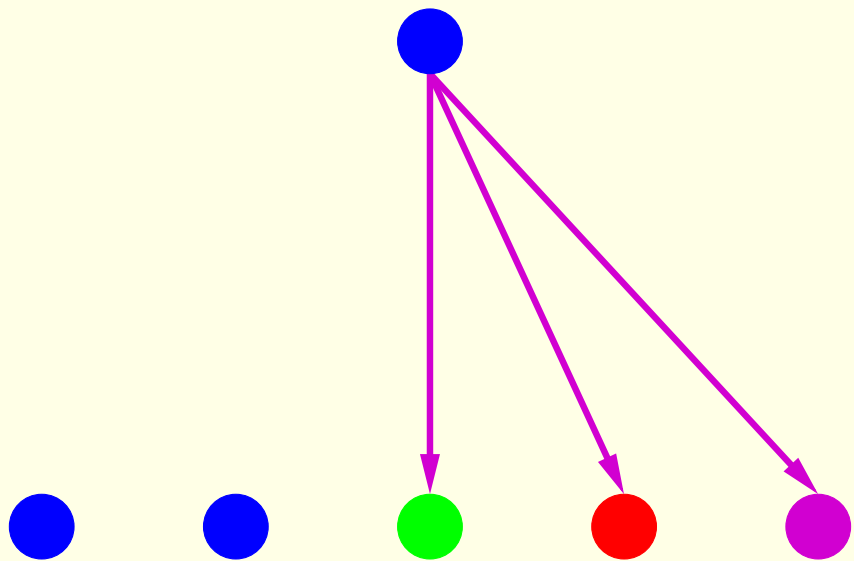












Complexity result

Theorem 1. *NP-completeness of the minimum makespan problem, even with*

- *simple fork dependence graph,*
- *infinite number of identical resources ($t_i = 1$),*
- *fully homogeneous communication network ($link(i, j) = 1$).*

 Even more difficult than standard macro-dataflow model!

 HEFT straightforwardly extended to the one-port model

New Heuristic

ILHA to handle several ready tasks simultaneously:

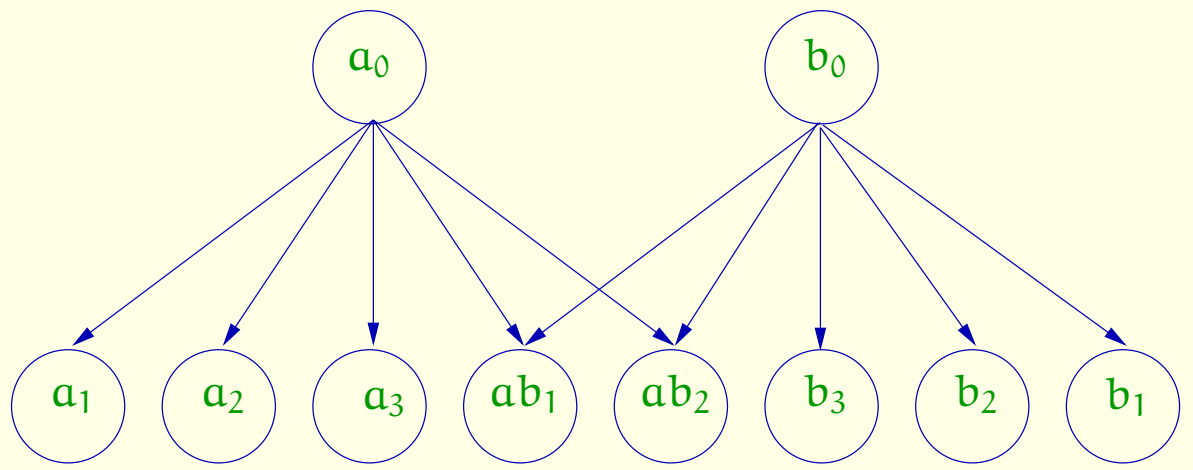
- At each step, consider several (independent) ready tasks
- Globally load-balance workload
- Allocate so as to minimize the number of communications

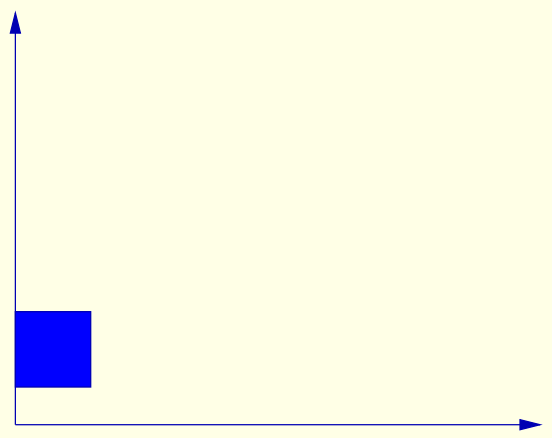
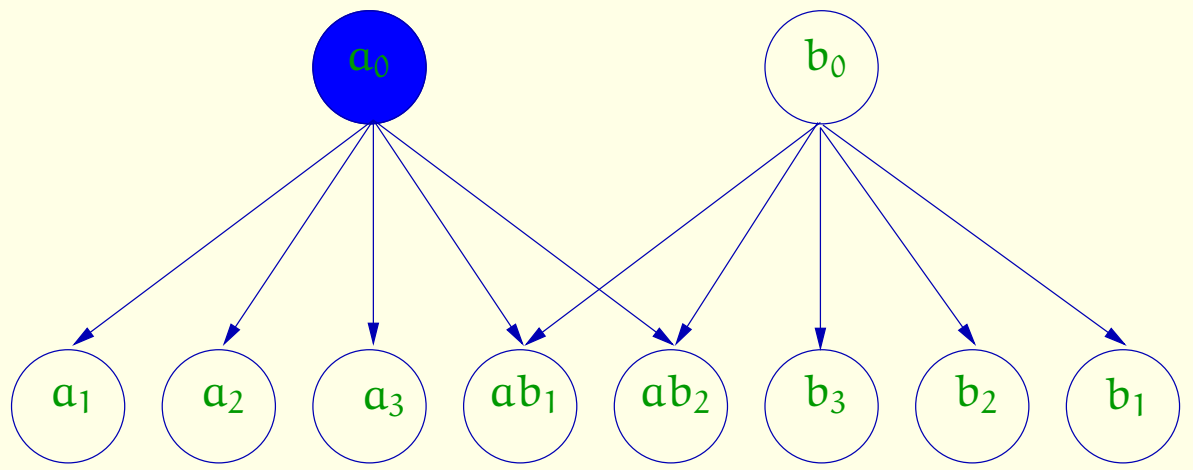
First phase

- Load-balancing algorithm to compute maximum load for each processor
- For each of the several tasks under consideration, if all predecessors already assigned to the same processor: assign to that processor if not yet saturated
- Otherwise proceed to next task

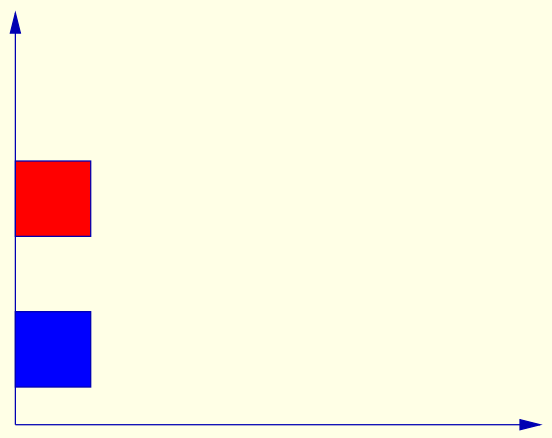
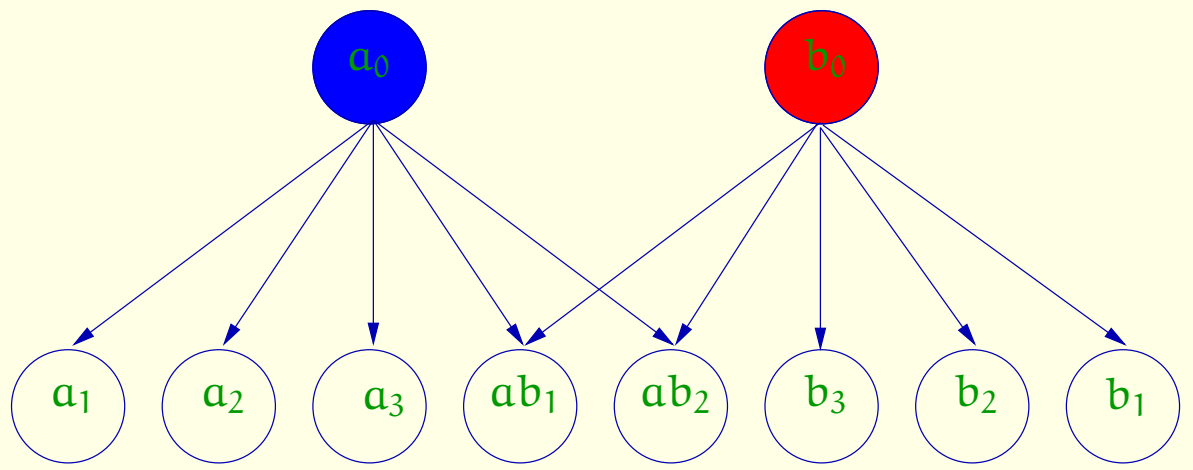
Second phase

