Mapping task graphs to the CELL BE processor

Martino Ruggiero
martino.ruggiero@unibo.it

Luca Benini
luca.benini@unibo.it

University of Bologna
Italy

Outline

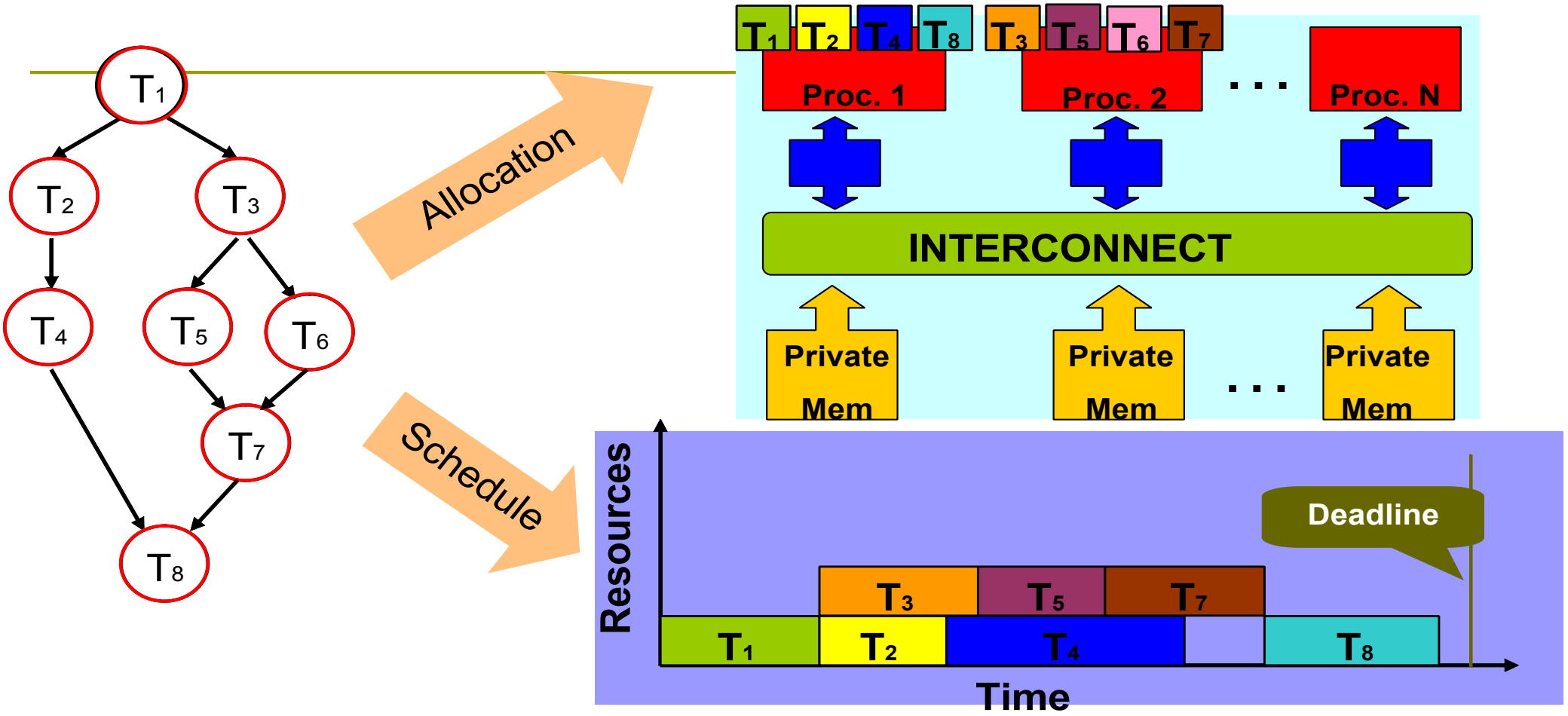
- Introduction
- Target HW architecture
- Target Application & Modeling
- Our approach
- Experimental results

Resource Management on Multi Processor System-on-Chip Platforms

- Providing support for multimedia applications on MPSoC platforms remains a significant research challenge.

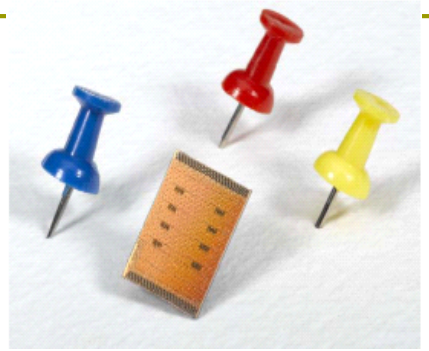
- New tools for efficient mapping of applications onto hardware platforms

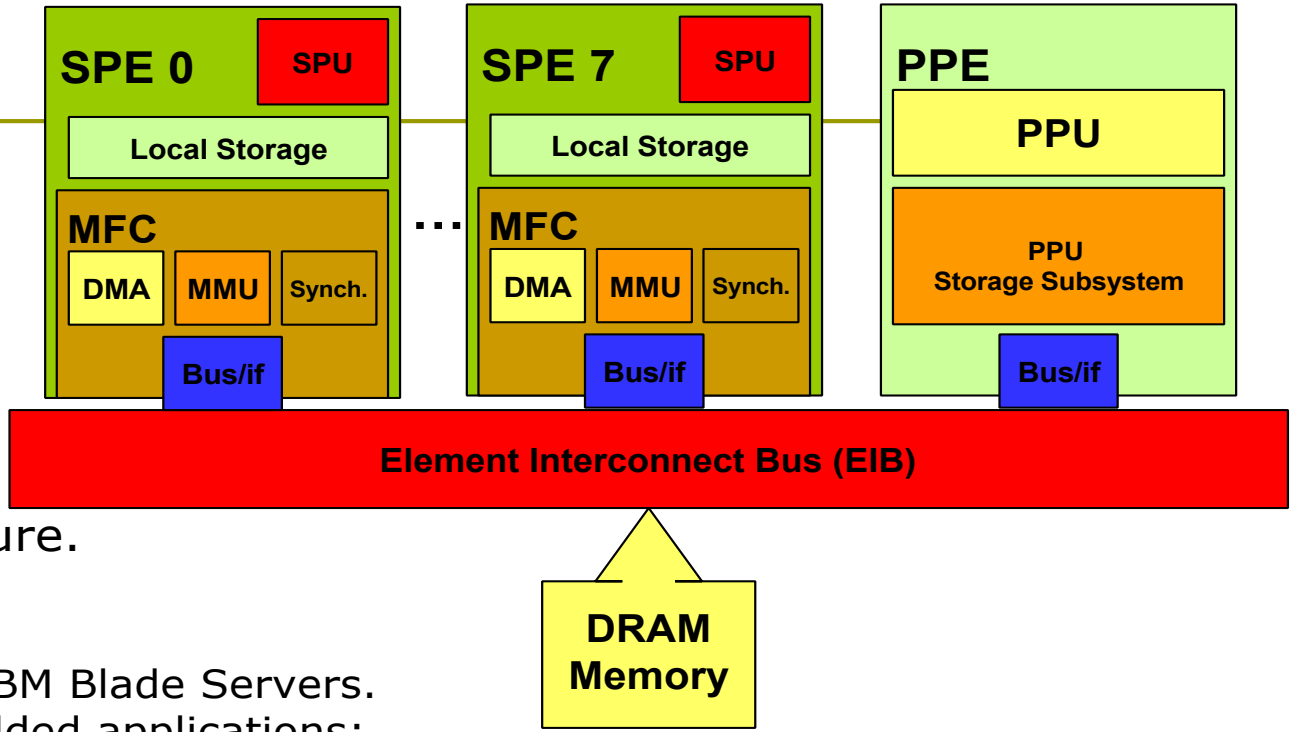
The Optimization Challenge



- The problem of allocating and scheduling task graphs on processors in a distributed real-time system is NP-hard.

