Allocating Series of Workflows on Computing Grids

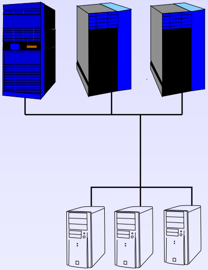
Loris Marchal,

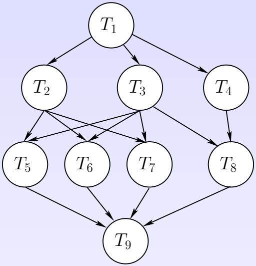
joint work with Matthieu Gallet, Mathias Jacquelin, and Frédéric Vivien

CNRS
INRIA GRAAL project-team
Laboratoire de l'Informatique du Parallélisme
École Normale Supérieure de Lyon, France

ASTEC workshop, June 4, 2009.

Introduction





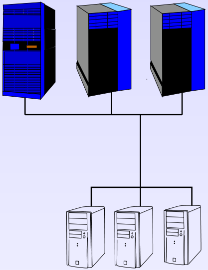
Our problem

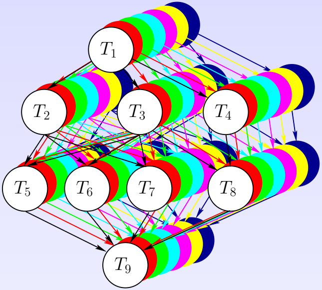
- A fully heterogeneous platform
- A complex task graph G_A to be executed many times

Possible solutions

- Use any heuristic to schedule as if it were a single task graph
- Take advantage of the problem regularity

Introduction





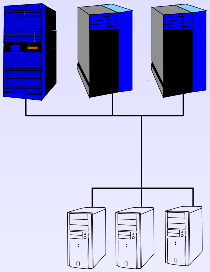
Our problem

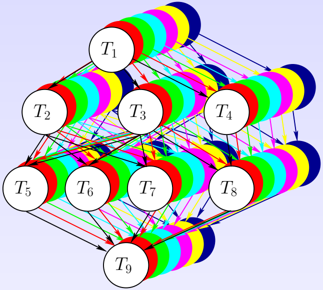
- A fully heterogeneous platform
- A complex task graph G_A to be executed many times

Possible solutions

- Use any heuristic to schedule as if it were a single task graph
- Take advantage of the problem regularity

Introduction





Our problem

- A fully heterogeneous platform
- A complex task graph G_A to be executed many times

Possible solutions

- Use any heuristic to schedule as if it were a single task graph
- Take advantage of the problem regularity

Outline

Outline

Picking an appropriate objective

Makespan minimization

- Minimize the time elapsed between the processing of the first task and the completion of the overall work

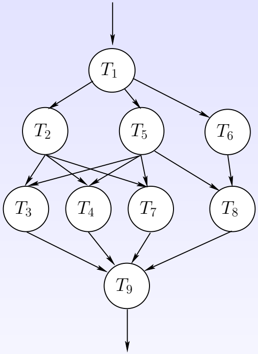
Steady-state scheduling

- Neglect initiation and termination phases
- Focus on the average of the schedule
- Maximize the platform throughput
  (Average number of task graphs completed per time unit)

Allocation

An allocation of the application graph to the platform graph is a function σ associating:

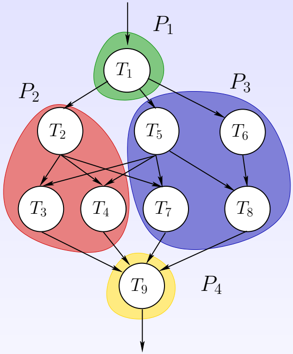
- to each task T_i, a processor $\sigma(T_i)$ which processes all instances of T_i;
- to each file $F_{i,j}$, a set of communication links $\sigma(F_{i,j})$ which carries all instances of this file from processor $\sigma(T_i)$ to processor $\sigma(T_j)$.



Allocation

An allocation of the application graph to the platform graph is a function σ associating:

- to each task T_i, a processor $\sigma(T_i)$ which processes all instances of T_i;
- to each file $F_{i,j}$, a set of communication links $\sigma(F_{i,j})$ which carries all instances of this file from processor $\sigma(T_i)$ to processor $\sigma(T_j)$.



Existing steady-state approach

Actual knowledge

Schedule maximizing the throughput known when the application graph is not too deep.
Scheduling strategies for mixed data and task parallelism on heterogeneous clusters, O. Beaumont, A. Legrand, L. Marchal, and Y. Robert, Parallel Processing Letters 13(2), 2003.

