Parallel heterogeneous Branch and Bound algorithms for multi-core and multi-GPU environments

Defended by Imen Chakroun

Under the supervision of Pr. Nouredine Melab

October 2010 - June 2013

Université Lille1 Sciences et Technologies

LIFL Laboratoire d'Informatique Fondamentale de Lille

cnrs dépasser les frontières

inria informatiques mathématiques

Outline

- Context and objectives
- Contributions
  - GPU-accelerated parallel B&B - Application to FSP
  - Heterogeneous B&B combining GPU and multi-core
  - HB&B@GRID: a distributed heterogeneous B&B
- Conclusions and Future Works

Exact Combinatorial Optimization

- Minimize or maximize an objective function $f(\Omega): \Omega \mapsto R$

→ Find $x^* \in \Omega$ such that $f(x^*) = (\min \text{ or } \max) f(x) \,/\, x \in \Omega$.

→ Find optimal configuration(s) among a finite set Ω of candidate solutions.

- **High-dimensional** and **complex** optimization problems exist in many areas of industry
  - Task allocation, job scheduling, network routing, cutting, packing, etc.

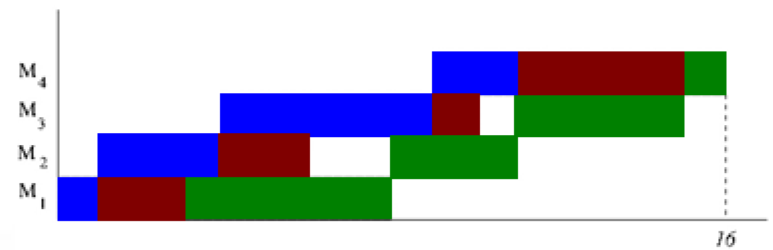
The permutation Flowshop Scheduling Problem (FSP)

- Scheduling a pool of N jobs on a set of M machines
  - Jobs have to be processed on the machines on the **same order**.
  - A machine $M_k (k = 1,2,\ldots,M)$ can handle **at most one job at a time**.

- Objective
  - Find a **processing order** on each M_k such that the time required to complete all jobs is **minimized**.

	M_1	M_2	M_3	M_4
J_1	5	3	4	1
J_2	2	2	1	4
J_3	1	3	5	2

Processing Times



Optimal Solution

Branch and Bound Algorithms (B&B)

- B&B is a search algorithm based on an implicit enumeration of all candidate solutions.
- Exploration is performed by building a **tree**.

- **Branching** = splitting into sub-problems.
- **Bounding** = computing lower/upper bounds.
- **Selection** = choosing the fragment node to explore.
- **Pruning** = eliminating unpromising branches.

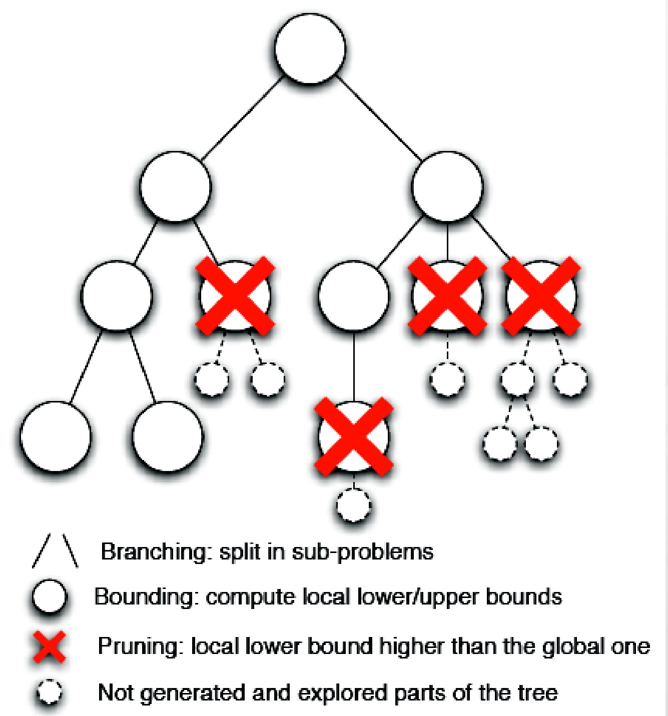
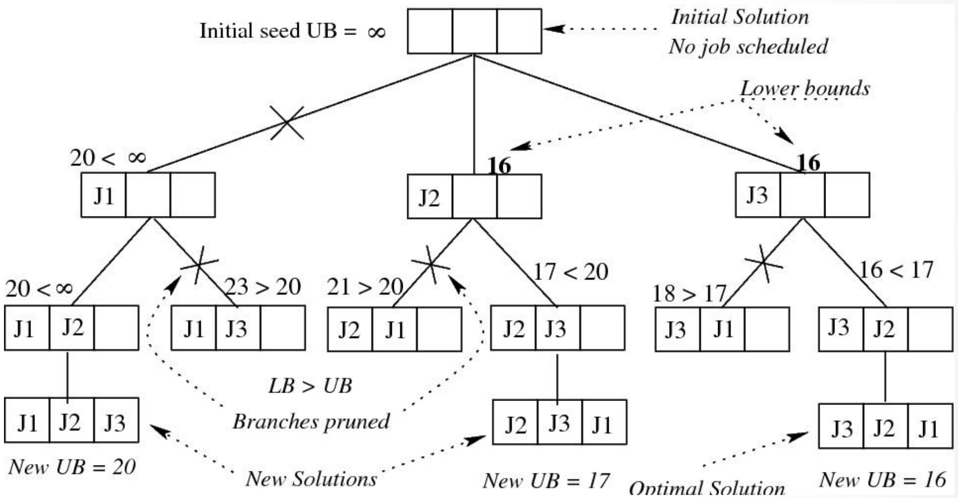