Target HW Architecture: STI Cell BE





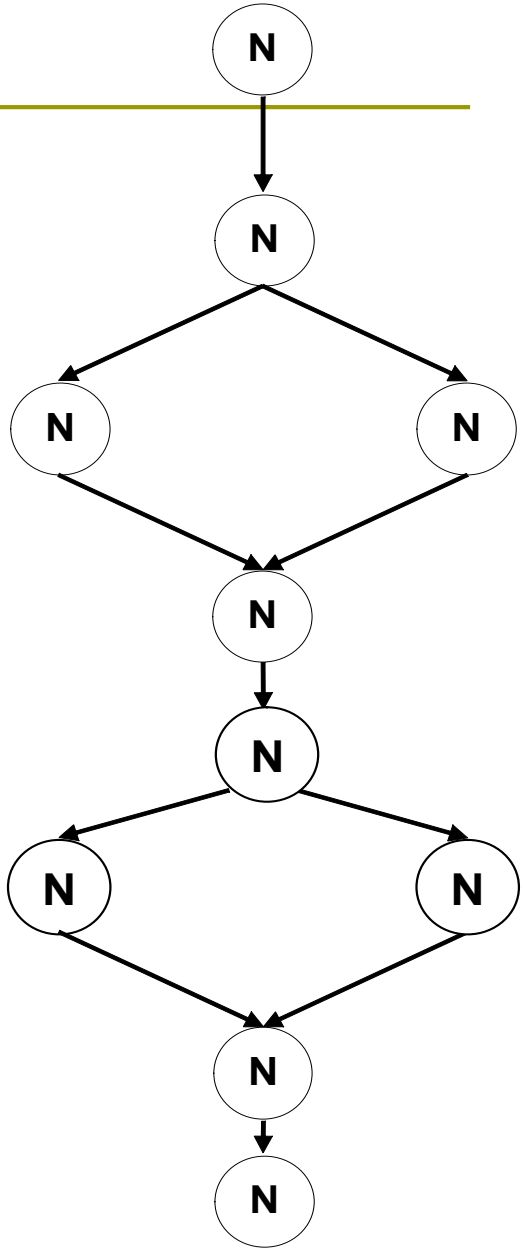
A **multi-core** system architecture.

- It addresses:
  - **Server** applications:
    - Next generation IBM Blade Servers.
  - **High-performance** embedded applications:
    - Gaming (Sony PS3).
    - Aerospace and defence.
    - Medical imaging.
- **Heterogeneous** system architecture:
  - One 64-bit Power Processor Element (PPE)
  - 8 Synergistic Processing Elements (SPEs)
  - Element Interconnect Bus:
  - DMA-based
- **Limited Local Storage**
  - 256KB for Instructions and data
- **Explicit Resource management**

Target Application: Task Graph (TG)

- Statically scheduled Task Graph Applications:
  - Explicit parallelism;
  - Message Passing Communication;
  - Single-token Communication.

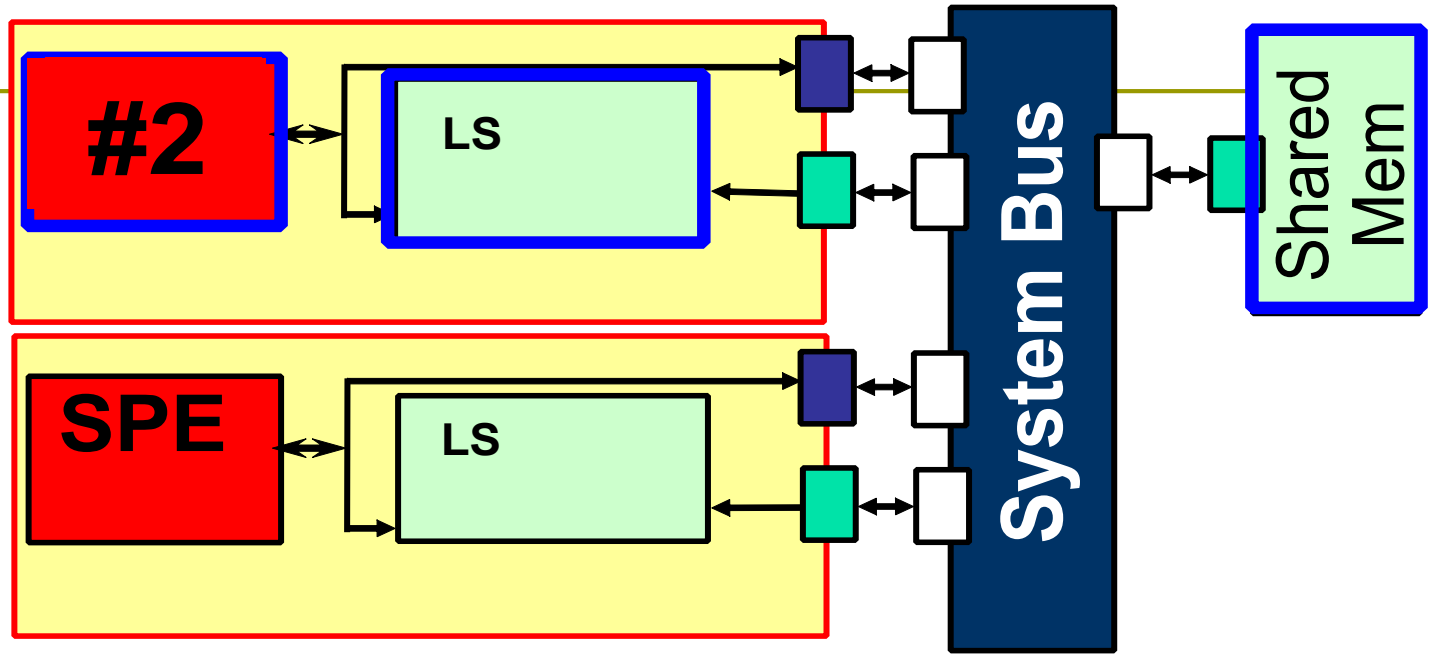
- A **TG** is a couple **<T,A>**, where:
  - **T** is the set of **nodes** modelling generic tasks (e.g. elementary operations, subprograms, ...);
  - **A** the set of **arcs** modelling precedence constraints (e.g. due to data communication);
  - WCET for Comp. & Comm. Modelling.



Task memory requirements

Each task has three kinds of memory requirements:

- Program Data;
- Internal State;
- Communication queues.



Program Data & Internal State can be allocated:

- On the local LS;
- On the remote Shared Memory.

The communication task might run:

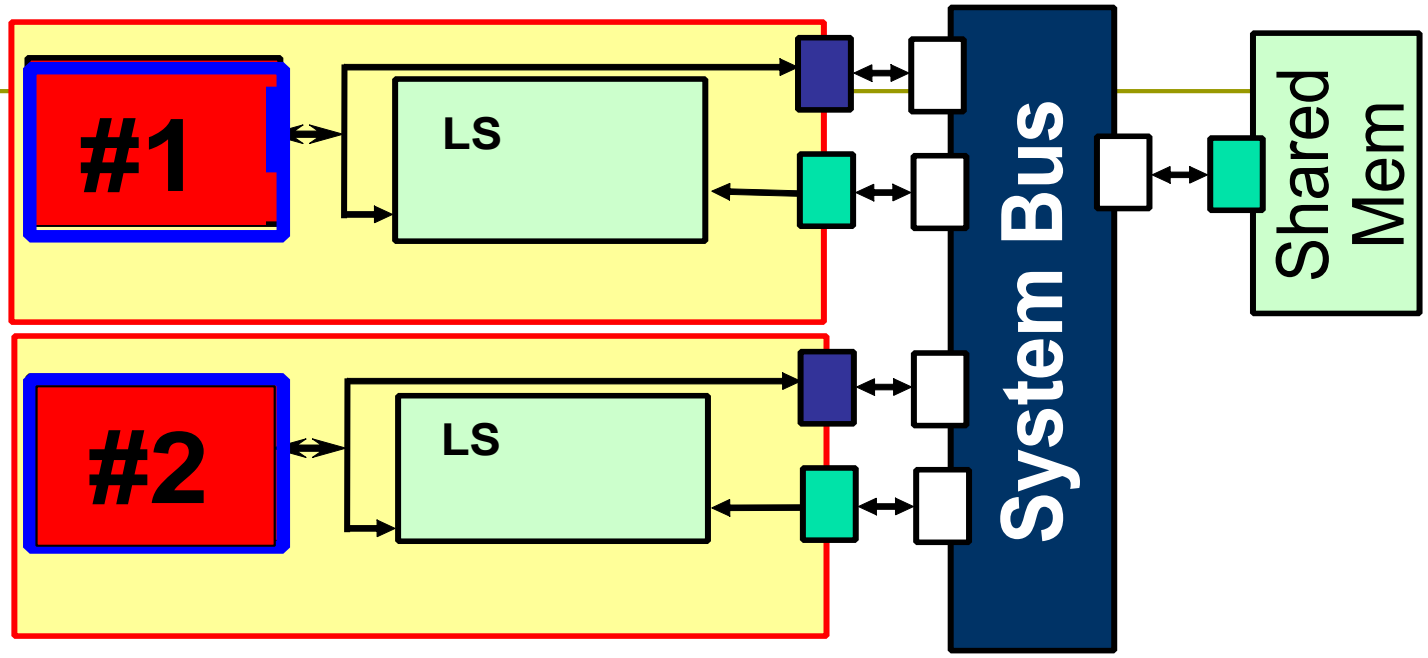
- On the same SPE → negligible communication cost
- On a remote SPE → costly read or write procedure
- On Shared Memory → costly message exchange procedure

- Communication queues in LS → more efficient message passing
  - **Memory size limit!**

Task memory requirements

Each task has three kinds of memory requirements:

- Program Data;
- Internal State;
- Communication queues.