Each remaining task assigned to the processor that allows for earliest completion time (similar to HEFT)

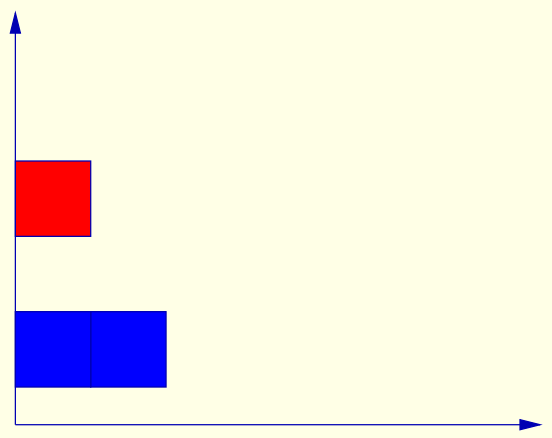
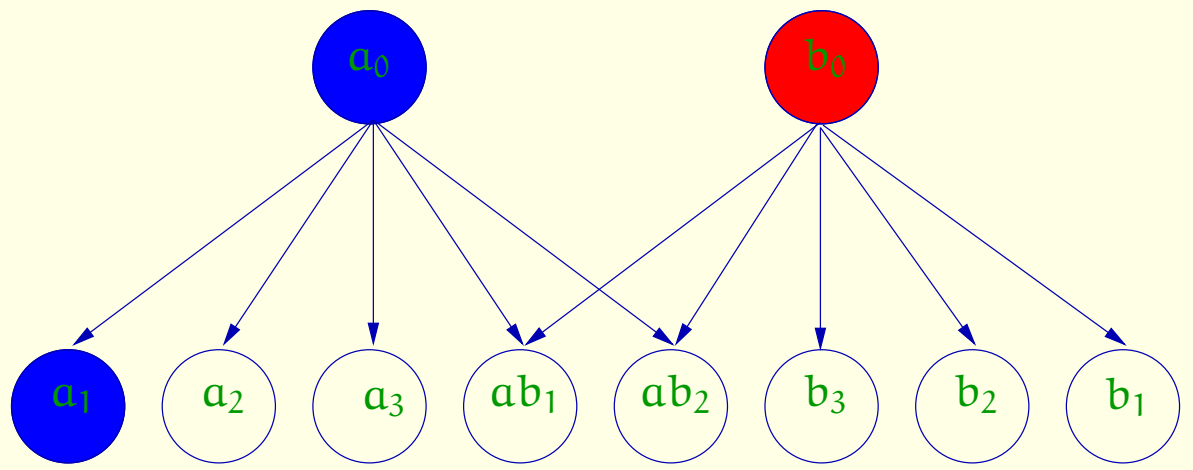




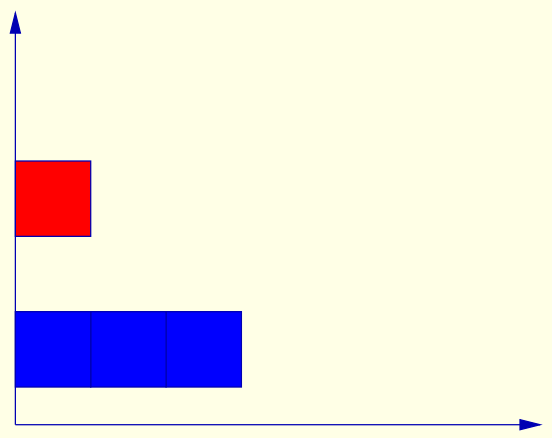
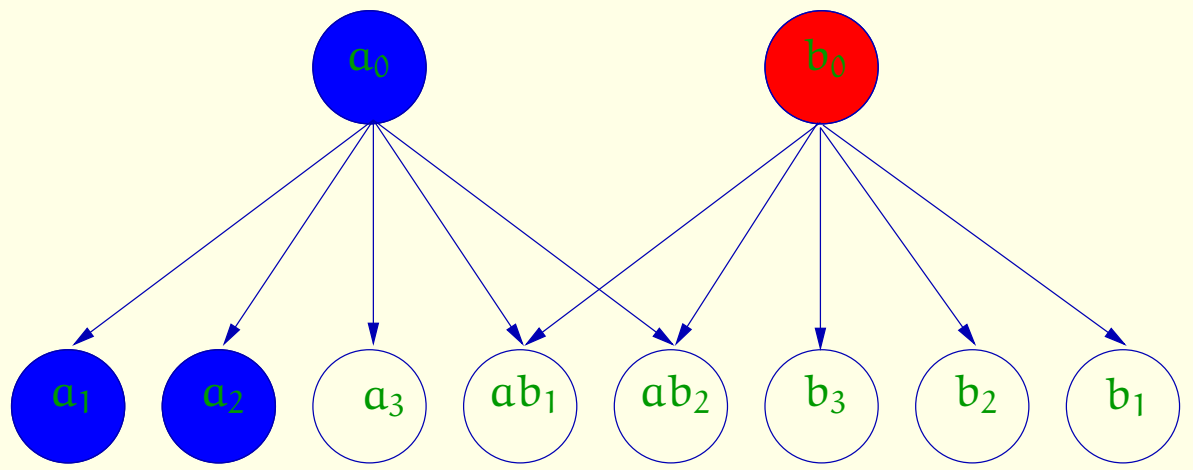
HEFT



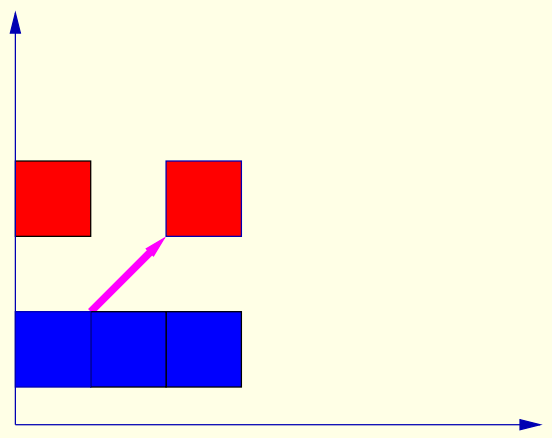
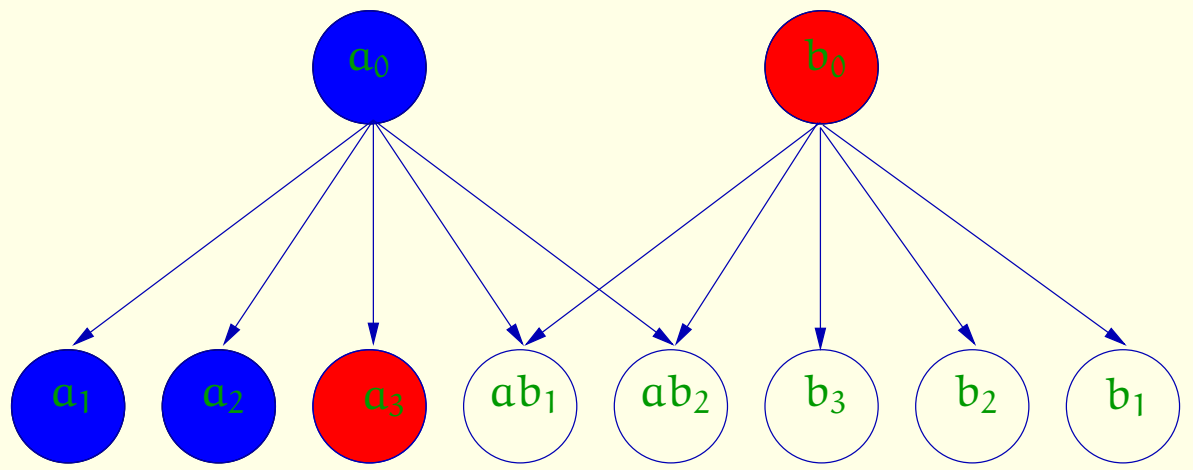
HEFT