Illustration on FSP

- Scheduling 3 jobs on 4 machines
  → 3! = 6 candidate solutions
- For 50 jobs on 20 machines
  → 50! candidate solutions !!!!

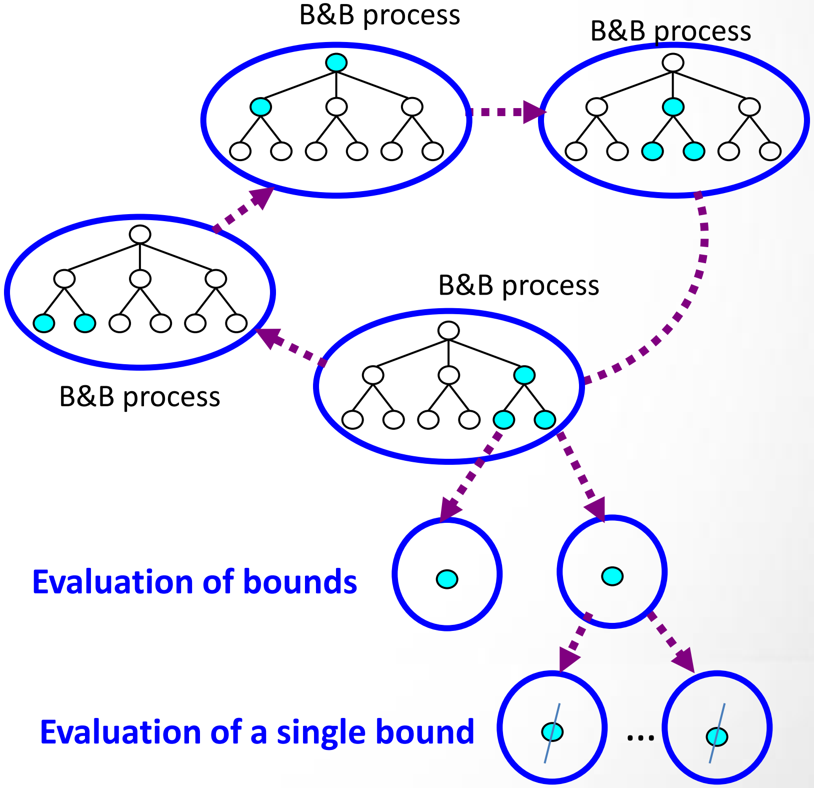


- Efficient bounding is **not sufficient for large instances**
  - Several years of computation for Ta056 [Mezmaz *et al.*, IPDPS 2007]

→**Massive parallelism** is required to deal with **very large** instances.
→ All the parallelism levels provided through today's heterogeneous platforms [Top500] should be exploited

Parallel models for B&B

- Parallel B&B models [Melab 2005]
  - Multi-parametric parallel model
  - **Parallel tree exploration model**
  - **Parallel bounding model**
  - Parallel evaluation of a single solution/bound



- Parallel bounding model
  - Highly data parallel and attractive for SIMD architectures (e.g. GPU)
- Parallel tree exploration model
  - Massively parallel but highly irregular ➔ challenging for GPU + multi-core

Parallel Branch and Bound algorithms

- The implementation of the models is influenced by the target execution platform [Roucairol 1996, Bader 2004]

- Many architecture-oriented contributions have been proposed:
  - Networks or clusters of workstations [Quinn 1990, Tschöke 1995, Aida 2002].
  - Shared memory machines [Mans 1995, Casado 2008].
  - Graphics Processing Units [Carneiro 2011, Lalami 2012].