Program Data & Internal State can be allocated:

- On the local LS;
- On the remote Shared Memory.

The communication task might run:

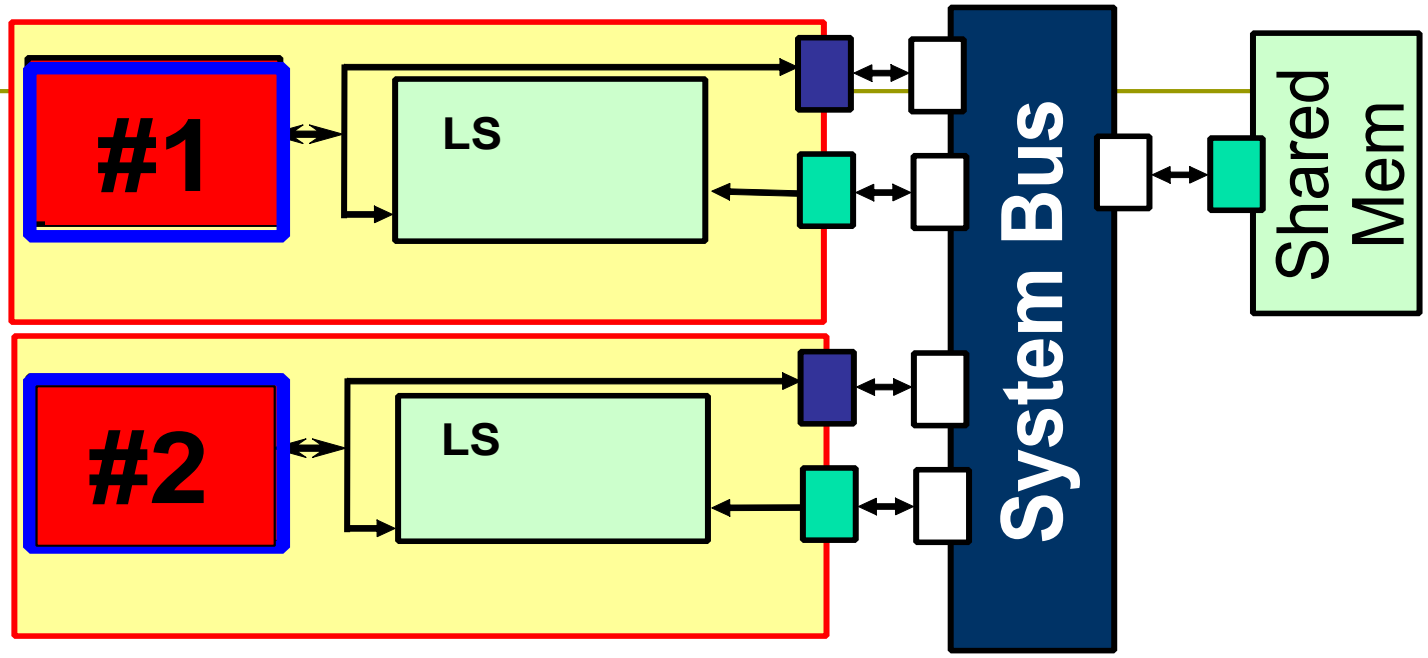
- On the same SPE → negligible communication cost
- On a remote SPE → costly read or write procedure
- On Shared Memory → costly message exchange procedure

- Communication queues in LS → more efficient message passing
  - **Memory size limit!**

Task memory requirements

Each task has three kinds of memory requirements:

- Program Data;
- Internal State;
- Communication queues.



Program Data & Internal State can be allocated:

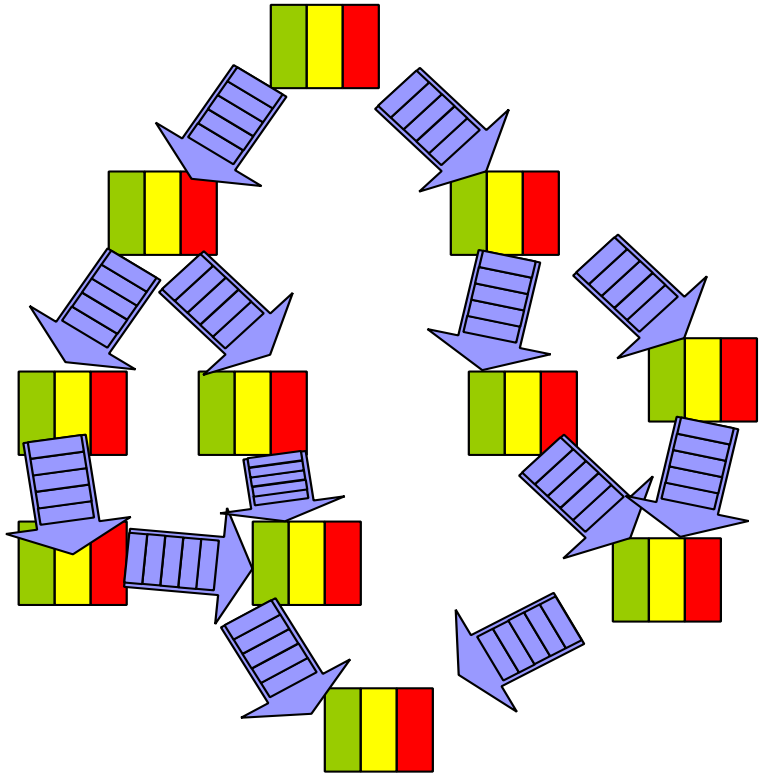
- On the local LS;
- On the remote Shared Memory.

The communication task might run:

- On the same SPE → negligible communication cost
- On a remote SPE → costly read or write procedure
- On Shared Memory → costly message exchange procedure

- Communication queues in LS → more efficient message passing
  - **Memory size limit!**

Task & Application Models



- Task is split into 3 phases:
  - Reading input queues
  - Task Execution
  - Writing output queues
- Tasks communicate through queues
  - FIFO Buffering
  - Semaphore synchronization
- No task preemption

Related Work

Main approaches:

- **Incomplete:**
  - Low computational cost;
  - No guarantees about the quality of the final solution;
- **Complete:**
  - Mainly based on Integer Linear Programming;
  - High computational cost;
  - Suitable for small problems instances;
- **Problem decomposition:**
  - Good way to tackle problem complexity;
  - Divide up the problem into sub-problems & leverage their structures;
  - Mainly heuristic approach.