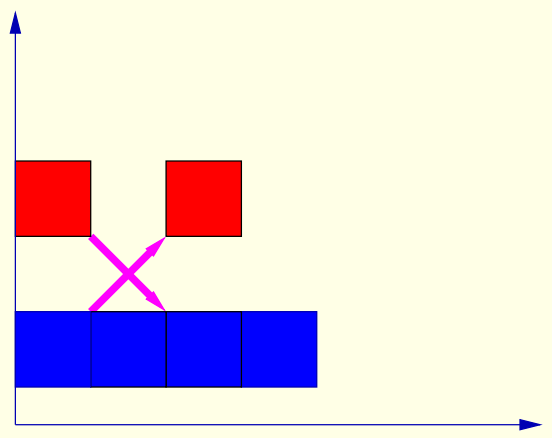
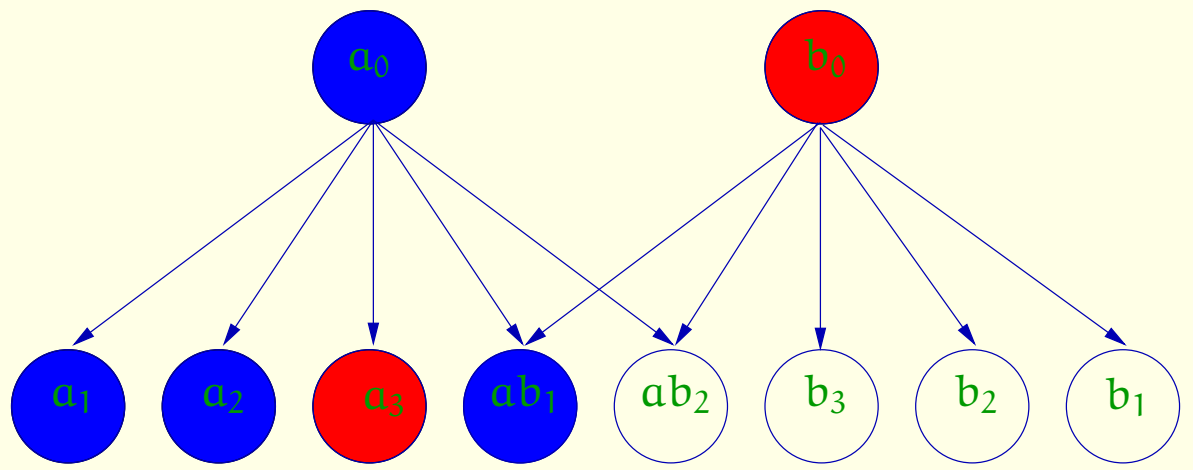
HEFT



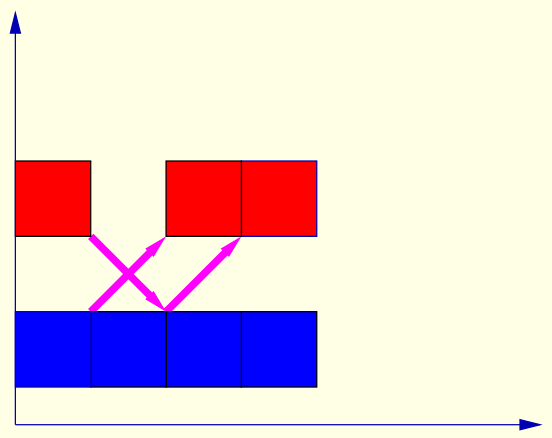
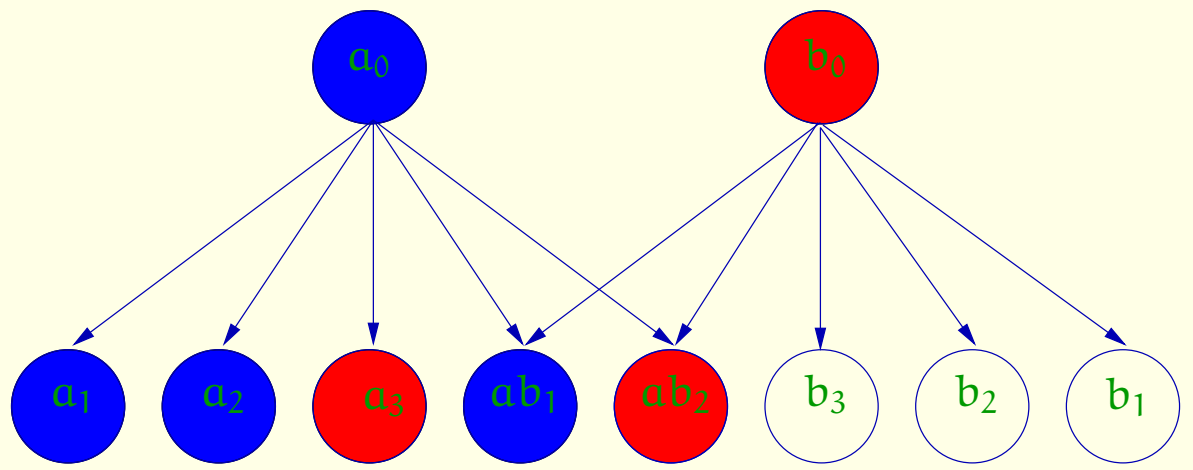
HEFT



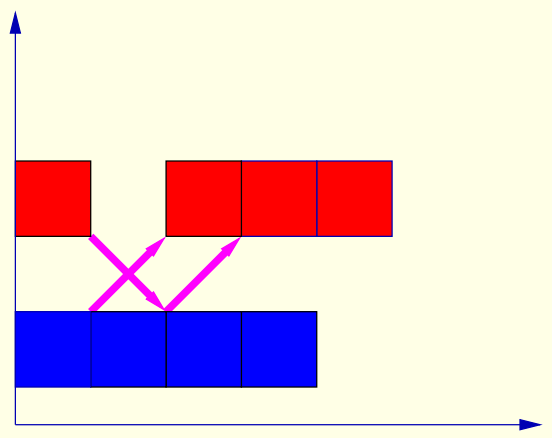
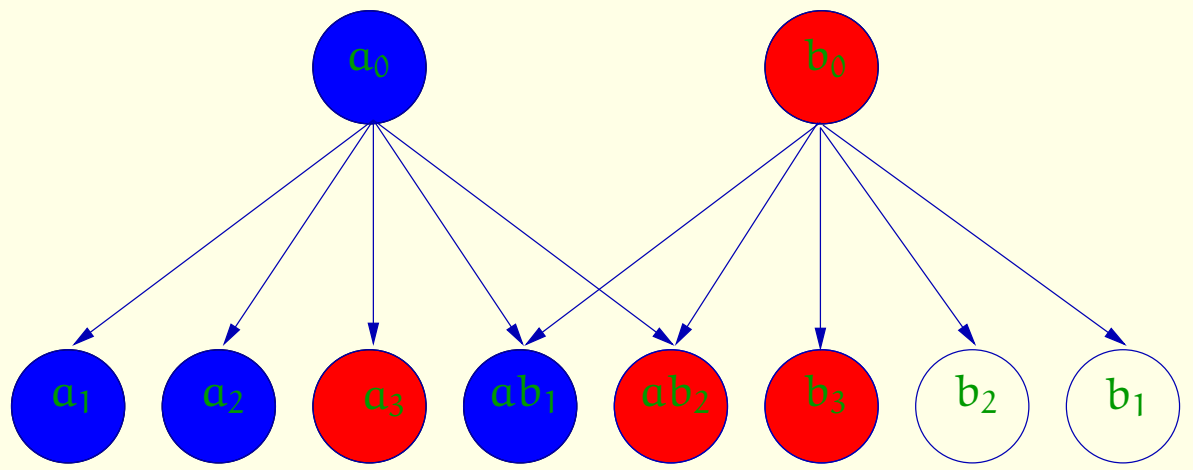
HEFT



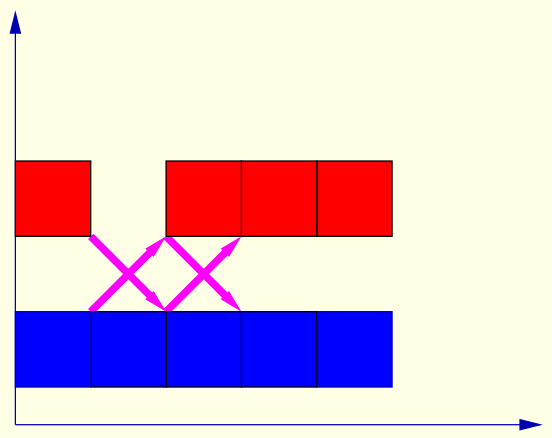
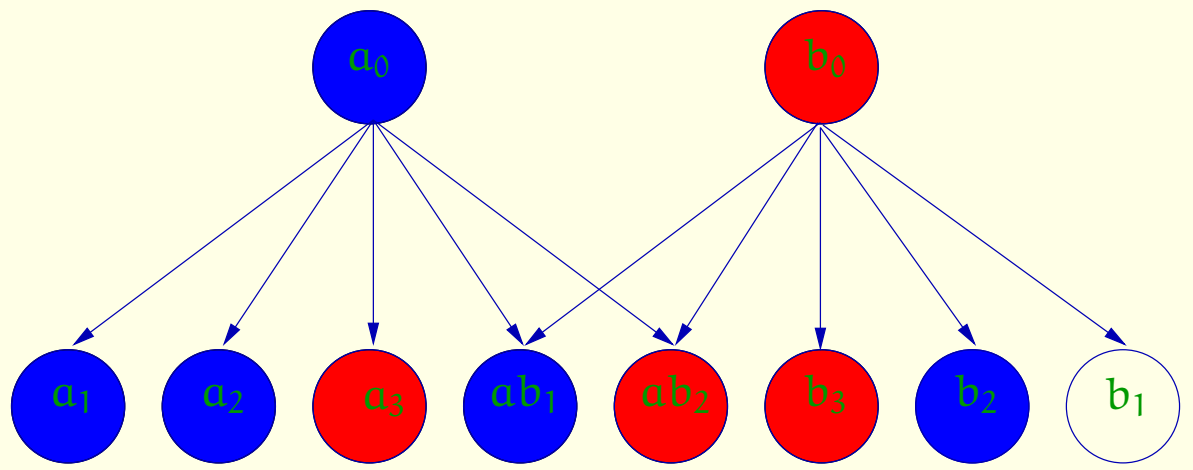
HEFT