Problem

Requires a lot of control as a schedule can use many different allocations

Question

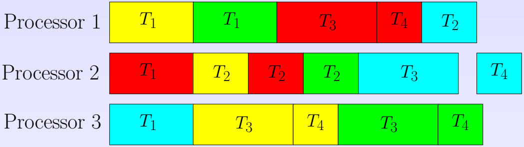
Can we build simpler but as efficient schedules?

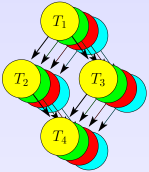
Tool

Single-allocation steady-state schedules

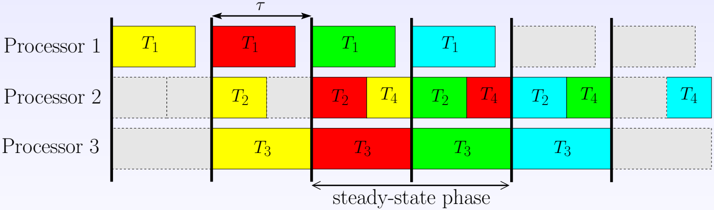
Example of schedules

Any schedule:





Periodic schedule, with only one single allocation:



- Regularity of schedule $\rightarrow$ optimization much more tractable
- We may lose in performance because of these constraints

Complexity

Problem DAG-Single-Alloc

Given a directed acyclic application graph, a platform graph, and a bound B, is there an allocation with throughput $\rho \geq B$?

Theorem.

DAG-Single-Alloc is NP-complete

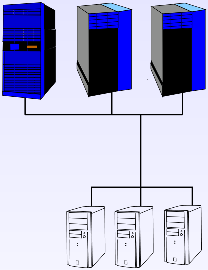
Outline

Notations: platform

- $G_P = (V_P, E_P)$: platform graph
- $V_P = P_0, \ldots, P_{n-1}$: processors
- $E_P = (P_q \rightarrow P_r)$: communication links
- Path $P_q \rightsquigarrow P_r$: set of links

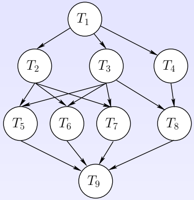
- Limited incoming bandwidth B_q^{in}
- Limited outgoing bandwidth B_q^{out}
- Limited bandwidth per link $\text{bw}_{q,r}$

- Unrelated processors
- Initially, P_0 holds the input files
- All output files must be sent back to P_0



Notations: application

- $G_A = (V_A, E_A)$: Directed Acyclic Graph
- $V_A = T_0, \ldots, T_{k-1}$: tasks to process
- $E_A = (F_{i,j})_{i,j}$: files to transmit between tasks



- Many instances of G_A

- Time to transmit a file: $\frac{\mathrm{data}_{i,j}}{\mathrm{bw}_{q,r}}$
- Time to compute a task: $w_{i,q}$

Objective: maximize the throughput

- Minimize the period τ (time needed to process/transmit one instance of each task/file transfer)

Integer variables

- $y_q^k = 1$ if task T_k is processed on processor P_q, and $y_q^k = 0$ otherwise
- Each task is processed exactly once:

$$\forall T_k, \quad \sum_{P_q} y_q^k = 1$$

- $x_{q,r}^{k,l} = 1$ if file $F_{k,l}$ is transferred using path $P_q \leadsto P_r$, and $x_{q,r}^{k,l} = 0$ otherwise
- A file transfer must originate from where the file was produced:

$$x_{q,r}^{k,l} \leq y_q^k$$

Integer variables

- $y_q^k = 1$ if task T_k is processed on processor P_q, and $y_q^k = 0$ otherwise
- Each task is processed exactly once:

$$\forall T_k, \quad \sum_{P_q} y_q^k = 1$$

- $x_{q,r}^{k,l} = 1$ if file $F_{k,l}$ is transferred using path $P_q \rightsquigarrow P_r$, and $x_{q,r}^{k,l} = 0$ otherwise
- A file transfer must originate from where the file was produced:

$$x_{q,r}^{k,l} \leq y_q^k$$

Constraints on computations

- The processor computing a task must hold all necessary input data, i.e., it either received or computed any required input data:

$$y_r^k + \sum_{P_q \rightsquigarrow P_r} x_{q,r}^{k,l} \geq y_r^l$$

- The computing time of a processor is no larger that τ:

$$\sum_{T_k} y_q^k \times w_{q,k} \leq \tau$$

Constraints on computations

- The processor computing a task must hold all necessary input data, i.e., it either received or computed any required input data:

$$y_r^k + \sum_{P_q \rightsquigarrow P_r} x_{q,r}^{k,l} \geq y_r^l$$

- The computing time of a processor is no larger that τ:

$$\sum_{T_k} y_q^k \times w_{q,k} \leq \tau$$

Constraints on communications

- The amount of data carried by the link $P_q \to P_r$ is:

$$d_{q,r} = \sum_{\substack{P_s \rightsquigarrow P_t \text{ with} \\ P_q \to P_r \in P_s \rightsquigarrow P_t}} \sum_{F_{k,l}} x_{s,t}^{k,l} \times \mathrm{data}_{k,l}$$

- The link bandwidth must not be exceeded:

$$\frac{d_{q,r}}{\mathrm{bw}_{q,r}} \leq \tau$$

- The output bandwidth of a processor P_q must not be exceeded:

$$\sum_{P_q \to P_r \in E_P} \frac{d_{q,r}}{B_q^{\mathrm{out}}} \leq \tau$$

- The input bandwidth of a processor P_q must not be exceeded:

$$\sum_{P_q \to P_r \in E_P} \frac{d_{q,r}}{B_r^{\mathrm{in}}} \leq \tau$$

Constraints on communications

- The amount of data carried by the link $P_q \to P_r$ is:

$$d_{q,r} = \sum_{\substack{P_s \rightsquigarrow P_t \text{ with} \\ P_q \to P_r \in P_s \rightsquigarrow P_t}} \sum_{F_{k,l}} x_{s,t}^{k,l} \times \mathrm{data}_{k,l}$$

- The link bandwidth must not be exceeded:

$$\frac{d_{q,r}}{\mathrm{bw}_{q,r}} \leq \tau$$

- The output bandwidth of a processor P_q must not be exceeded:

$$\sum_{P_q \to P_r \in E_P} \frac{d_{q,r}}{B_q^{\mathrm{out}}} \leq \tau$$

- The input bandwidth of a processor P_q must not be exceeded:

$$\sum_{P_q \to P_r \in E_P} \frac{d_{q,r}}{B_r^{\mathrm{in}}} \leq \tau$$

Constraints on communications

- The amount of data carried by the link $P_q \to P_r$ is:

$$d_{q,r} = \sum_{\substack{P_s \rightsquigarrow P_t \text{ with} \\ P_q \to P_r \in P_s \rightsquigarrow P_t}} \sum_{F_{k,l}} x_{s,t}^{k,l} \times \mathrm{data}_{k,l}$$

- The link bandwidth must not be exceeded:

$$\frac{d_{q,r}}{\mathrm{bw}_{q,r}} \leq \tau$$

- The output bandwidth of a processor P_q must not be exceeded:

$$\sum_{P_q \to P_r \in E_P} \frac{d_{q,r}}{B_q^{\mathrm{out}}} \leq \tau$$

- The input bandwidth of a processor P_q must not be exceeded:

$$\sum_{P_q \to P_r \in E_P} \frac{d_{q,r}}{B_r^{\mathrm{in}}} \leq \tau$$

Constraints on communications

- The amount of data carried by the link $P_q \to P_r$ is:

$$d_{q,r} = \sum_{\substack{P_s \rightsquigarrow P_t \text{ with} \\ P_q \to P_r \in P_s \rightsquigarrow P_t}} \sum_{F_{k,l}} x_{s,t}^{k,l} \times \mathrm{data}_{k,l}$$

- The link bandwidth must not be exceeded:

$$\frac{d_{q,r}}{\mathrm{bw}_{q,r}} \leq \tau$$

- The output bandwidth of a processor P_q must not be exceeded:

$$\sum_{P_q \to P_r \in E_P} \frac{d_{q,r}}{B_q^{\mathrm{out}}} \leq \tau$$

- The input bandwidth of a processor P_q must not be exceeded:

$$\sum_{P_q \to P_r \in E_P} \frac{d_{q,r}}{B_r^{\mathrm{in}}} \leq \tau$$

Objective

Minimize the maximum time τ spent by all resources

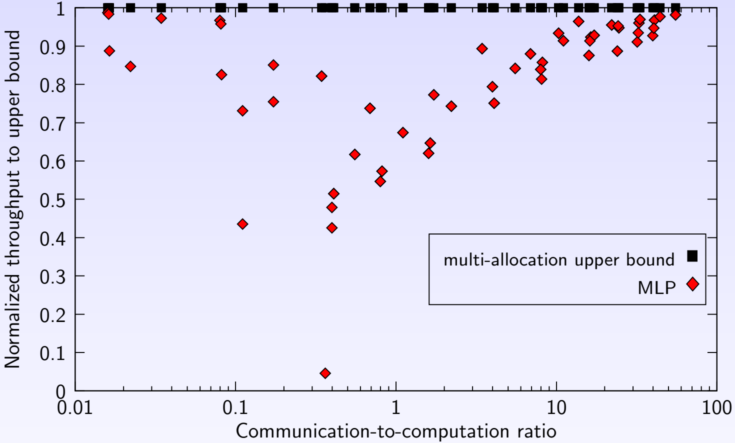
Throughput: $1/\tau$.

Theorem.

An optimal solution of the above linear program describes an allocation with maximal throughput

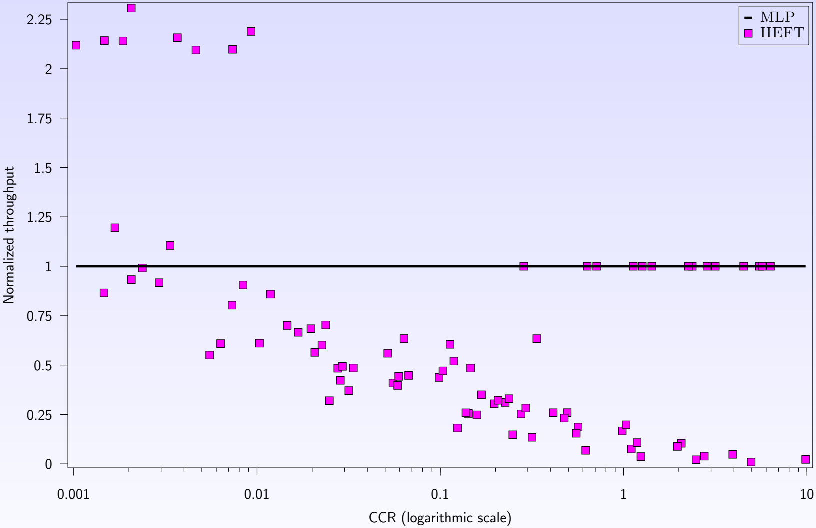
- NP-complete problem
- Mixed-linear programs for small instances

Mono-allocation vs. multi-allocation



Single allocation solutions achieve most of the performance of multi-allocation solutions

Mono-allocation vs. traditional dynamic approach



As soon as communications matter
the steady-state approach is more efficient

Outline

Greedy mapping strategies

- Simple mapping:
  - put the "largest" task on the best processor
  - continue with the second "largest" task, put it on the processor which decreases the least the throughput
  - ...
- Refined greedy:
  - take communication times into account when sorting tasks
  - when mapping a task, select the processor such that the maximum occupation time of all resources (processors and links) is minimized

Rounding of the linear program

1. Solve the linear program over the rationals
2. Based on the rational solution, select an integer variable and its value:

   RLP-max:

   - Select the y_i^k with maximum value
   - Set y_j^k to 1

   RLP-rand:

   - Select a task T_k not yet mapped
   - Randomly choose a processor P_i with probability y_i^k
   - Set y_j^k to 1

3. Goto step 1 until all variables are set

Delegating computations

- Start from the solution where all tasks are processed by the source processor
- Try to move a (connected) subset of tasks to another processor to increase the throughput
- Repeat this process until no more improvement is found

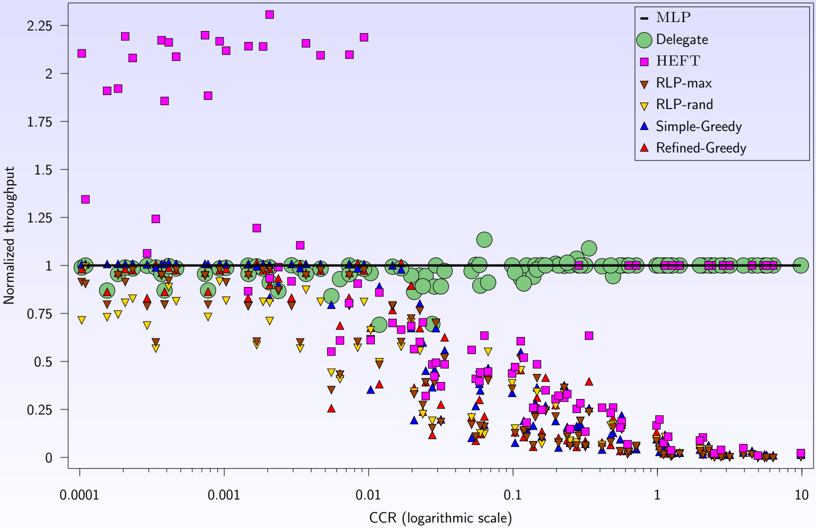
Several issues to overcome:

- Find interesting groups of tasks to delegate
  - for all tasks, we test all possible immediate neighborhoods, and then try to increase the group along chains
- Hard to find a good evaluation metric: some moves do not directly decrease throughput, but are still interesting
  - for a given mapping, we sort all resource occupation times by lexicographical order and use the ordered list instead of the throughput in comparisons

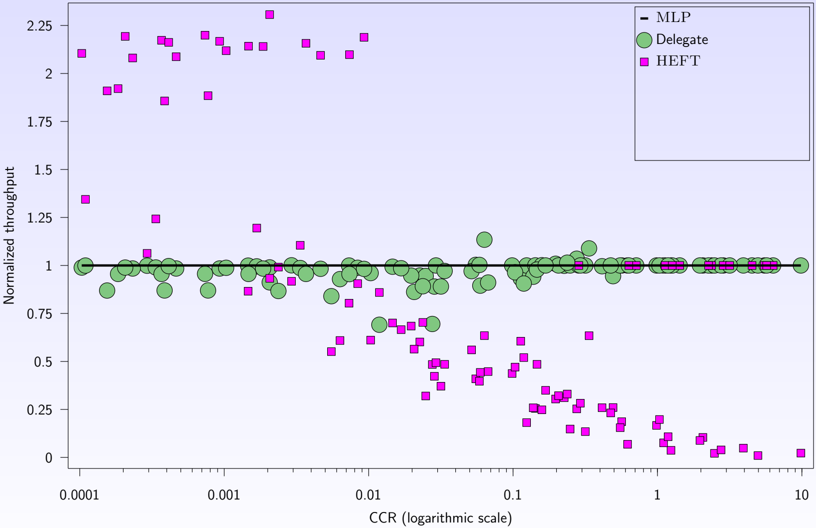
Performance evaluation – methodology

- Reference heuristic: HEFT
- LP and MLP solved with CPLEX 11
- Simulations done using SimGrid
- Platforms: actual Grids, from SimGrid repository
  (only a subset of processors is available for computation)
- Applications: random task graphs + one real application
  - "Small problems": 8–12 tasks
  - "Large problems": up to 47 tasks (MLP not used)
  - for each application, we compute a $\mathrm{CCR} = \dfrac{\text{communications}}{\text{computations}}$
  - we try to cover a large CCR range

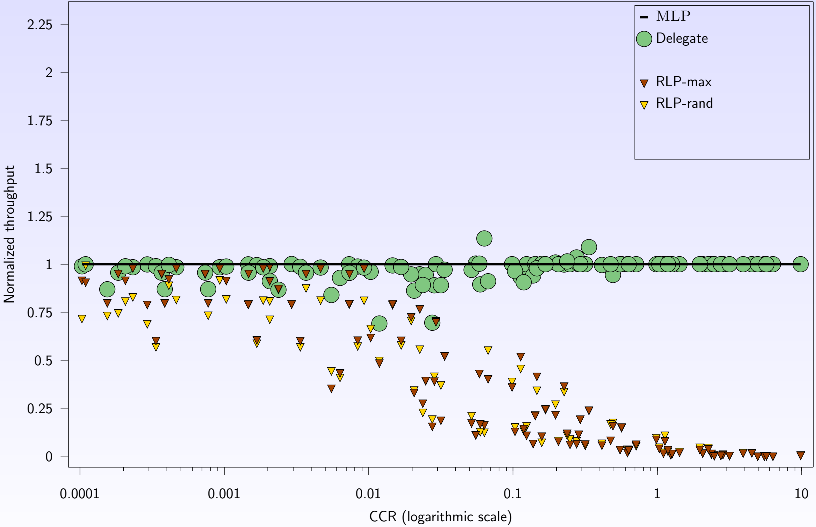
Performance evaluation – results on small problems