Our approach

Our Focus:

- **Statically Scheduled Task Graph Applications**

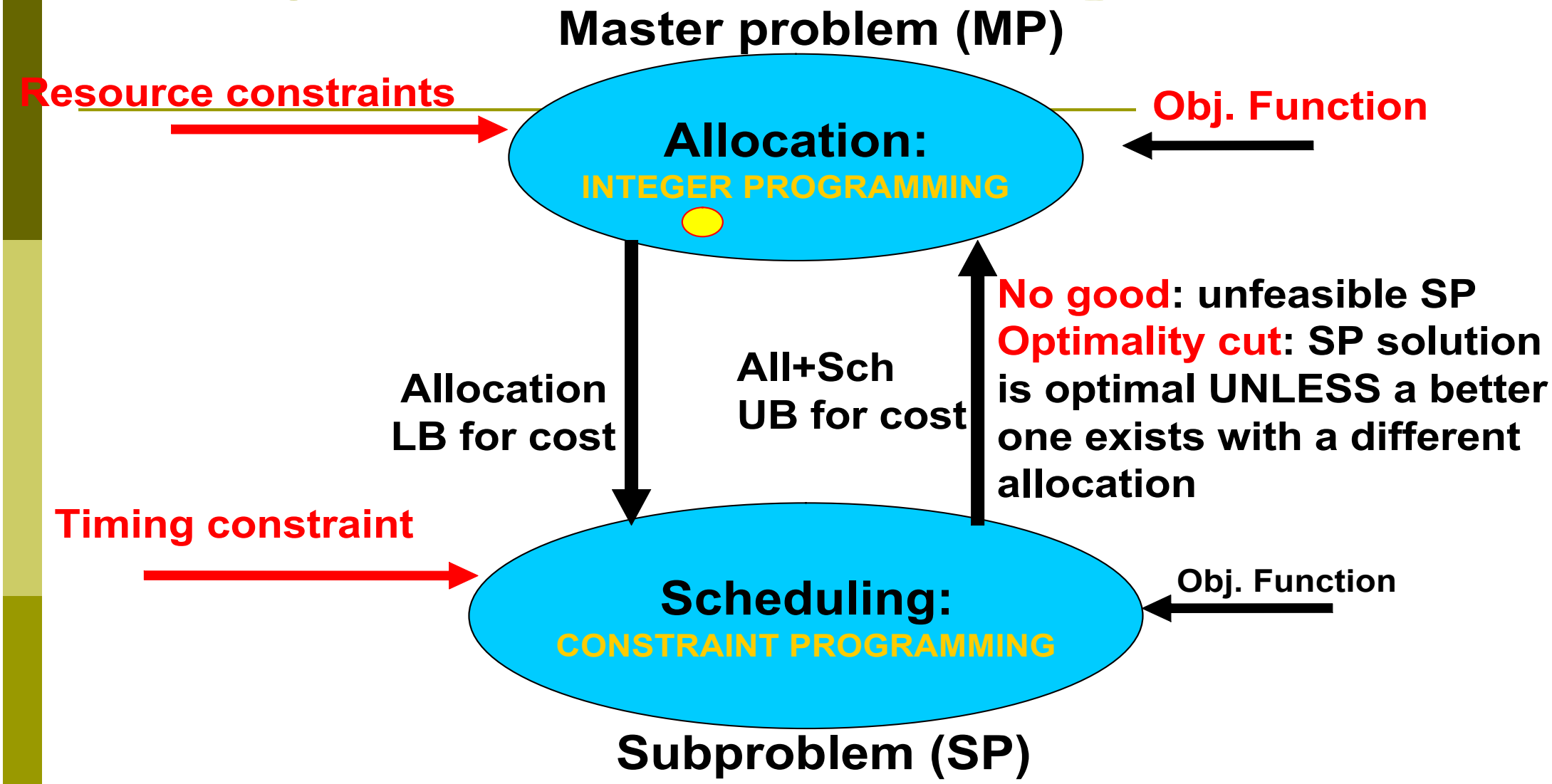
Our Objective:

- **Complete approach to allocation and scheduling;**
- **High computational efficiency w.r.t. commercial solvers;**
- **High accuracy of generated solutions;**

Our Methodology:

- **Problem decomposition;**
- **Allocation Sub-problem:**
  - Integer Programming.
- **Scheduling Sub-problem:**
  - Constraint Programming.

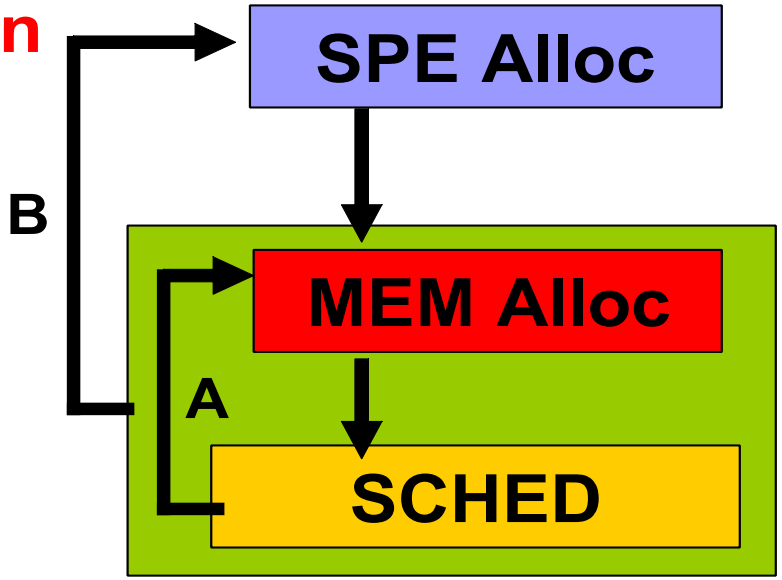
Logic Based Benders Decomposition



Iterations stop when MP becomes unfeasible!

Multi-stage Benders Decomposition

Goal: minimize makespan



- When the SCHED problem is solved, one or more cuts (**A**) are generated to forbid the current memory device allocation and the process is restarted from the MEM stage;
  - if the scheduling problem is feasible, an upper bound on the value of the next solution is also posted.
- When the MEM & SCHED sub-problem ends, more cuts (**B**) are generated to forbid the current task-to-SPE assignment.
- When the SPE stage becomes infeasible the process is over converging to the optimal solution for the problem overall.

SPE Allocation

Given a graph with **n** tasks, **m** arcs and a platform with **p** processing Elements

$$\min z$$

$$s.t.$$

$$z \geq \sum_{i=0}^{n-1} T_{ij} \forall j = 0,\ldots, p-1$$

Needed to express the objective function

$$\sum_{j=0}^{p-1} T_{ij} = 1 \forall i = 0,\ldots, n-1$$

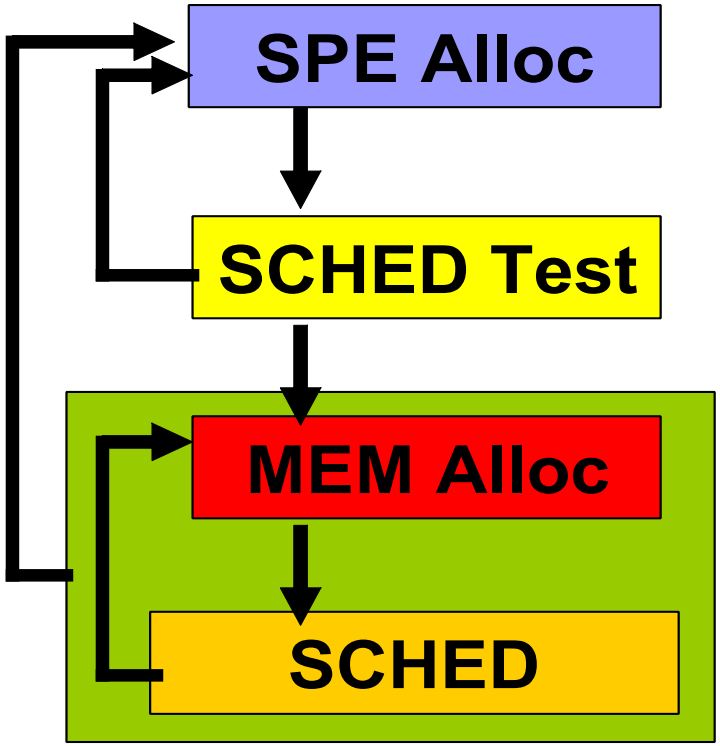
Each task can be assigned to a single PE;

$$T_{ij} \in \{0,1\} \forall i = 0,\ldots, n-1; \forall j = 0,\ldots, p-1$$

The **makespan** objective function depends only on scheduling decision variables.

We adopt an ***heuristic* objective function**:
to spread tasks as much as possible on different SPEs, which often provides good makespan values pretty quickly.
It forces the objective variable z to be
greater than the total number of tasks allocated on any PE.