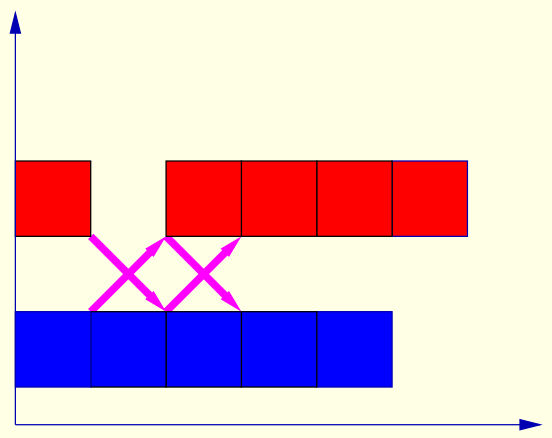
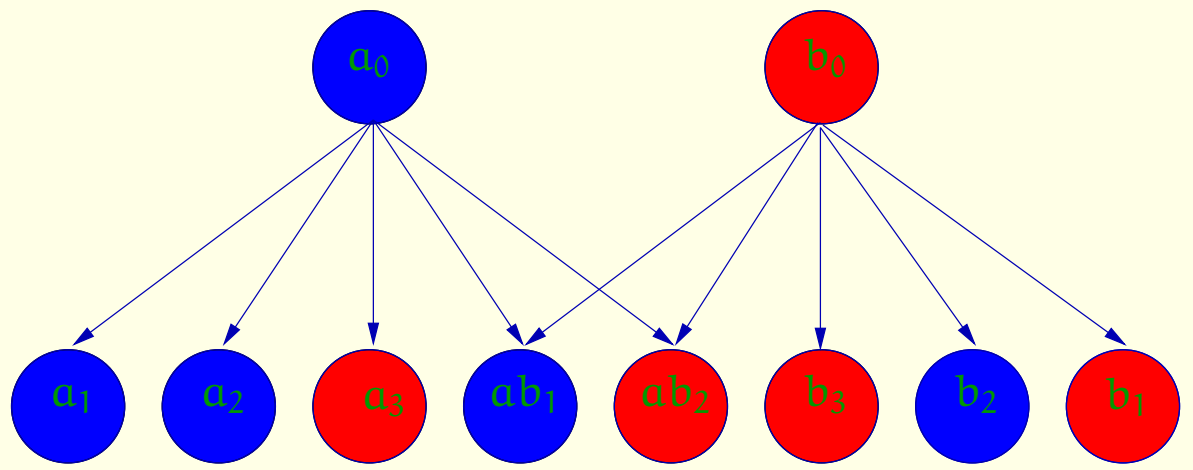
HEFT



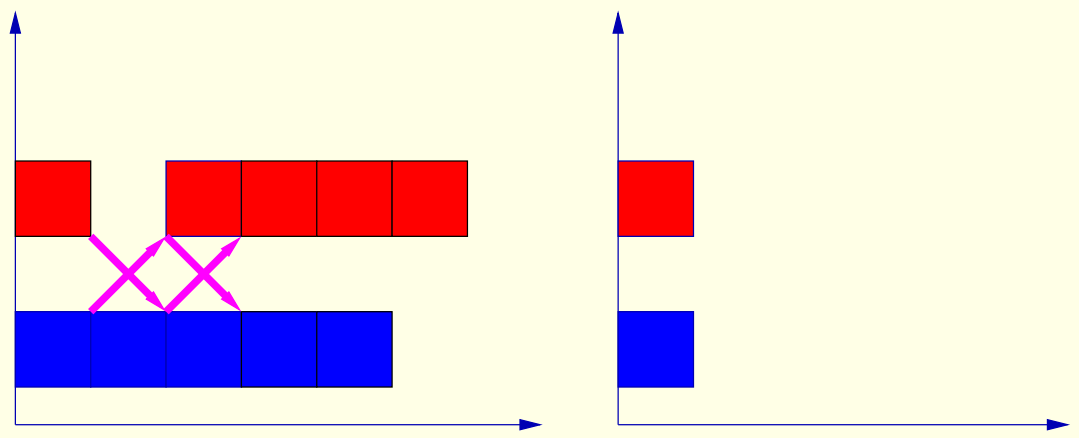
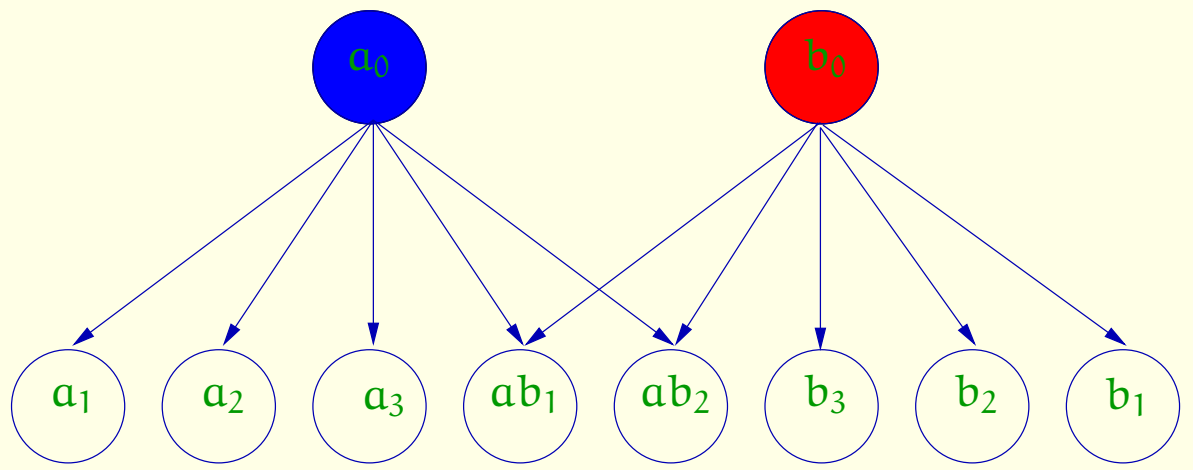
HEFT



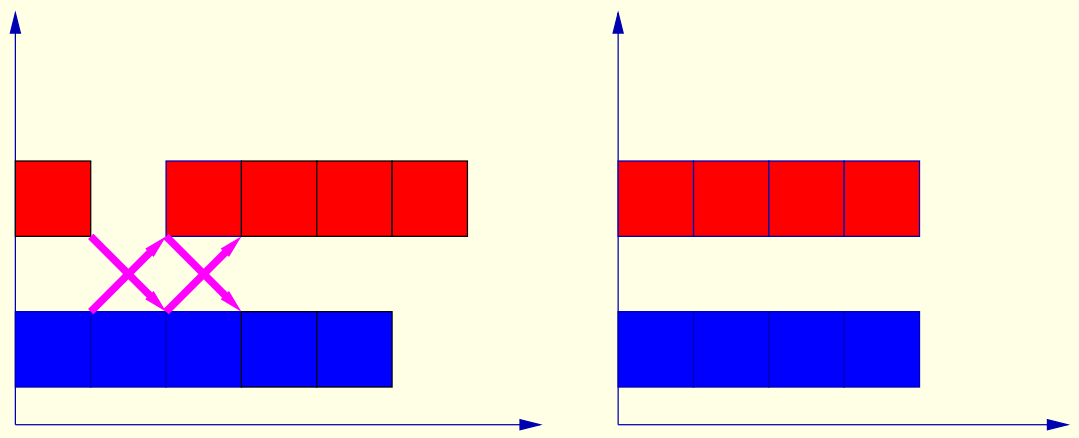
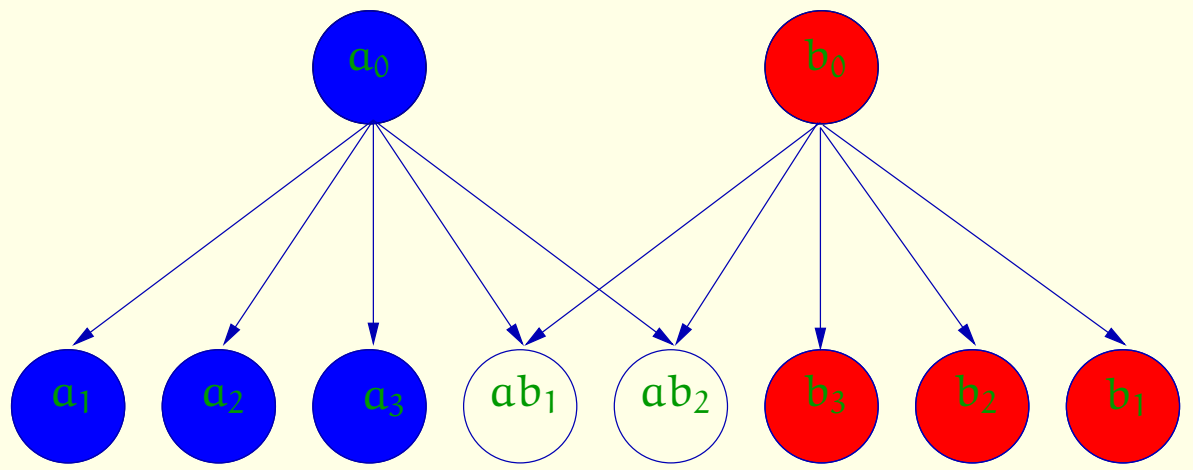
HEFT