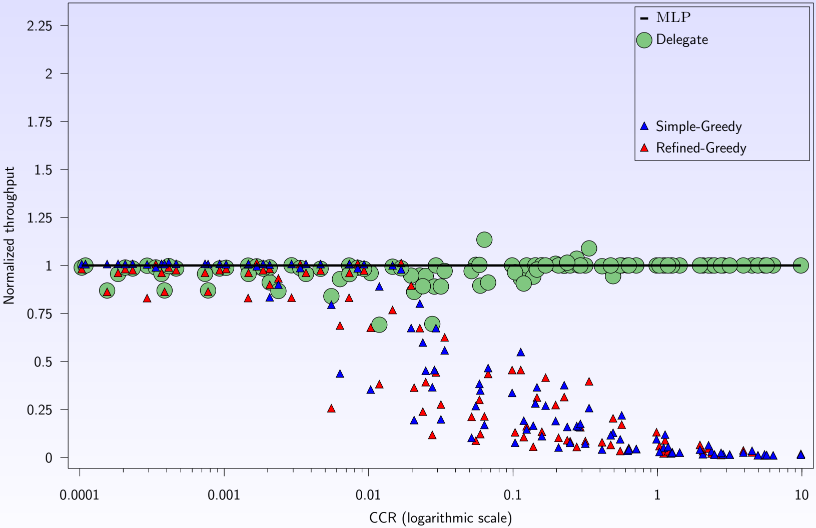
Performance evaluation – results on small problems



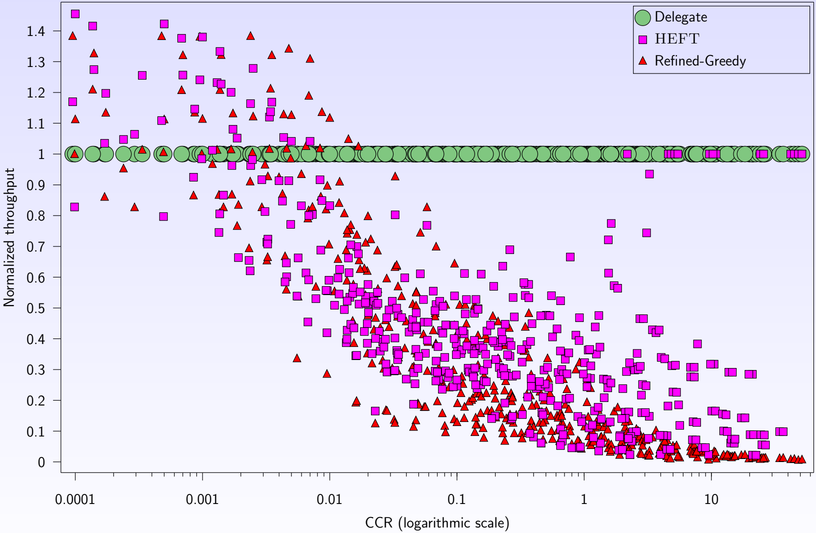
Performance evaluation – results on small problems



Performance evaluation – results on small problems



Performance evaluation – results on large problems



Performance evaluation – running times

Average running times in seconds to schedule 1000 instances:

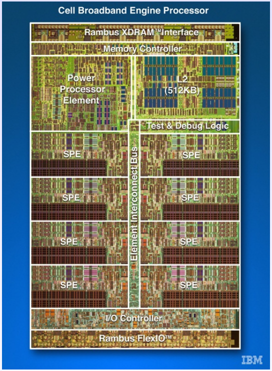
	small task graphs	large task graphs
HEFT *	14.30	83.36
MLP	49.45	n/a
Delegate	16.74	40.49
Simple-Greedy	0.11	0.61
Refined-Greedy	0.12	0.81
RLP-max	166.38	1301.80
RLP-rand	16.78	812.30

*: HEFT running time grows with the number of instances

Outline

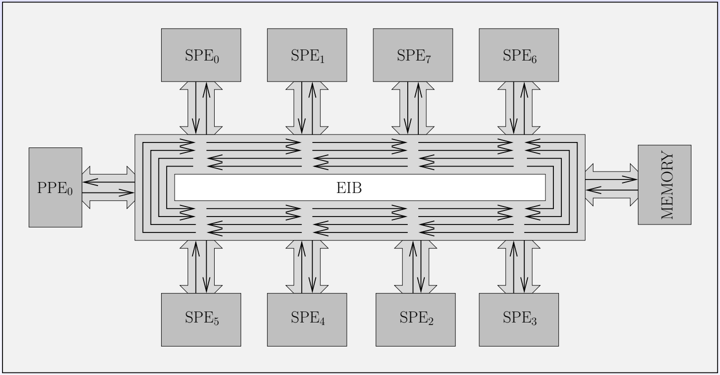
CELL brief introduction

- Multicore heterogeneous processor
- Accelerator extension to Power architecture