Schedulability test



- SPE allocation choices are by themselves very relevant:
  - a bad SPE assignment is sufficient to make the scheduling problem unfeasible.
- if the given allocation with **minimal task durations** is already infeasible for the scheduling component, then it is useless to complete it with the memory assignment that cannot lead to any feasible solution overall.

Memory device allocation

$$M_i \in \{0,1\} \qquad \forall i = 0,\ldots,n-1$$
$$W_r \in \{0,1\} \qquad \forall r = 0,\ldots,m-1$$
$$R_r \in \{0,1\} \qquad \forall r = 0,\ldots,m-1$$

$$R_r + W_r \leq 1 \qquad if\ pe(h) \neq pe(k)$$
$$R_r = W_r \qquad if\ pe(h) = pe(k)$$

$$base\_usage(j) = \sum_{\substack{a_r=(-,t_h)\\ pe(h)=j}} comm(r)R_r + \sum_{i=0}^{p-1} mem(i)Mi + \sum_{\substack{a_r=(t_h,t_k)\\ pe(k)=j\\ pe(h)\neq pe(k)}} comm(r)W_r$$

$$\forall j = 0,\ldots,p-1; \qquad \forall i \qquad s.t. \qquad pe(i) = j:$$
$$base\_usage(j) + \sum_{a_r=(-,t_h)} (1-R_r)comm(r) + (1-M_i)mem(i) + \sum_{a_r=(-,t_h)} (1-W_r)comm(r) \leq C_j$$

Memory device allocation

$$M_i \in \{0,1\} \qquad \forall i = 0,\ldots,n-1$$
$$W_r \in \{0,1\} \qquad \forall r = 0,\ldots,m-1$$
$$R_r \in \{0,1\} \qquad \forall r = 0,\ldots,m-1$$

Mi = 1 if task i allocates its computation data on the local memory of the SPE it is assigned to

Wr = 1 if the communication buffer is on SPE pe(h) (that of the producer), Rr = 1 if the buffer is on SPE pe(k) (that of the consumer).

$$R_r + W_r \geq 1$$
$$R_r = W_r \qquad if\ pe(h) = pe(k)$$

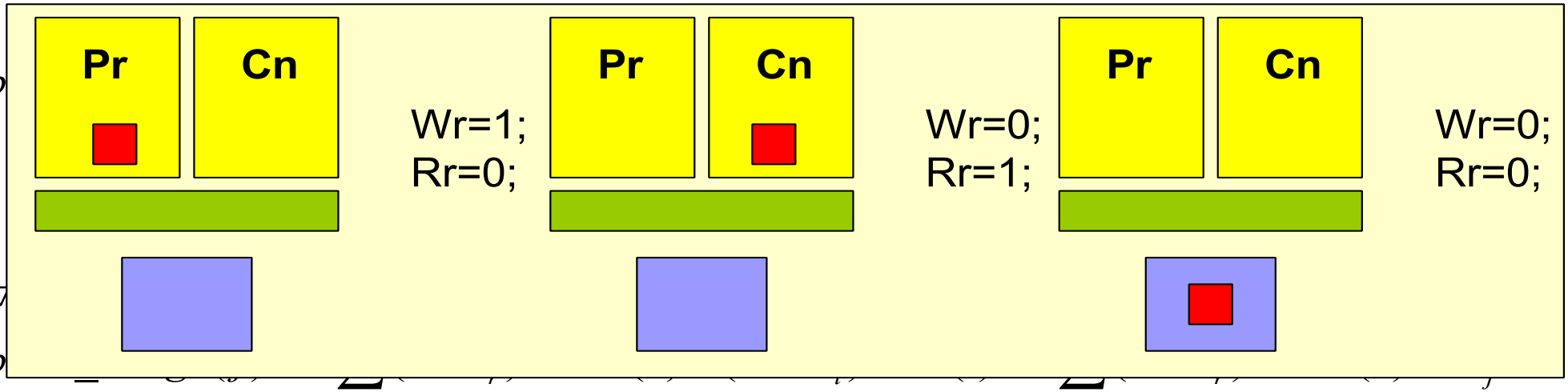
$$base\_usage(j) = \sum_{\substack{a_r=(-,t_h)\\ pe(h)=j}} comm(r)R_r + \sum_{i=0}^{p-1} mem(i)Mi + \sum_{\substack{a_r=(t_h,t_k)\\ pe(k)=j\\ pe(h)\neq pe(k)}} comm(r)W_r$$

$$\forall j = 0,\ldots,p-1; \qquad \forall i \qquad s.t. \qquad pe(i) = j:$$
$$base\_usage(j) + \sum_{a_r=(-,t_h)} (1-R_r)comm(r) + (1-M_i)mem(i) + \sum_{a_r=(-,t_h)} (1-W_r)comm(r) \leq C_j$$

Memory device allocation

$$M_i \in \{0,1\} \qquad \forall i = 0,\ldots,n-1$$
$$W_r \in \{0,1\} \qquad \forall r = 0,\ldots,m-1$$
$$R_r \in \{0,1\} \qquad \forall r = 0,\ldots,m-1$$

$$R_r + W_r \leq 1 \qquad if\ pe(h) \neq pe(k)$$
$$R_r = W_r \qquad if\ pe(h) = pe(k)$$



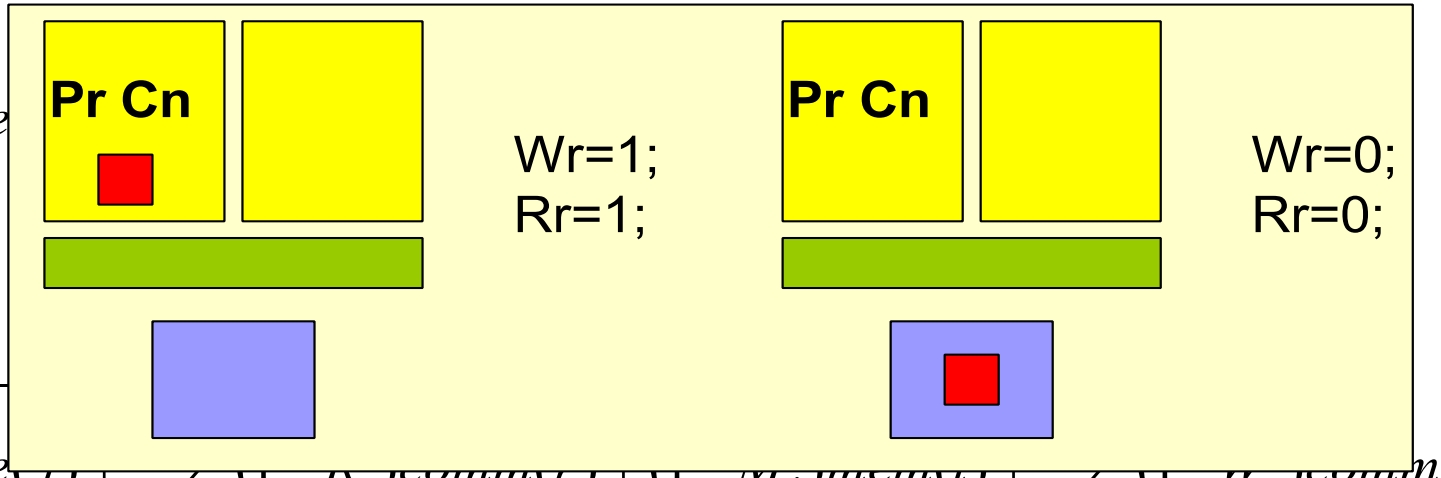
$a_r = (-, t_h)$ $\qquad$ $a_r = (-, t_h)$

Memory device allocation

$$M_i \in \{0,1\} \qquad \forall i = 0,\ldots,n-1$$
$$W_r \in \{0,1\} \qquad \forall r = 0,\ldots,m-1$$
$$R_r \in \{0,1\} \qquad \forall r = 0,\ldots,m-1$$

$$R_r + W_r \leq 1 \qquad if\ pe(h) \neq pe(k)$$
$$R_r = W_r \qquad if\ pe(h) = pe(k)$$

$base\_usage$



$\forall j = 0,\ldots,p -$

$base\_usage(j) + \sum_{a_r=(-,t_h)} (1-R_r)comm(r) + (1-M_i)mem(i) + \sum_{a_r=(-,t_h)} (1-W_r)comm(r) \leq C_j$