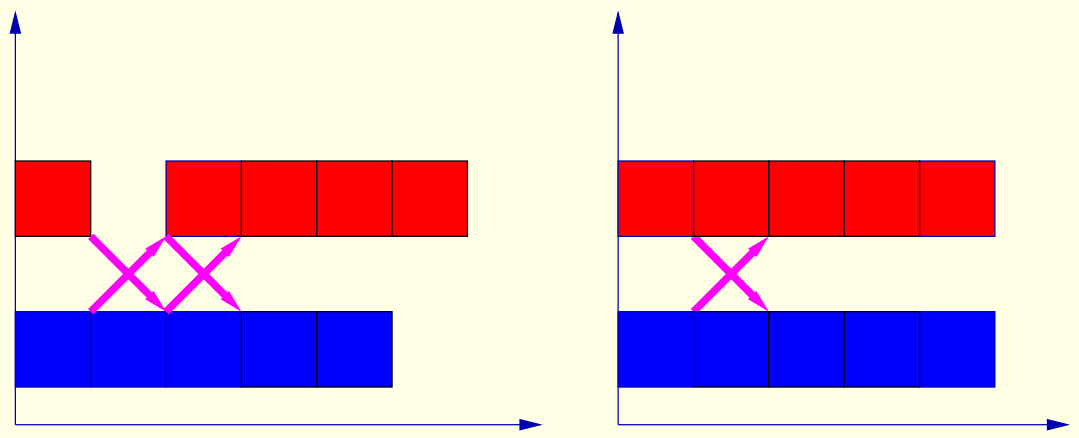
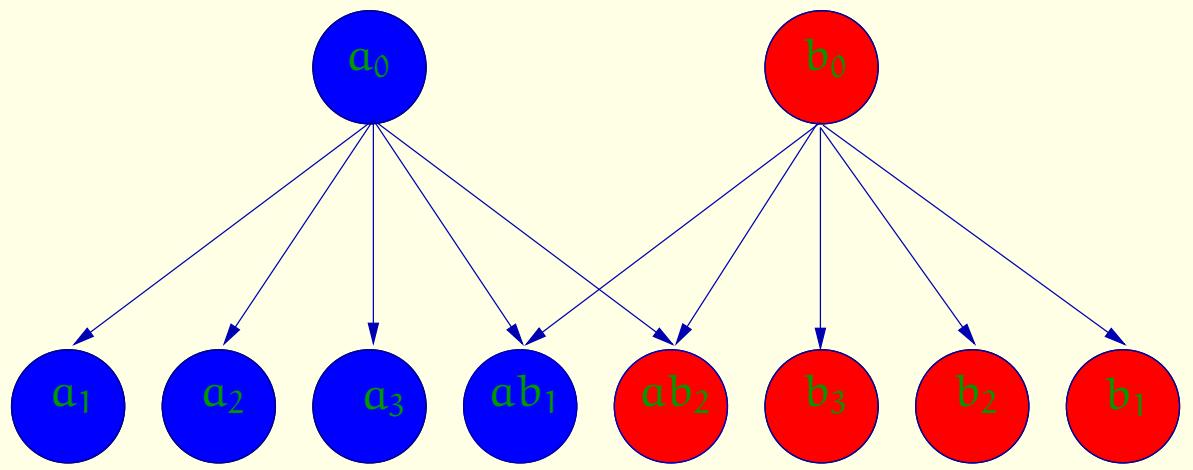
HEFT



ILHA



ILHA



ILHA

Summary

Models

- Realistic (bi-directional) one-port scheduling model
- Scarcity of communication resources fully taken into account
- Intrinsic complexity of task graph scheduling under this model

ILHA heuristic

- Search for a better load-balance + generation of fewer communications
- Scheduling a chunk of ready tasks simultaneously
⇒ global view of potential communications

Problems

- Not easy to extend to hierarchical architectures
- Accurate knowledge of the task graph?
- Precise estimation of task and communication weights?

Divisible load scheduling

Framework (Robertazzi and others)

A divisible job can be arbitrarily split in a linear fashion among any number of processors \Rightarrow perfectly parallel job

Notations

- α_i fraction of workload assigned to P_i ($\sum_i \alpha_i = 1$)
- $\alpha_i w_i$ time for P_i to process its load fraction
- $\alpha_i c_i$ time to transmit to P_i its load fraction
- T_i time elapsed before P_i begins its processing
 $\Rightarrow T_f = \max_i (T_i + \alpha_i w_i)$ overall computational time

Problems

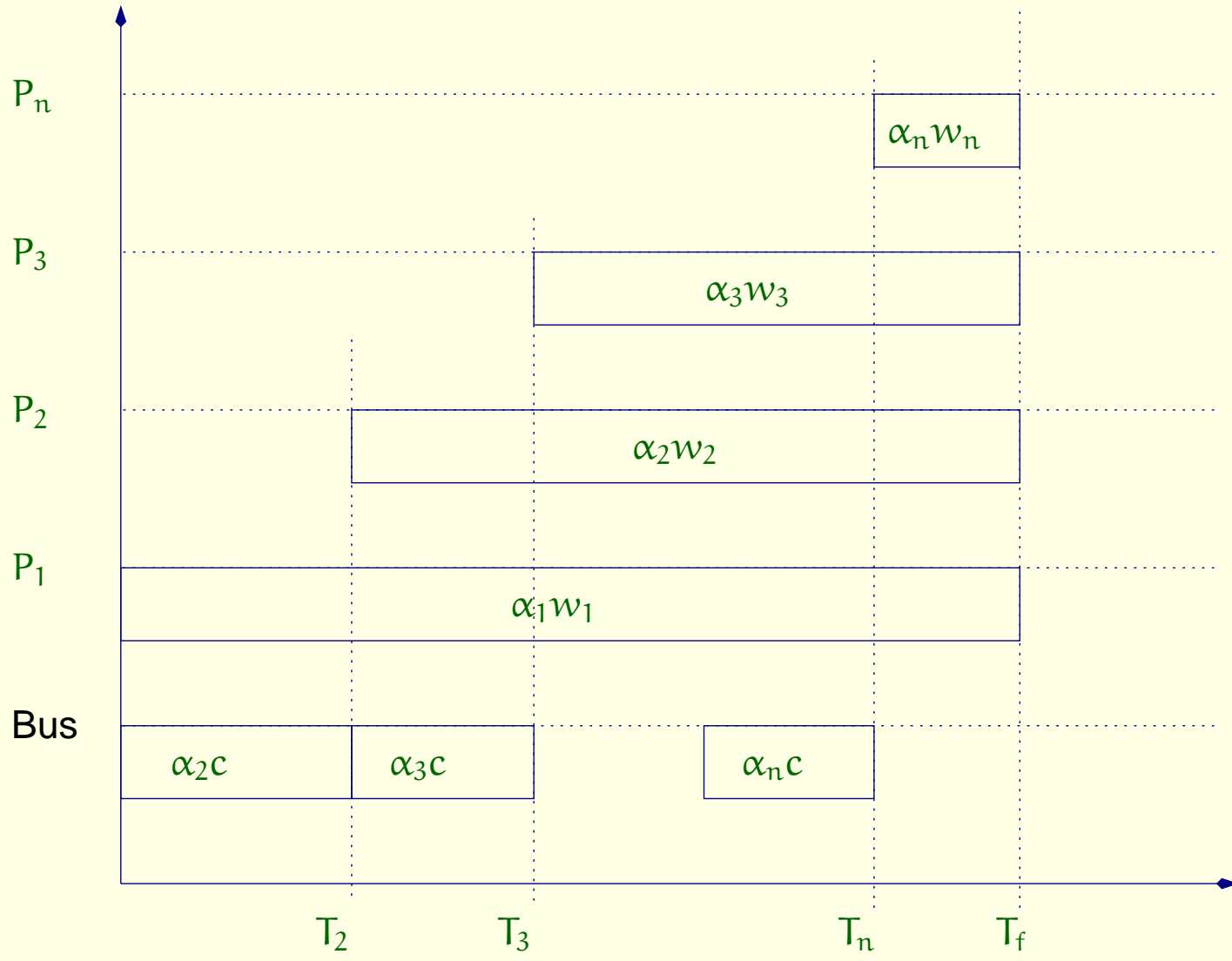
- Ordering of communications
- Fraction of load for each processor

Case of a bus

- Only one communication at a given time step
- $c_i = c, \forall i$

Solution

- All processors finish their work at the same time
- Closed-form solution
- All orderings optimal



Case of a fork

- Closed-form solution for given ordering
- Optimal solution not known
- 😞 Disappointing!