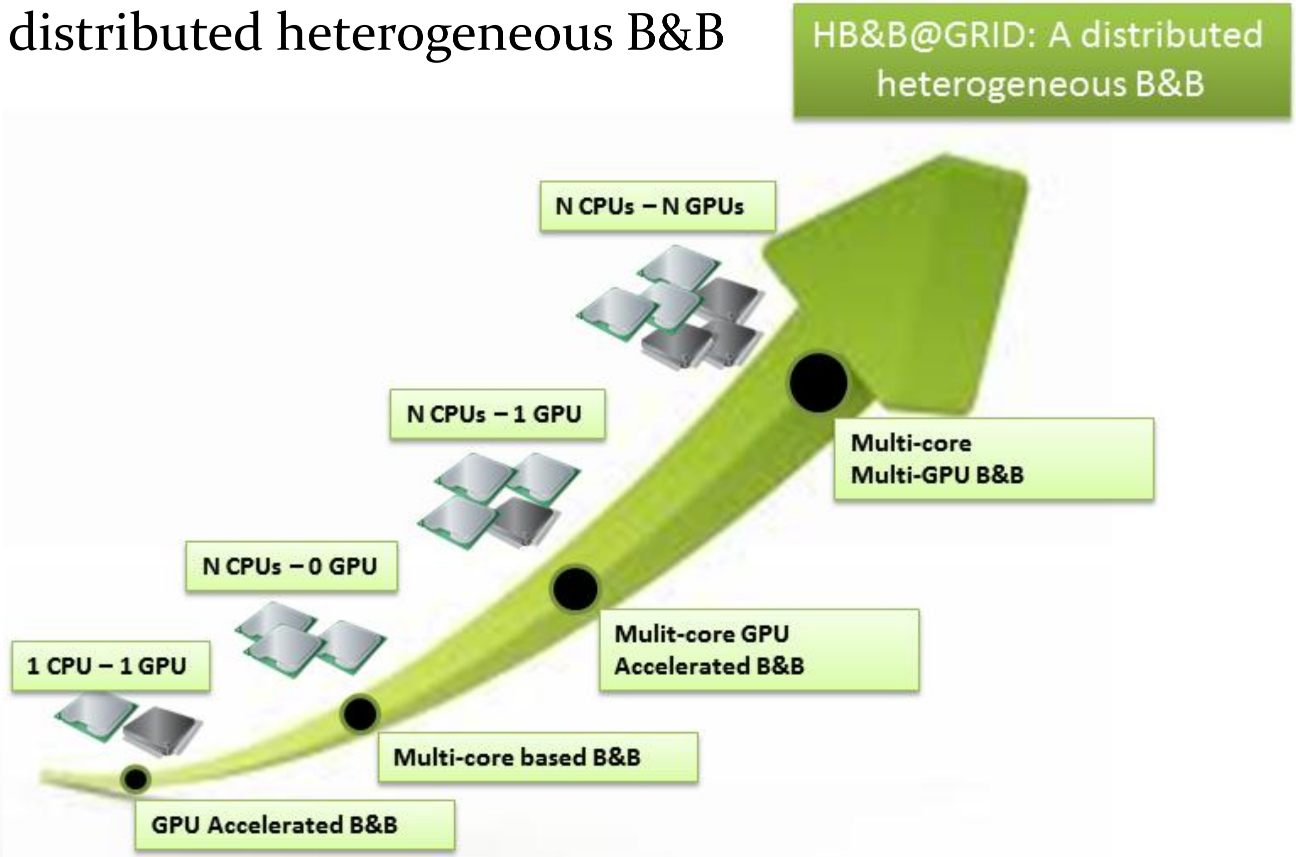
- Few existing works related to B&B on GPU
  → Among the two pioneering works
- No works on parallel heterogeneous (GPU + multi-core) B&B

Objectives

- Revisit the design and implementation of B&B algorithms for **GPU-enhanced multi-core environments.**
- The **heterogeneous** B&B should be **portable** in a transparent way on **laptops, workstations, clusters and computational grids.**
- Dealing with challenging issues related to:
  - **GPU computing:** thread divergence, hierarchical memory optimization, CPU-GPU data transfer, …
  - **Multi-core computing:** synchronization, …
  - **Hybrid computing combining GPU and multi-core:** work sharing, …
  - **Heterogeneous cluster and grid computing:** portability, scalability, …