Memory device allocation

$$M_i \in \{0,1\} \qquad \forall i = 0,\ldots,n-1$$
$$W_r \in \{0,1\} \qquad \forall r = 0,\ldots,m-1$$
$$R_r \in \{0,1\} \qquad \forall r = 0,\ldots,m-1$$

$$R_r + W_r \leq 1 \qquad if\ pe(h) \neq pe(k)$$
$$R_r = W_r \qquad if\ pe(h) = pe(k)$$

$$base\_usage(j) = \sum_{\substack{a_r=(-,t_h)\\ pe(h)=j}} comm(r)R_r + \sum_{i=0}^{p-1} mem(i)Mi + \sum_{\substack{a_r=(t_h,t_k)\\ pe(k)=j\\ pe(h)\neq pe(k)}} comm(r)W_r$$

$$\forall j = 0,\ldots,p-1; \qquad \forall i \qquad s.t. \qquad pe(i) = j:$$
$$base\_usage(j) + \sum_{a_r=(-,t_h)} (1-R_r)comm(r) + (1-M_i)mem(i) + \sum_{a_r=(-,t_h)} (1-W_r)comm(r) \leq C_j$$

Memory device allocation

$M_i \in \{0,1\} \qquad \forall i = 0,\ldots,n-1$

$W_r \in \{0,1\} \qquad \forall r = 0,\ldots,m-1$

$R_r \in \{0,1\} \qquad \forall r = 0,\ldots,m-1$

mem(i) is the amount of memory required to store internal data of task i;
comm(r) is the size of the communication buffer associated to arc r.

$R_r + W_r$

$R_r = W_r$

The **base_usage(j)** expression is the amount of memory needed to store all data **permanently** allocated on the local device of processor j.

$$base\_usage(j) = \sum_{\substack{a_r=(-,t_h)\\ pe(h)=j}} comm(r)R_r + \sum_{i=0}^{p-1} mem(i)Mi + \sum_{\substack{a_r=(t_h,t_k)\\ pe(k)=j\\ pe(h)\neq pe(k)}} comm(r)W_r$$

$$\forall j = 0,\ldots,p-1; \qquad \forall i \qquad s.t. \qquad pe(i) = j:$$

$$base\_usage(j) + \sum_{a_r=(-,t_h)} (1-R_r)comm(r) + (1-M_i)mem(i) + \sum_{a_r=(-,t_h)} (1-W_r)comm(r) \leq C_j$$

Scheduling subproblem

Each communication buffer must be written before it can be read.

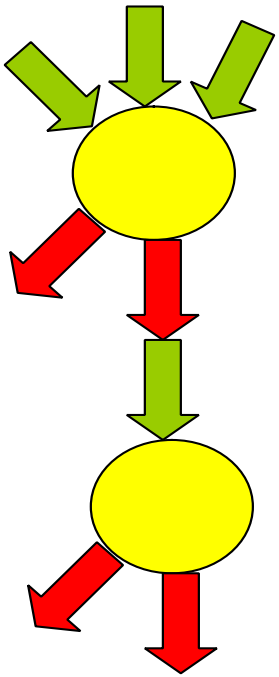
$$\forall l = 0,\ldots,h-2 \qquad end(rd_{rl}) = start(rd_{rl+1})$$

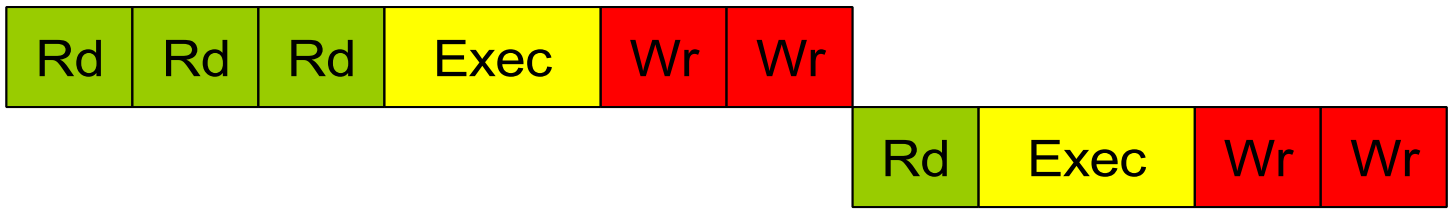
$$end(rd_{rh-1}) = start(exec_i)$$

$$end(exec_i) = start(wr_{rh})$$

$$\forall l = h,\ldots,k-2 \qquad end(wr_{rl}) = start(wr_{rl+1})$$

$$\forall r = 0,\ldots,m-1 \qquad end(wr_r) \leq start(rd_r)$$





Scheduling subproblem

Each communication buffer must be written before it can be read.

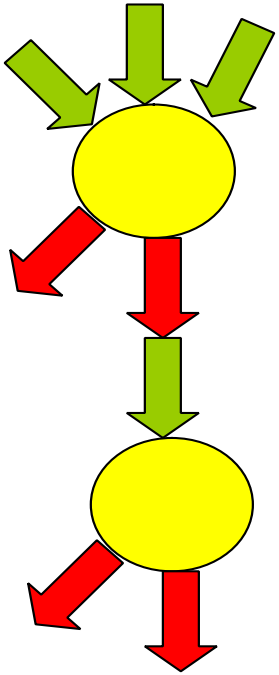
$$\forall l = 0,\ldots,h-2 \qquad end(rd_{rl}) = start(rd_{rl+1})$$