Summary

Several open problems despite the simple model

Steady-state scheduling

Framework

- Deal with large problems on large-scale distributed platforms
- Absolute minimization of total execution time not really required
- Asymptotically optimal schedules by problem relaxation:
 1. Neglect initialization and clean-up phases
 2. Derive optimal steady-state scheduling using linear programming
 3. Prove asymptotic optimality of the associated schedule

Packet routing (Bertsimas and Gamarnik)

- $G = (V, E)$ non-oriented graph modeling the target architectural platform
- Set of same-size packets to be routed through the network.
- Each packet characterized by a source node and a destination node
- Need one unit of time to circulate a packet on any edge, but at most one packet can circulate on one edge at a given time-step
- For each (v_k, v_l) in G , n_{kl} number of packets to be routed from v_k to v_l , and $\mathcal{P} = \{(k, l) \in V^2, n_{kl} \neq 0\}$
- Scheduling algorithm asymptotically optimal when $n = \sum_{(k,l) \in \mathcal{P}} n_{kl} \rightarrow +\infty$
- Remove temporal constraints: don't write that a packet must have reached a node before leaving it
- x_{ij}^{kl} number of packets circulating from v_k to v_l and using the edge between v_i and v_j , $\forall (i, j) \in E, \forall (k, l) \in \mathcal{P}$

Relaxed linear program

MINIMIZE C_{\max} ,

SUBJECT TO

$$\left\{ \begin{array}{ll} (1) \sum_{i, (k,i) \in E} x_{ki}^{kl} = n_{kl} & \forall (k, l) \in \mathcal{P} \\ (2) \sum_{i, (i,l) \in E} x_{il}^{kl} = n_{kl} & \forall (k, l) \in \mathcal{P} \\ (3) \sum_{j, (j,i) \in E} x_{ji}^{kl} = \sum_{r, (i,r) \in E} x_{ir}^{kl} & \forall (k, l) \in \mathcal{P}, \forall i \neq k, l \\ (4) C_{i,j} = \sum_{(k,l) \in \mathcal{P}} x_{ij}^{kl} & \forall (i, j) \in E \\ (5) C_{i,j} \leq C_{\max}, & \forall (i, j) \in E \\ (6) x_{ij}^{kl} \geq 0, \quad C_{i,j} \geq 0, & \forall (k, l) \in \mathcal{P}, (i, j) \in E \end{array} \right.$$

Linear program with $O(|E||\mathcal{P}|)$ rational variables and $O(|V||\mathcal{P}| + |E|)$ constraints
 \Rightarrow solution in polynomial time, independent of the total number of packets n

Actual scheduling

- Split the execution into phases
- Reproduce a “rounded” version of the relaxed solution during each phases

Algorithm

Input Compute the optimal value C_{\max} from relaxed linear program

Step 1 During each phase $[l\Omega, (l+1)\Omega]$, where $l = 0, \dots, \lceil \frac{C_{\max}}{\Omega} \rceil - 1$, and for each edge $(i, j) \in E$, circulate on the edge as many packets of type (k, l) as available in node i at time $l\Omega$, but no more than $a_{ij}^{kl} = \lfloor \frac{x_{ij}^{kl} \Omega}{C_{\max}} \rfloor$, the number of packets (rounded from below) of type (k, l) which circulate on the edge (i, j) during Ω time-steps in the relaxed problem

Step 2 At time-step $T = \lceil \frac{C_{\max}}{\Omega} \rceil \Omega$, all the packets that have not been fully routed are handled sequentially

Choose Ω of the order of $\sqrt{C_{\max}}$, then schedule makespan is $C_{\max} + O(\sqrt{C_{\max}})$, asymptotically optimal

Mixed task-data parallelism

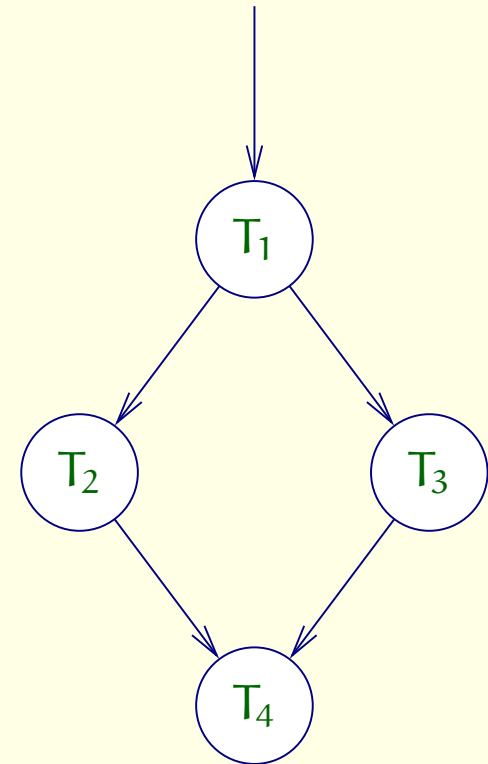
Pipelined execution of task graphs

- Complex application = suite of identical, independent problems to be solved
- In turn, each problem consists of a set of tasks
- Dependences (precedence constraints) between these tasks
- Typical example: repeated execution of the same algorithm on several distinct data samples

How to execute such a complex application on a heterogeneous "grid" computing platform?

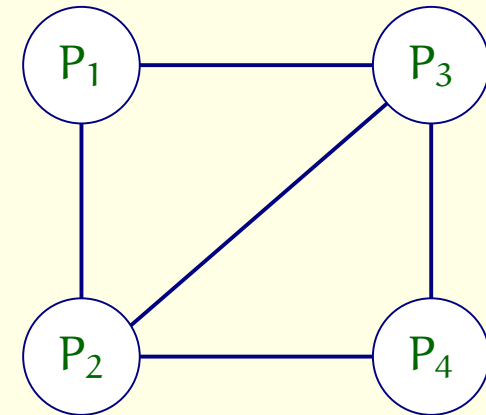
Application → task graph

- Let $\mathcal{P}^{(1)}, \mathcal{P}^{(2)}, \dots, \mathcal{P}^{(n)}$ be the n problems to solve, where n is large
- Problem $\mathcal{P}^{(m)}$ corresponds to a copy $G^{(m)} = (V^{(m)}, E^{(m)})$ of the same task graph (V, E)
- The number $|V|$ of nodes in V is the number of task **types**
Overall, there are $n \cdot |V|$ tasks to process



Architecture \rightarrow platform graph

- p processor nodes P_1, P_2, \dots, P_p
- Edge $e_{ij} : P_i \rightarrow P_j$ labeled by c_{ij} = time to transfer a message of unit length between P_i and P_j , in either direction
 $c_{ij} = +\infty$ is no physical link \rightarrow (virtually) complete graph
- full overlap, single-port operation mode



Application/network parameters

Execution times

- Processor P_i requires $w_{i,k}$ time units to process a task of type T_k .
- Simple case: $w_{i,k} = w_i \times \delta_k$:
 w_i relative speed of processor P_i ,
 δ_k weight of task T_k

Communication times

- Each edge $e_{k,l} : T_k \rightarrow T_l$ in the task graph is weighted by a communication cost $data_{k,l}$ (data output by T_k and required as input to T_l)
- If task $T_k^{(m)}$ is processed on P_i and task $T_l^{(m)}$ is processed on P_j , the time to transfer the data from P_i to P_j is $data_{k,l} \times c_{i,j}$

Steady-state equations → definitions

- For each edge $e_{k,l} : T_k \rightarrow T_l$ and for each processor pair (P_i, P_j) :
 $s(P_i \rightarrow P_j, e_{k,l}) =$ (average) fraction of time spent each time-unit by P_i to send to P_j data involved by the edge $e_{k,l}$
- $sent(P_i \rightarrow P_j, e_{k,l}) =$ (fractional) number of such files sent per time-unit:

$$s(P_i \rightarrow P_j, e_{k,l}) = sent(P_i \rightarrow P_j, e_{k,l}) \times (data_{k,l} \times c_{i,j}) \quad (1)$$

- For each task type $T_k \in V$ and for each processor P_i ::
 $\alpha(P_i, T_k) =$ (average) fraction of time spent each time-unit by P_i to process tasks of type T_k
 $consumed(P_i, T_k) =$ (fractional) number of tasks of type T_k processed per time unit by processor P_i :

$$\alpha(P_i, T_k) = consumed(P_i, T_k) \times w_{i,k} \quad (2)$$

Steady-state equations → operation

Activities during one time-unit

$$\forall P_i, \forall T_k \in V, 0 \leq \alpha(P_i, T_k) \leq 1 \quad (3)$$

$$\forall P_i, P_j, \forall e_{k,l} \in E, 0 \leq s(P_i \rightarrow P_j, e_{k,l}) \leq 1 \quad (4)$$

One-port model for outgoing communications

$$\forall P_i, \sum_{P_j \in n(P_i)} \sum_{e_{k,l} \in E} s(P_i \rightarrow P_j, e_{k,l}) \leq 1 \quad (5)$$

One-port model for incoming communications

$$\forall P_i, \sum_{P_j \in n(P_i)} \sum_{e_{k,l} \in E} s(P_j \rightarrow P_i, e_{k,l}) \leq 1 \quad (6)$$