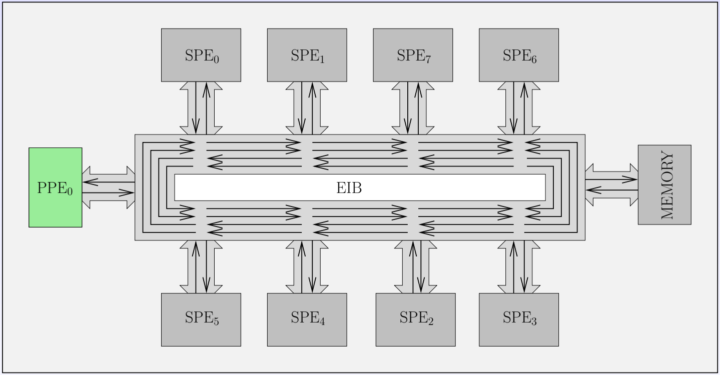
CELL brief introduction

- Multicore heterogeneous processor
- Accelerator extension to Power architecture



CELL brief introduction

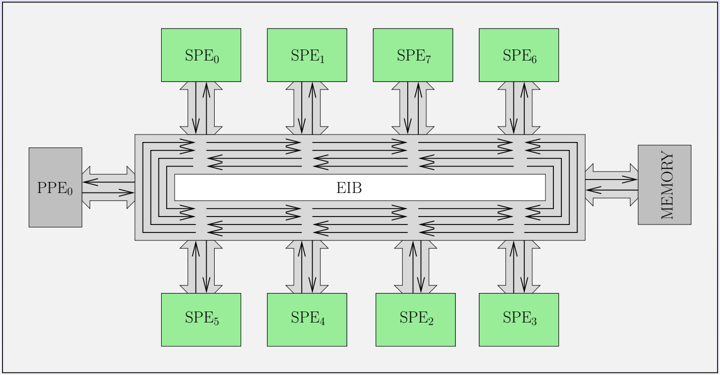
- Multicore heterogeneous processor
- Accelerator extension to Power architecture



- 1 PPE core
  - VMX unit
  - L1, L2 cache
  - 2 way SMT

CELL brief introduction

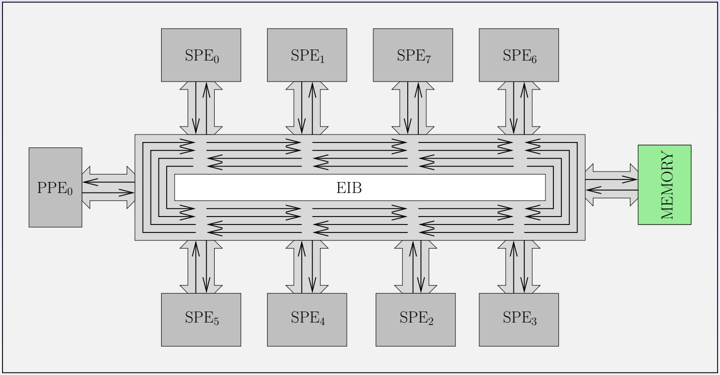
- Multicore heterogeneous processor
- Accelerator extension to Power architecture



- 8 SPEs
  - 128-bit SIMD instruction set
  - Local store 256KB
  - Dedicated Asynchronous DMA engine

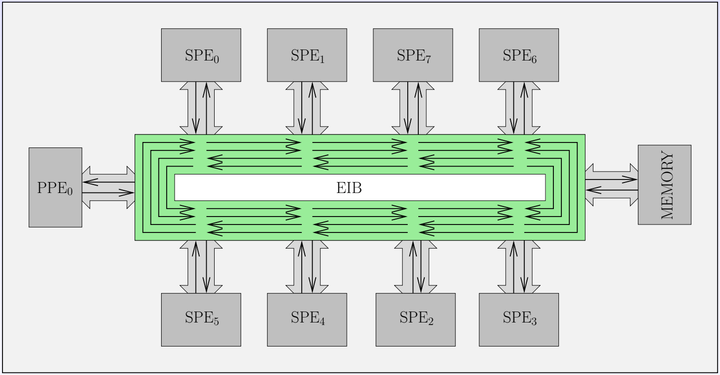
CELL brief introduction

- Multicore heterogeneous processor
- Accelerator extension to Power architecture