$$end(rd_{rh-1}) = start(exec_i)$$

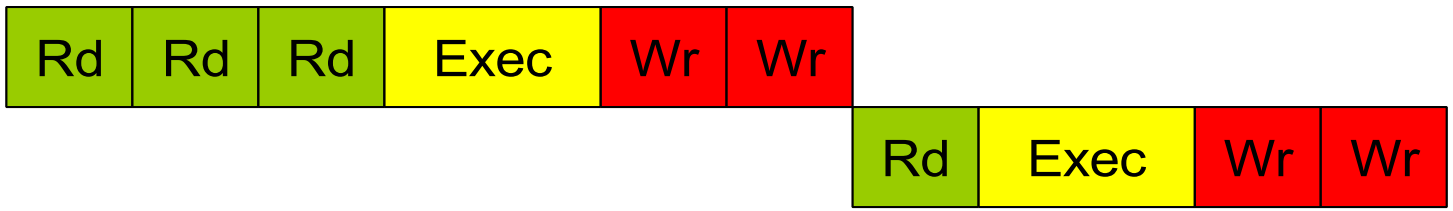
$$end(exec_i) = start(wr_{rh})$$

$$\forall l = h,\ldots,k-2 \qquad end(wr_{rl}) = start(wr_{rl+1})$$

$$\forall r = 0$$

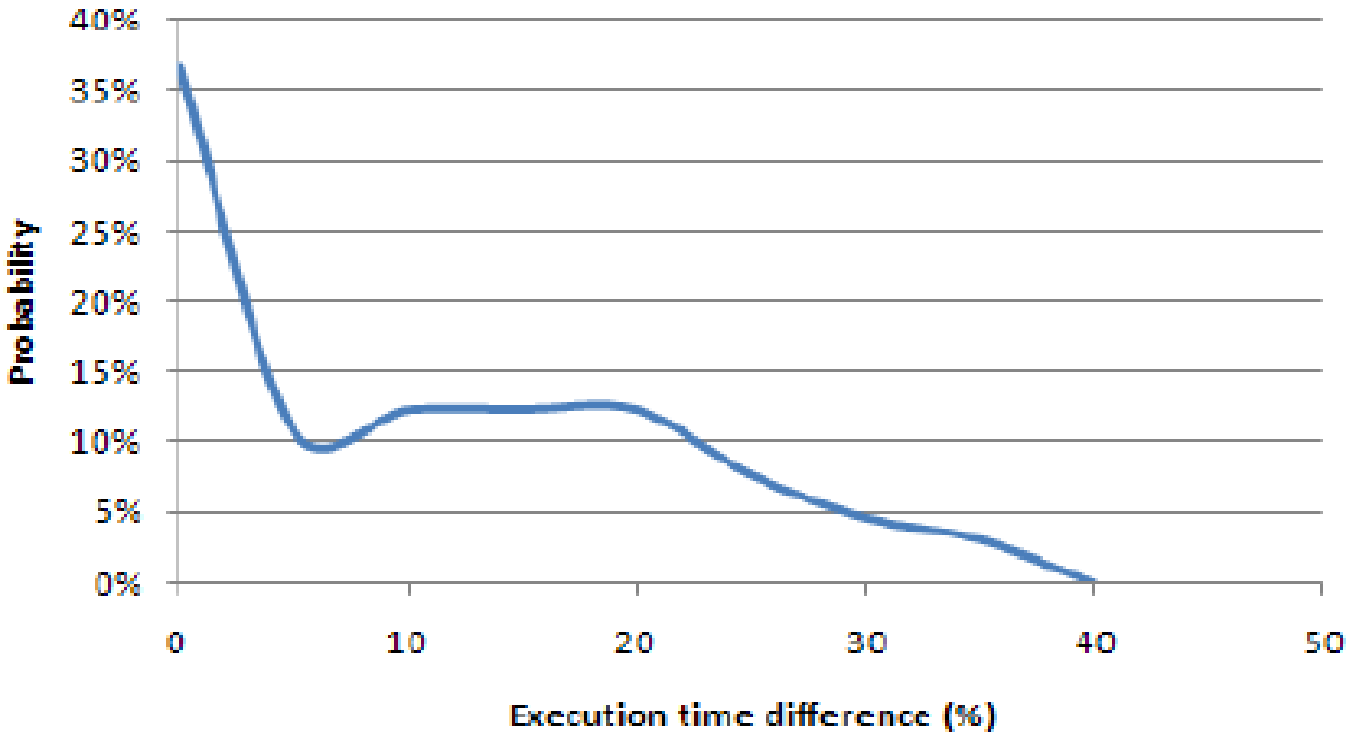


- One activity for each:
  - execution phase (exec)
  - buffer reading/writing operation (rd,wr).
- Task are not preemptive;



Exact vs. Heuristic Scheduler

- Heuristic:
  - RR resource allocation + List scheduling
- Up to 40% makespan difference
- 15% in average



TD vs pure-CP

		CP		TD	
Number of tasks	Number of arcs	time (sec.)	> TL	time (sec.)	> TL
15	9-13	0.01	0	0.31	0
15	14-26	0.02	0	0.62	0
25	30-55	0.10	0	369.66	2
25	56-65	0.05	0	530.96	2
30	47-71	1.25	2	620.13	11
30	73-82	0.12	0	834.45	8

- Set of instances where task durations are not dependant by allocation decisions

- Set of instances where task durations are dependant by allocation decisions

		CP		TD	
Number of tasks	Number of arcs	time (sec.)	> TL	time (sec.)	> TL
10-11	4-11	16.70	0	3.67	0
12-13	8-14	116.92	2	11.19	0
14-15	8-15	81.50	8	10.25	0
16-17	11-17	34.66	11	29.53	0
18-19	13-19	66.47	15	72.56	1
20-21	16-22	400.41	16	248.00	2
22-23	19-26	30.78	18	355.15	3
24-25	20-29	—	20	200.00	9
26-27	23-29	—	20	425.00	6
28-29	25-35	—	20	742.73	9

TD vs BD

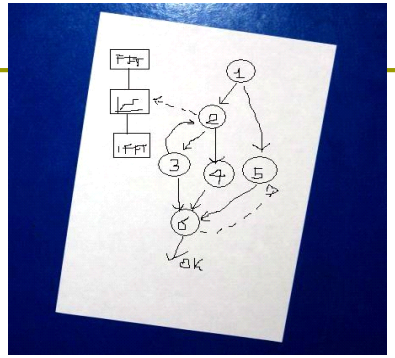
	TD			BD		Timed out		
ntasks	SPE it.	MEM it.	time	PM it.	time	TD ∧ BD	¬ TD ∧ BD	TD ∧ ¬ BD
10-11	12	13	3.95	12	71.10	0	0	0
12-13	17	21	11.59	13	151.38	0	1	0
14-15	19	28	14.78	14	145.19	0	0	0
16-17	29	38	42.61	18	388.89	0	2	0
18-19	46	70	245.17	28	863.00	1	5	0
20-21	70	90	665.35	23	1291.90	4	8	0
22-23	33	69	1304.92	19	1686.00	12	6	1
24-25	29	42	1486.15	8	1623.00	11	4	3
26-27	18	41	1523.50	4	1701.67	12	4	3
28-29	13	19	1800.00	3	1721.00	19	0	1

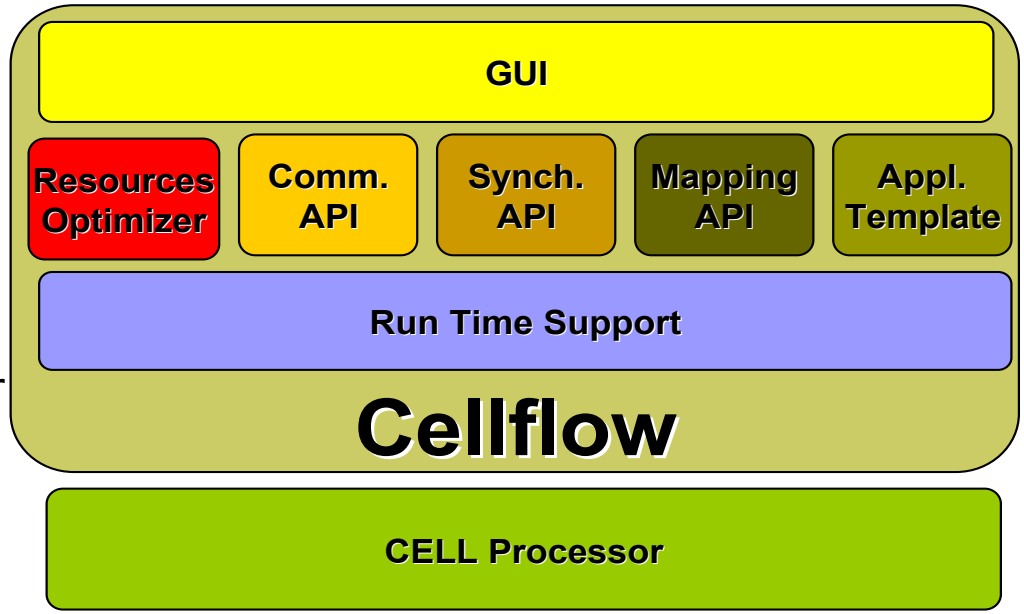
Table 1. Performance tests

- Up to the 20 − 21 group, TD is much more efficient than BD.
- Starting from group 22−23, the high number of timed out instances biases the average execution time.
- TD is doing considerably better until group 24 − 25.
  - After that, most instances are not solved within the time limit by any of the approaches
- TD has a lower execution time, despite it generally performs more iterations than BD:
  - TD works by solving many easy sub-problems
  - BD performs fewer and slower iterations.