Full overlap

$$\forall P_i, \sum_{T_k \in V} \alpha(P_i, T_k) \leq 1 \quad (7)$$

Steady-state equations → input/output

T_{begin} connected to every task with no predecessor in the graph

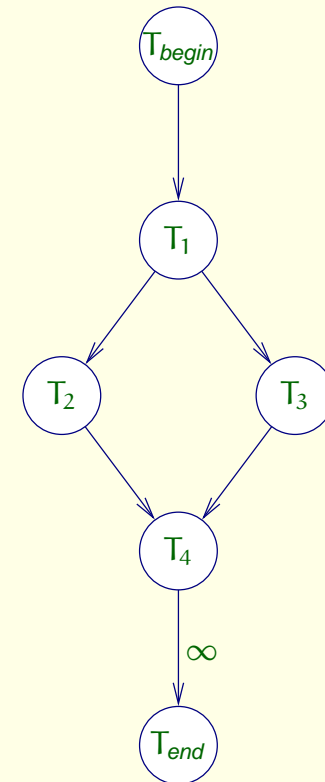
T_{end} connected to every task with no successor in the graph

$w_{i,Begin} = w_{i,End} = 0$ for each processor P_i

$$\forall P_i, \text{consumed}(P_i, T_{begin}) = 0$$

$$\forall P_i, \forall P_j \in n(P_i), \forall e_{k,end} : T_k \rightarrow T_{end},$$

$$\begin{cases} \text{sent}(P_i \rightarrow P_j, e_{k,end}) = 0 \\ \text{sent}(P_j \rightarrow P_i, e_{k,end}) = 0 \end{cases}$$



Steady-state equations \rightarrow conservation laws

- Each time unit, P_i receives from its neighbors a given number of files of type $e_{k,l}$
- P_i itself executes some tasks T_k , namely $consumed(P_i, T_k)$ tasks T_k , thereby generating as many new files of type $e_{k,l}$
- What does happen to these files?
Some are sent to the neighbors of P_i ,
Some are consumed by P_i to execute tasks of type T_l

$$\forall P_i, \forall e_{k,l} \in E : T_k \rightarrow T_l,$$

$$\sum_{P_j \in n(P_i)} sent(P_j \rightarrow P_i, e_{k,l}) + consumed(P_i, T_k) =$$

$$\sum_{P_j \in n(P_i)} sent(P_i \rightarrow P_j, e_{k,l}) + consumed(P_i, T_l) \quad (8)$$

Steady-state equations \rightarrow master processor(s)

$\forall e_{k,l} \in E : T_k \rightarrow T_l$ with $k \neq \textit{begin}$,

$$\sum_{P_j \in \mathfrak{n}(P_{\text{master}})} \textit{sent}(P_j \rightarrow P_{\text{master}}, e_{k,l}) + \textit{consumed}(P_{\text{master}}, T_k) = \sum_{P_j \in \mathfrak{n}(P_{\text{master}})} \textit{sent}(P_{\text{master}} \rightarrow P_j, e_{k,l}) + \textit{consumed}(P_{\text{master}}, T_l) \quad (9)$$

Working out an example (1/2)

P_1 is the master processor:

$$\begin{aligned} \text{sent}(P_2 \rightarrow P_1, e_{1,2}) + \text{sent}(P_3 \rightarrow P_1, e_{1,2}) + \text{consumed}(P_1, T_1) = \\ \text{consumed}(P_1, T_2) + \text{sent}(P_1 \rightarrow P_2, e_{1,2}) + \text{sent}(P_1 \rightarrow P_3, e_{1,2}) \end{aligned}$$

$$\begin{aligned} \text{sent}(P_2 \rightarrow P_1, e_{1,3}) + \text{sent}(P_3 \rightarrow P_1, e_{1,3}) + \text{consumed}(P_1, T_1) = \\ \text{consumed}(P_1, T_3) + \text{sent}(P_1 \rightarrow P_2, e_{1,3}) + \text{sent}(P_1 \rightarrow P_3, e_{1,3}) \end{aligned}$$

$$\begin{aligned} \text{sent}(P_2 \rightarrow P_1, e_{2,4}) + \text{sent}(P_3 \rightarrow P_1, e_{2,4}) + \text{consumed}(P_1, T_2) = \\ \text{consumed}(P_1, T_4) + \text{sent}(P_1 \rightarrow P_2, e_{2,4}) + \text{sent}(P_1 \rightarrow P_3, e_{2,4}) \end{aligned}$$

$$\begin{aligned} \text{sent}(P_2 \rightarrow P_1, e_{3,4}) + \text{sent}(P_3 \rightarrow P_1, e_{3,4}) + \text{consumed}(P_1, T_3) = \\ \text{consumed}(P_1, T_4) + \text{sent}(P_1 \rightarrow P_2, e_{3,4}) + \text{sent}(P_1 \rightarrow P_3, e_{3,4}) \end{aligned}$$

$$\begin{aligned} \text{sent}(P_2 \rightarrow P_1, e_{4,end}) + \text{sent}(P_3 \rightarrow P_1, e_{4,end}) + \text{consumed}(P_1, T_4) = \\ \text{consumed}(P_1, T_{end}) + \text{sent}(P_1 \rightarrow P_2, e_{4,end}) + \text{sent}(P_1 \rightarrow P_3, e_{4,end}) \end{aligned}$$

Working out an example (2/2)

For P_2 :

$$\begin{aligned} & sent(P_1 \rightarrow P_2, e_{begin,1}) + sent(P_3 \rightarrow P_2, e_{begin,1}) + sent(P_4 \rightarrow P_2, e_{begin,1}) + consumed(P_2, T_{begin}) = \\ & \quad consumed(P_2, T_1) + sent(P_2 \rightarrow P_1, e_{begin,1}) + sent(P_2 \rightarrow P_3, e_{begin,1}) + sent(P_2 \rightarrow P_4, e_{begin,1}) \end{aligned}$$

$$\begin{aligned} & sent(P_1 \rightarrow P_2, e_{1,2}) + sent(P_3 \rightarrow P_2, e_{1,2}) + sent(P_4 \rightarrow P_2, e_{1,2}) + consumed(P_2, T_1) = \\ & \quad consumed(P_2, T_2) + sent(P_2 \rightarrow P_1, e_{1,2}) + sent(P_2 \rightarrow P_3, e_{1,2}) + sent(P_2 \rightarrow P_4, e_{1,2}) \end{aligned}$$

$$\begin{aligned} & sent(P_1 \rightarrow P_2, e_{1,3}) + sent(P_3 \rightarrow P_2, e_{1,3}) + sent(P_4 \rightarrow P_2, e_{1,3}) + consumed(P_2, T_1) = \\ & \quad consumed(P_2, T_3) + sent(P_2 \rightarrow P_1, e_{1,3}) + sent(P_2 \rightarrow P_3, e_{1,3}) + sent(P_2 \rightarrow P_4, e_{1,3}) \end{aligned}$$

$$\begin{aligned} & sent(P_1 \rightarrow P_2, e_{2,4}) + sent(P_3 \rightarrow P_2, e_{2,4}) + sent(P_4 \rightarrow P_2, e_{2,4}) + consumed(P_2, T_2) = \\ & \quad consumed(P_2, T_4) + sent(P_2 \rightarrow P_1, e_{2,4}) + sent(P_2 \rightarrow P_3, e_{2,4}) + sent(P_2 \rightarrow P_4, e_{2,4}) \end{aligned}$$

$$\begin{aligned} & sent(P_1 \rightarrow P_2, e_{3,4}) + sent(P_3 \rightarrow P_2, e_{3,4}) + sent(P_4 \rightarrow P_2, e_{3,4}) + consumed(P_2, T_3) = \\ & \quad consumed(P_2, T_4) + sent(P_2 \rightarrow P_1, e_{3,4}) + sent(P_2 \rightarrow P_3, e_{3,4}) + sent(P_2 \rightarrow P_4, e_{3,4}) \end{aligned}$$

$$\begin{aligned} & sent(P_1 \rightarrow P_2, e_{4,end}) + sent(P_3 \rightarrow P_2, e_{4,end}) + sent(P_4 \rightarrow P_2, e_{4,end}) + consumed(P_2, T_4) = \\ & \quad consumed(P_2, T_{end}) + sent(P_2 \rightarrow P_1, e_{4,end}) + sent(P_2 \rightarrow P_3, e_{4,end}) + sent(P_2 \rightarrow P_4, e_{4,end}) \end{aligned}$$

Optimal solution

STEADY-STATE SCHEDULING PROBLEM SSSP(G)