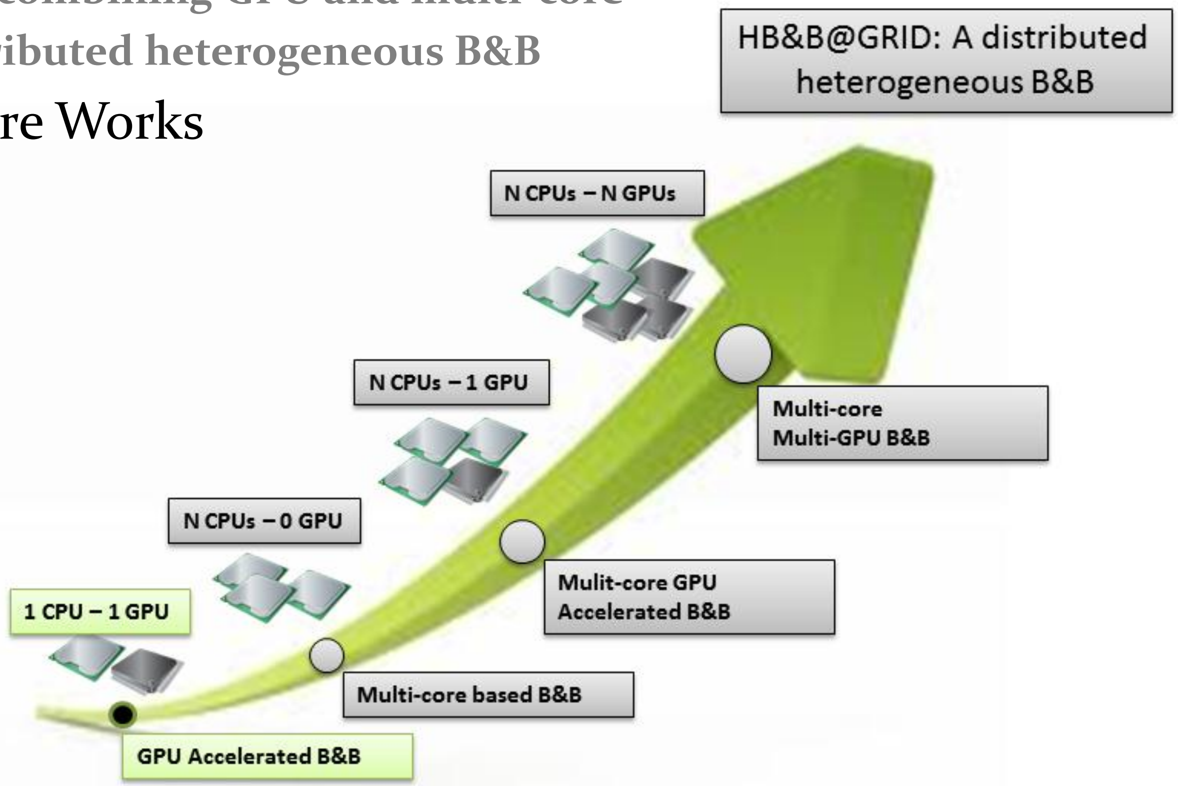
Contributions

- GPU-accelerated parallel B&B Application to FSP
- Heterogeneous B&B combining GPU and multi-core
- HB&B@GRID: a distributed heterogeneous B&B