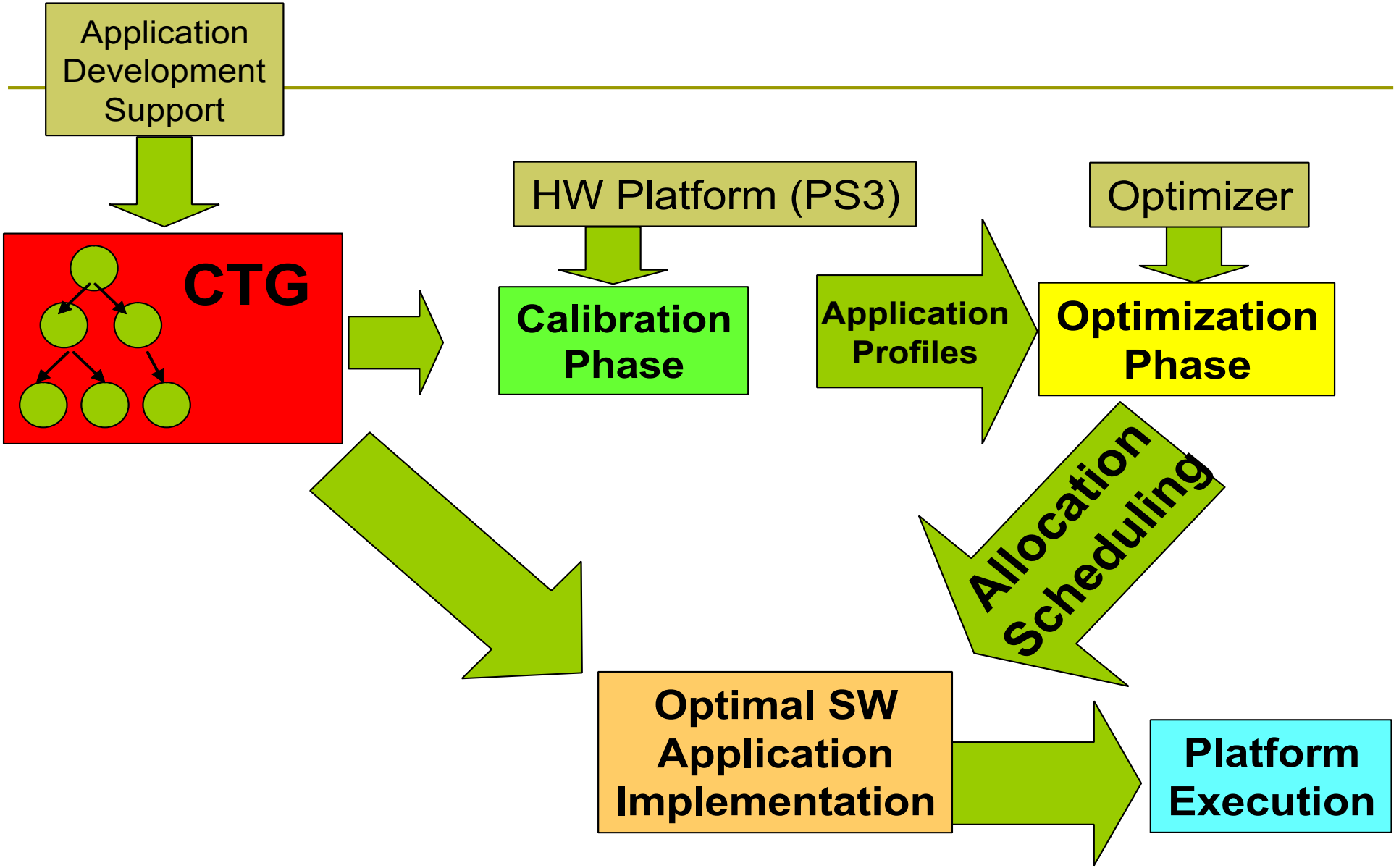
Cellflow

- A software development **toolkit** to help programmers in software implementation
- Starting from a high level task and data flow graph, software developers can easily and quickly build their **application infrastructure**.
- Programmers can intuitively translate high level representation into C-code using our facilities and libraries
- The main goals are:
  - guarantees on **high performance** and **constraint satisfaction**;
  - **predictable** application execution after the optimization step.

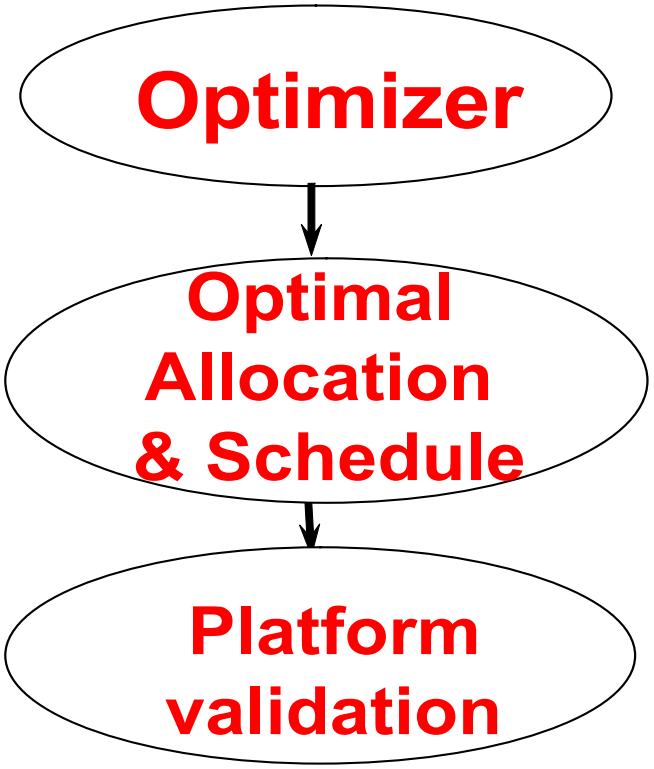


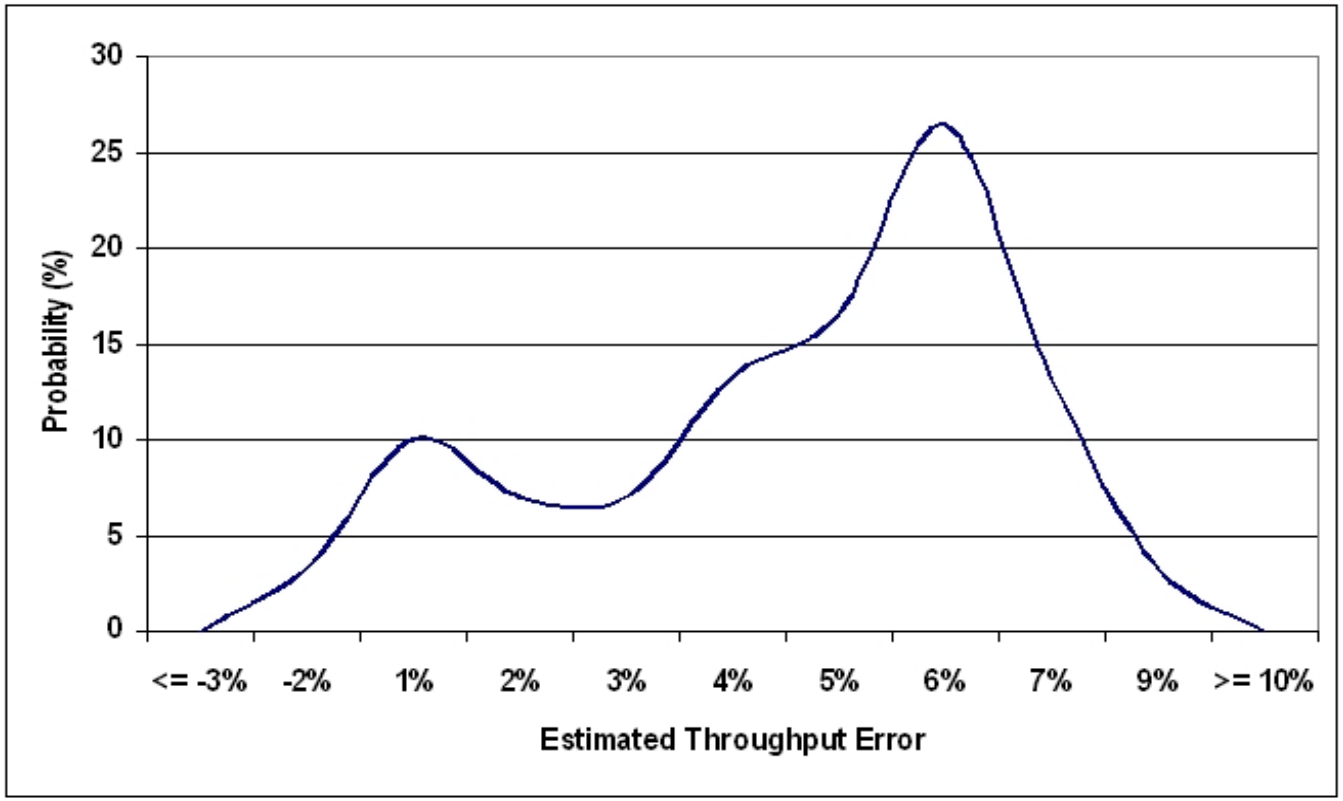


Application Development Flow



Validation of optimizer solutions





- Throughput comparison between the predicted by the optimizer and the real one;
- MAX error lower than 10%;
- AVG error equal to 4.8%, with standard deviation of 2.41;

Ongoing & Future Work

- Extensions
  - Scheduling parallel DMA activity
  - Full dataflow (FIFO buffers) support
- Interaction with high level tools:
  - Parallelization tools
  - Data distribution tools
- Dynamic resource management
  - Hybrid approaches (offline+online)

Thank you!