Maximize

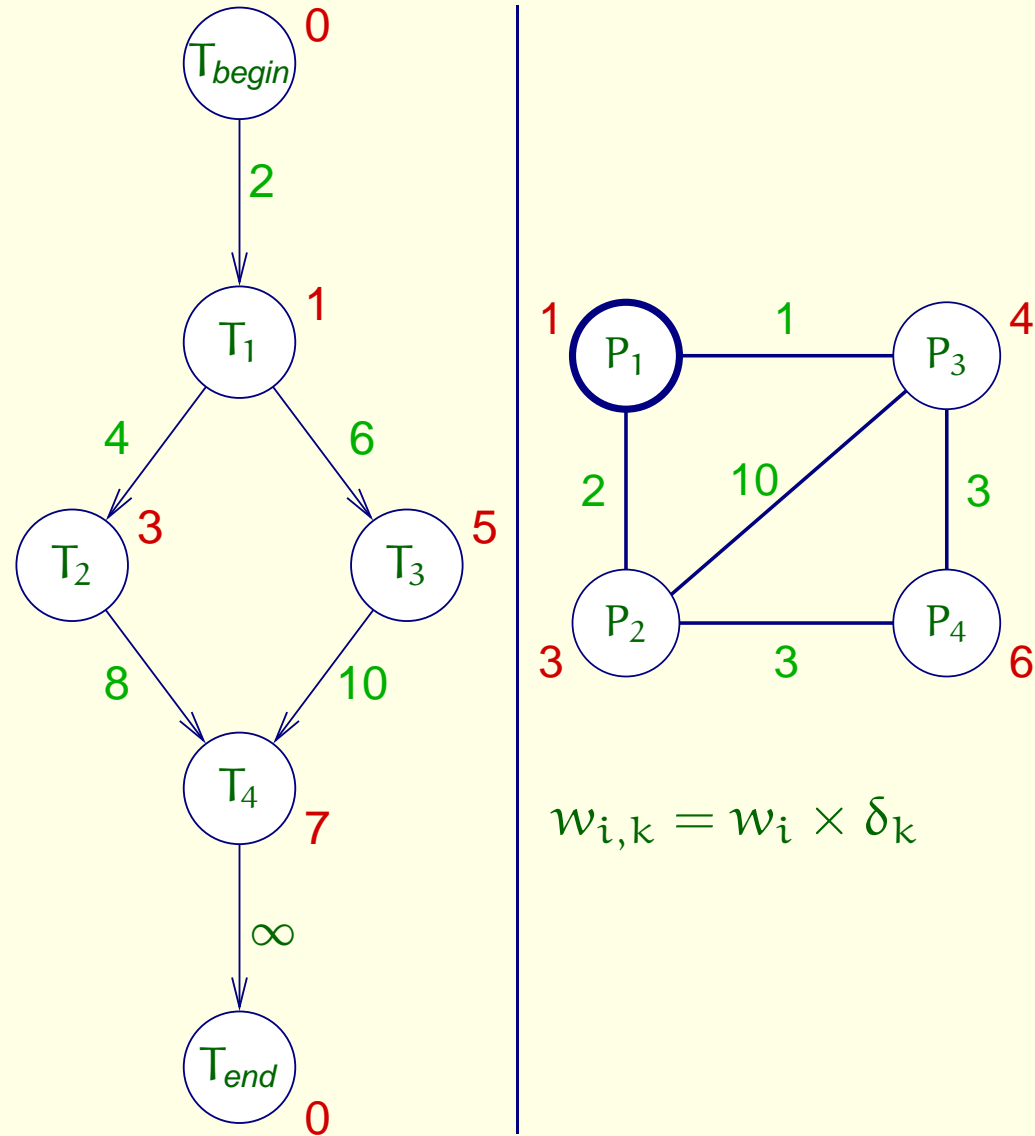
$$TP = \sum_{i=1}^p \sum_i \text{consumed}(P_i, T_{end}),$$

subject to

$$\left\{ \begin{array}{l} \forall i, j, \forall k, \quad 0 \leq \alpha(P_i, T_k) \leq 1 \\ \forall i, j, \forall e_{k,l} \in E, \quad 0 \leq s(P_i \rightarrow P_j, e_{k,l}) \leq 1 \\ \forall i, j, \forall e_{k,l} \in E, \quad s(P_i \rightarrow P_j, e_{k,l}) = \text{sent}(P_i \rightarrow P_j, e_{k,l}) \times (\text{data}_{k,l} \times c_{i,j}) \\ \forall i, \forall k, \quad \alpha(P_i, T_k) = \text{consumed}(P_i, T_k) \times w_{i,k} \\ \forall i, \quad \sum_{P_j \in n(P_i)} \sum_{e_{k,l} \in E} s(P_i \rightarrow P_j, e_{k,l}) \leq 1 \\ \forall i, \quad \sum_{P_j \in n(P_i)} \sum_{e_{k,l} \in E} s(P_j \rightarrow P_i, e_{k,l}) \leq 1 \\ \forall i, \quad \sum_{T_k \in V} \alpha(P_i, T_k) \leq 1 \\ \forall i, \quad \text{consumed}(P_i, T_{begin}) = 0 \\ \forall i, j, \forall e_{k,end} \quad \text{sent}(P_i \rightarrow P_j, e_{k,end}) = 0 \\ \forall P_i, \forall e_{k,l} \in E \quad \sum_{P_j \in n(P_i)} \text{sent}(P_j \rightarrow P_i, e_{k,l}) + \text{consumed}(P_i, T_k) = \\ \sum_{P_j \in n(P_i)} \text{sent}(P_i \rightarrow P_j, e_{k,l}) + \text{consumed}(P_i, T_l) \\ \forall e_{k,l} \in E \text{ with } k \neq \text{begin}, \quad \sum_{P_j \in n(P_{\text{master}})} \text{sent}(P_j \rightarrow P_{\text{master}}, e_{k,l}) + \text{consumed}(P_{\text{master}}, T_k) = \\ \sum_{P_j \in n(P_{\text{master}})} \text{sent}(P_{\text{master}} \rightarrow P_j, e_{k,l}) + \text{consumed}(P_{\text{master}}, T_l) \end{array} \right.$$

Theorem 2. *The solution to the previous linear programming problem provides the optimal solution to SSSP(G)*

Back to the example (1)



Back to the example (2)

	T_1	T_2	T_3	T_4
P_1	$7/64$	$13719/91840$	$7213/18368$	$4573/13120$
P_2	0	$1851/11480$	$531/1148$	$617/1640$
P_3	0	$33/82$	0	$49/82$
P_4	0	$6/41$	0	$35/41$

Table 1: Optimal solutions for $\alpha(P_i, T_k)$

	T_{begin}	T_1	T_2	T_3	T_4	T_{end}
P_1	0	$7/64 \approx 0.11$	$4573/91840 \approx 0.05$	$7213/91840 \approx 0.08$	$4573/91840 \approx 0.05$	$4573/91840 \approx 0.05$
P_2	0	0	$617/34440 \approx 0.02$	$177/5740 \approx 0.03$	$617/34440 \approx 0.02$	$617/34440 \approx 0.02$
P_3	0	0	$11/328 \approx 0.03$	0	$7/328 \approx 0.02$	$7/328 \approx 0.02$
P_4	0	0	$1/123 \approx 0.01$	0	$5/246 \approx 0.02$	$5/246 \approx 0.02$
Total						0.11

Table 2: Optimal solutions for $consumed(P_i, T_k)$

Back to the example (3)

- $TP = 7/64$: the whole platform processes 7 tasks every 64 seconds
- Actual period = 91840
- Fastest processor P_1 used alone \Rightarrow 4 tasks every 64 units of time
- Much better results, despite
 - all the dependences,
 - all the communication overheads,
 - the fact that the other processors are at least three times slower than P_1
- Solution is not trivial, in that processors do not execute tasks of all types

Related work

Master-slave on the grid

- Shao-Berman: network flow
- Heymann, Senar, Luque and Livny: adaptive strategy
- Goux-Kulkarni-Linderoth-Yoder: frameworks
- Heymann-Seynar-Luque-Livny: Legion
- Beaumont-Carter-Ferrante-Legrand-Robert: independent tasks

Mixed task and data parallelism

- Taura and Chien: pipeline execution of task graphs onto heterogeneous platforms, but all copies of a given task type must be executed on the same processor

Asymptotic results

- Bertsimas and Gamarnik: fluid relaxation technique (inspired by the work of Sevast'janov) to derive asymptotically optimal job shop scheduling algorithms

Limitations of static scheduling

Good news and bad news

- 😊 One-port model first step towards designing realistic scheduling heuristics
- 😊 Perfectly divisible load simplifies task allocation
- 😊 Steady-state circumvents computational complexity of scheduling problems while deriving efficient (often asymptotically optimal) scheduling algorithms
- 😞 Need to acquire a good knowledge of the platform graph
- 😞 Need to run extensive experiments or simulations

Knowledge of the platform graph

- For regular problems, the structure of the task graph (nodes and edges) only depends upon the application, not upon the target platform
- Problems arise from weights, i.e. the estimation of execution and communication times
- Classical answer: “use the past to predict the future”
- Divide scheduling into phases, during which machine and network parameters are collected (with NWS)
 - ⇒ This information guides scheduling decisions for next phase
- Moving from heterogeneous clusters to computational grids causes further problems
- Discovering the characteristics of the surrounding computing resources may prove a difficult task

Experiments versus simulations

- Real experiments difficult to drive (genuine instability of non-dedicated platforms)
- Simulations ensure reproducibility of measured data
- Key issue: run simulations against a realistic environment
- Trace-based scheduling: record platform parameters today, and simulate the algorithms tomorrow, against recorded data
- Use SIMGRID, an event-driven toolkit

Conclusion

- If the platform is well identified and relatively stable, try to:
 - (i) accurately model the (expected) hierarchical structure of the platform
 - (ii) design scheduling algorithms well-suited to this hierarchical structure
- If the platform is not stable enough, or if it evolves too fast, dynamic schedulers are the only option
- Otherwise, grab the opportunity to inject some static knowledge into dynamic schedulers:
 - ☹ Is this opportunity a niche?
 - 😊 Does it encompasses a wide range of applications?