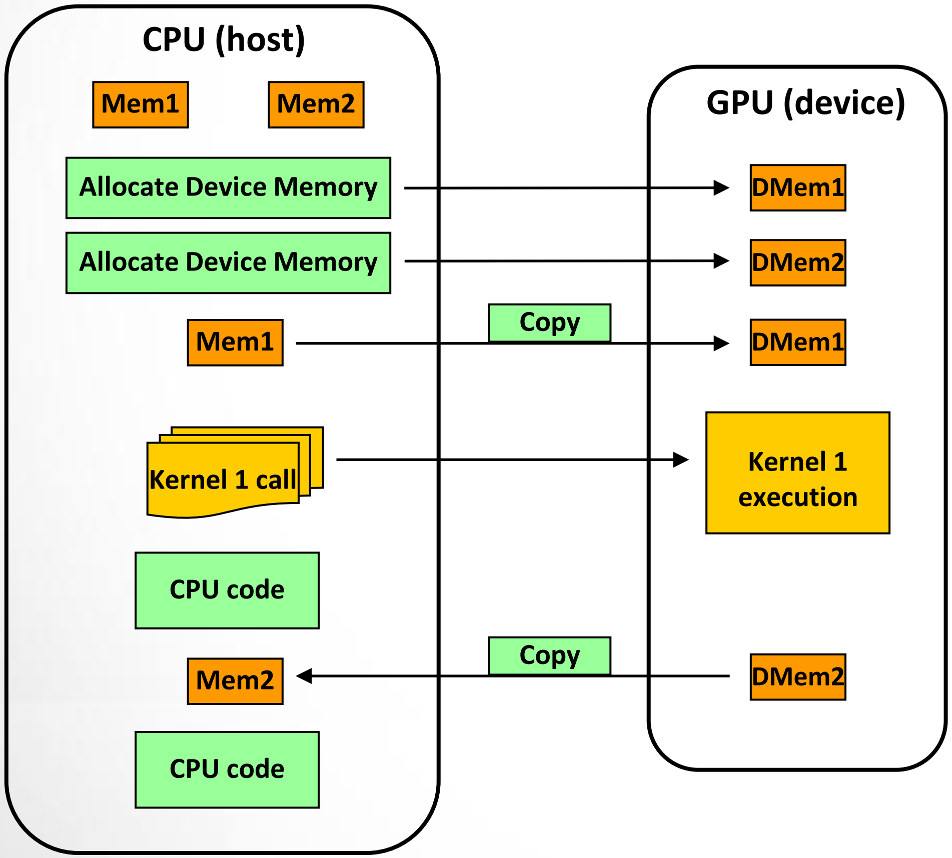


Outline

- Context and objectives
- **Contributions**
  - **GPU-accelerated parallel B&B - Application to FSP**
  - **Heterogeneous B&B combining GPU and multi-core**
  - **HB&B@GRID: a distributed heterogeneous B&B**
- Conclusions and Future Works