CELL brief introduction

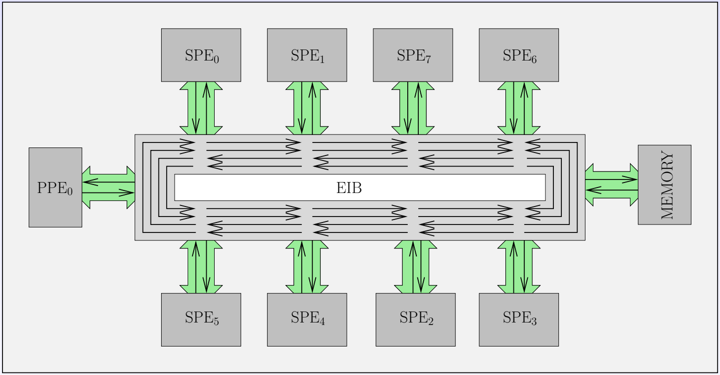
- Multicore heterogeneous processor
- Accelerator extension to Power architecture



- Element Interconnect Bus (EIB)
  - 200 GB/s bandwidth

CELL brief introduction

- Multicore heterogeneous processor
- Accelerator extension to Power architecture



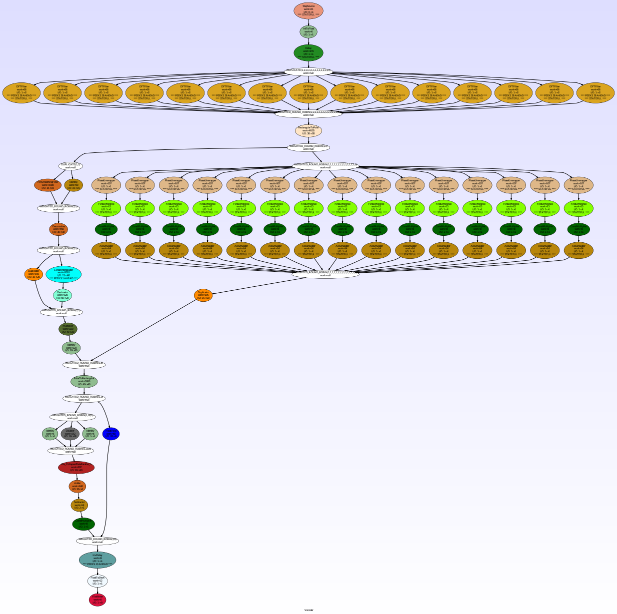
- 25 GB/s bandwidth

Platform modeling

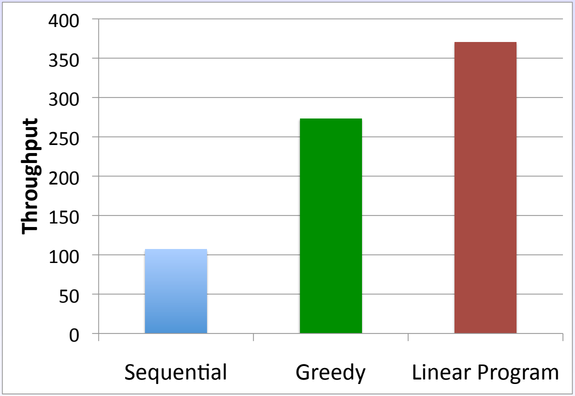
Simple CELL modeling:

- 1 PPE and 8 SPE: 9 processing elements $P_1, \ldots, P_9$, with *unrelated* speed,
- Each processing element access the communication bus with a (bidirectional) bandwidth $b = (25GB/s)$,
- The bus is able to route all concurrent communications without contention (in a first step),
- Constraints on the number of simultaneous communications, because of the size of the stack of the DMA engine
- Constraints on the size of the memory on each SPE

Target application: vocoder



Preliminary results



- Sequential: uses only the PPE core
- Greedy: greedy allocation of tasks to the processing elements

Still some work to do. . .

- Better communication modeling (no contention)
- Implementation on multiple CELL, clusters. . .
- More heterogeneity: CELL + other processing units (GPU)
- Test the heuristics on this platform

Outline

Conclusion

- Single-allocation steady-state schedules have performance close to those of multi-allocations steady-state schedules, as soon as communications matter.
- Best single-allocation steady-state schedules have better performance than HEFT, as soon as communications matter.
- Mixed-linear programming approach limited to "small" problems.
- Design of an efficient heuristic to approach optimal solution for "large" problems.

Perspectives

- Optimize Delegate running time.
- Simplify MLP to cope with larger problems (?)
- Use task duplication to improve throughput.
  (MLP adaptation is straightforward)
- Enhance the model to cope with different architectures