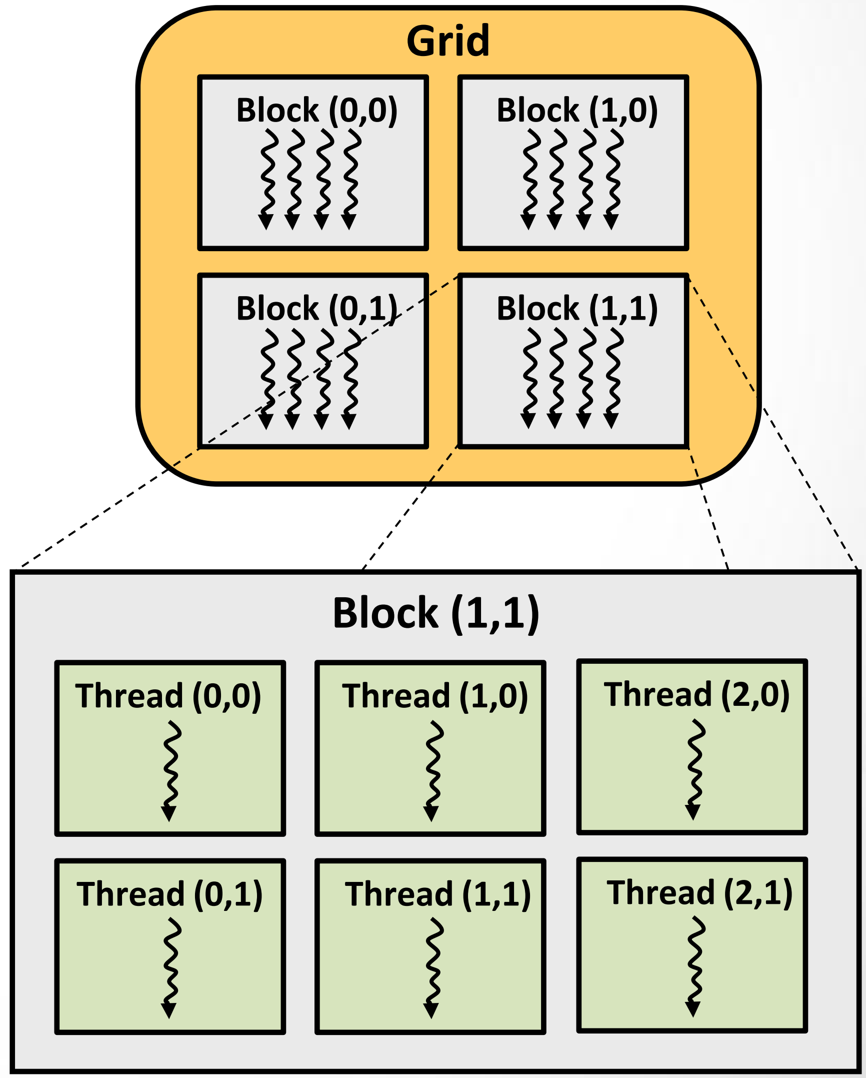
General GPU-based parallel model



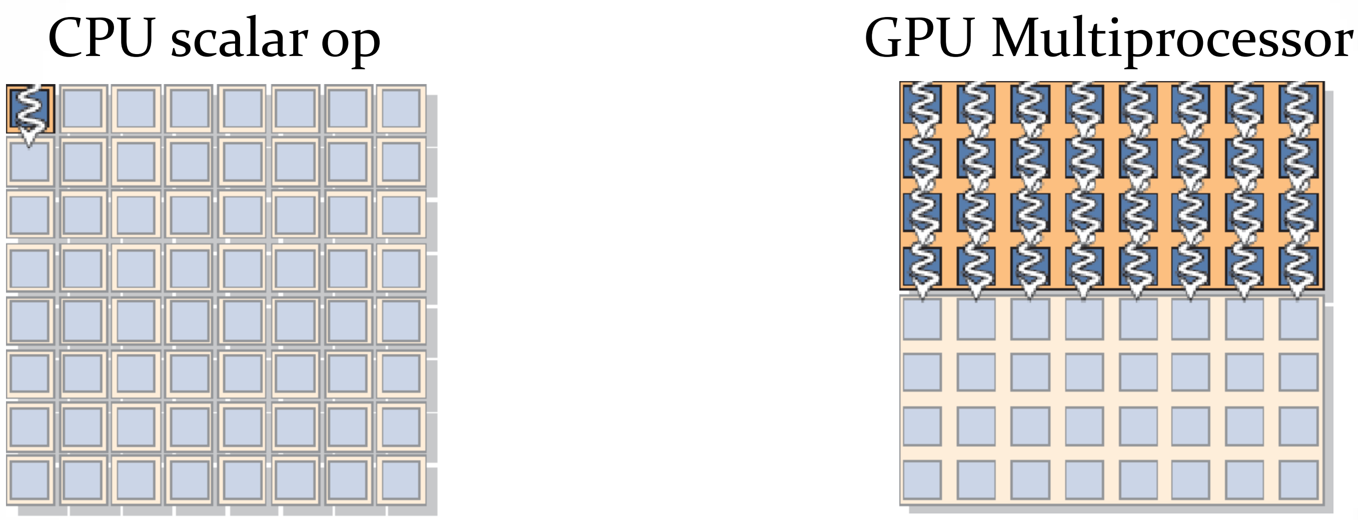
- Data must be transferred between CPU and GPU *via* the PCI bus express …
- … many data transfers might become a bottleneck for performance [Mahmoudi 2013]

Programming model: thread-based SPMD

- Kernel execution is invoked by CPU over a **compute grid …**
    - … split in a set of thread blocks
- All threads within the grid run the **same program**
    - Single Program Multiple Data



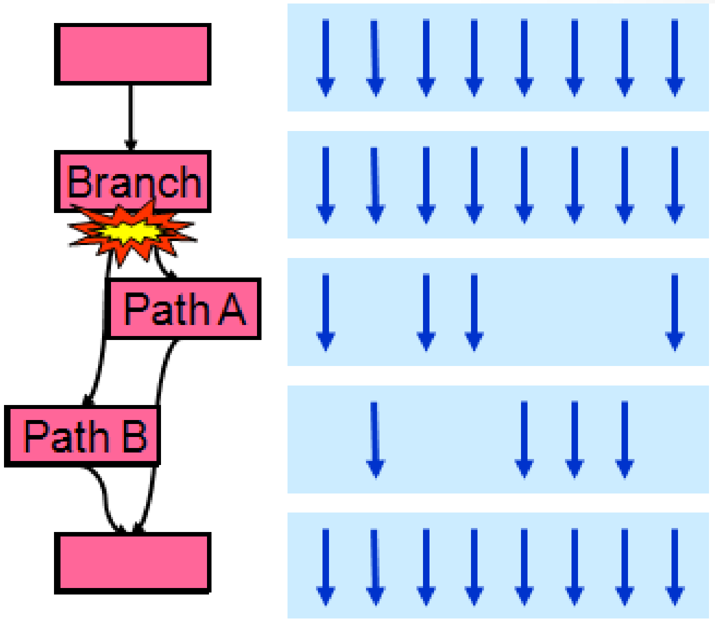
Execution model: SIMD



- GPU architectures are based on hyper-threading
- Fast context switching …
  - … between warps when stalled (*e.g.* an operand is not ready)
  - … enables to minimize stalls with little overhead
- Single instruction executed on multiple threads (SIMT) grouped into warps (32 threads)

Thread divergence issue

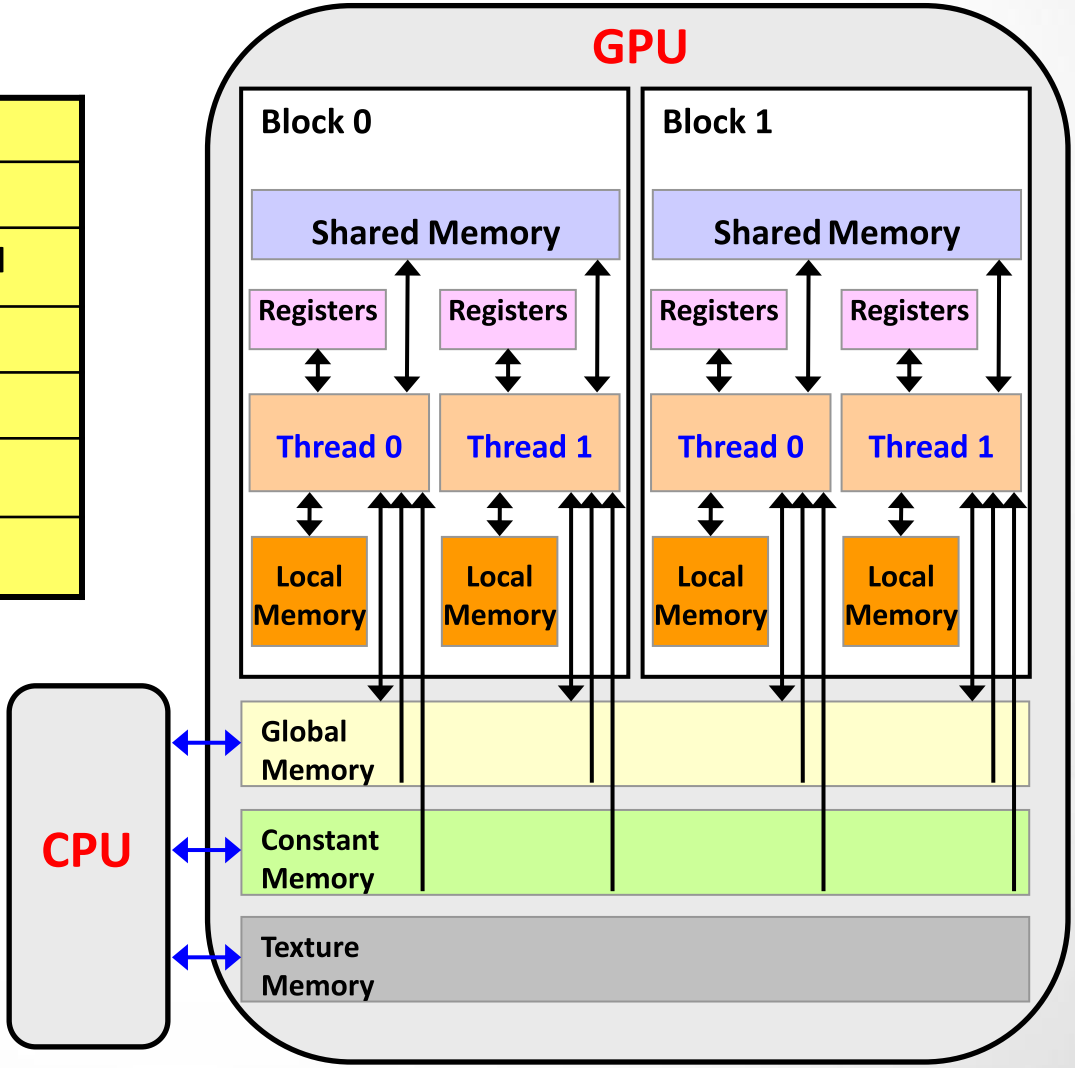
- If threads of a warp diverge *via* a data-dependent conditional branch …
  - … the different branch paths (threads) are executed **serially**
- When all paths complete threads **converge back to the same execution path**



→**Full efficiency achieved when all threads agree on their execution path**

Hierarchical memory levels

Memory type	Access latency	Size
Global	Medium	Big
Registers	Very fast	Very small
Local	Medium	Medium
Shared	Fast	Small
Constant	Fast (cached)	Medium
Texture	Fast (cached)	Medium

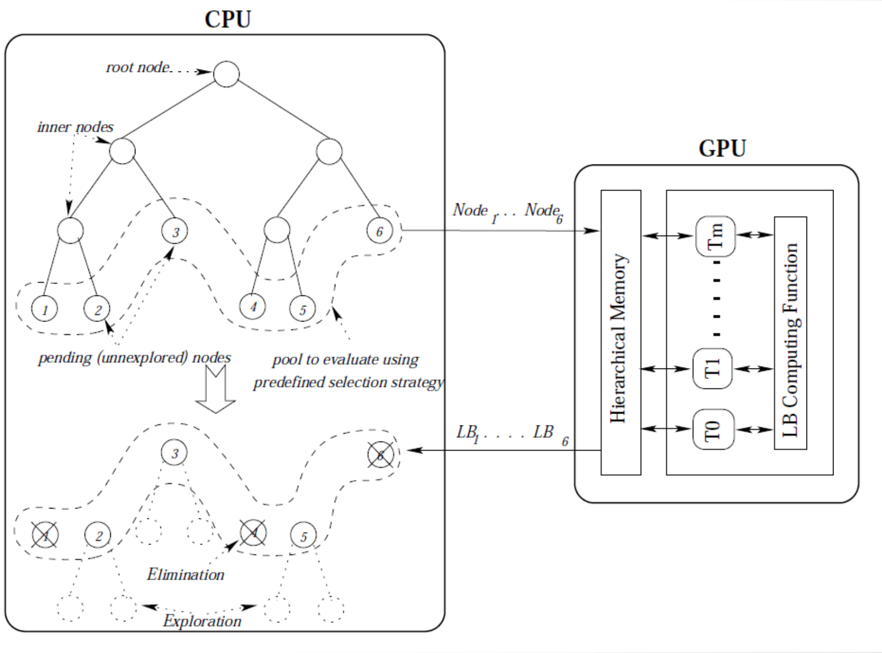


- Different hierarchical memory levels …
  - … with different sizes and latencies

GPU-accelerated B&B based on parallel bounding *(GB&B)*

- Bounding consumes on average **97% - 99%** of the B&B execution time

- Generation (selection and branching) and pruning of the subproblems …
  - … are performed **on CPU**
- Evaluation of their lower bounds …
  - … is executed on the **GPU** device



Thread divergence in FSP

- Lower bound proposed by [Lenstra *et al.* 1978] based on [Johnson 1954]
- Divergence related to the control flow instructions (*if-then-else*, *for*, …)

- When the first thread executes *else branch*, the remaining threads are disabled

```
if( pool[thread_idx].index_start != 0 )
    time = TimeMachines[1] ;
else
    time = TimeArrival[1] ;
```

- All threads finish the first 10 iterations together + two passes for the 90 other iterations

```
for(int k = 0 ; k < pool[thread_idx].index_start; k++)
   jobTime = jobEnd[k] ;
```

Reducing thread divergence

Branch refactoring = rewrite the conditional instructions into an uniform code

```
if( pool[thread_idx].limit1 != 0 )
   a = TimeMachines[1] ;
else
   a = TimeArrival[1] ;
```

$$
\begin{array}{l} if \ \ (x! = 0) \\ \quad a = b[1]; \\ else \\ \quad a = c[1]; \end{array}
\quad \Rightarrow \quad
\begin{array}{l} \text{if} \ \ (x! = 0) \\ \quad a = b[1] + 0 * c[1]; \\ \text{else} \\ \quad a = 0 * b[1] + c[1]; \end{array}
$$

$$\Rightarrow \qquad a = f(x) * b[1] + g(x) * c[1];$$

$$\text{where f(x)} = \begin{cases} 0 & \text{if} \quad x = 0 \\ 1 & \text{if} \quad x! = 0 \end{cases} \quad and \quad g(x) = \begin{cases} 1 & \text{if} \quad x = 0 \\ 0 & \text{if} \quad x! = 0 \end{cases}$$

```
int coeff = __cosf(pool[tid].limit1);
a = (1 - coeff) * TimeMachines[1] + coeff * TimeArrival[1];
```

I. Chakroun, M.Mezmaz, N. Melab, and A.Bendjoudi. **Reducing thread divergence in a GPU-accelerated branch-and-bound algorithm. Concurrency and Computation: Practice and Experience** vol 25, 8, 1121-1136, 2013 - John Wiley & Sons.

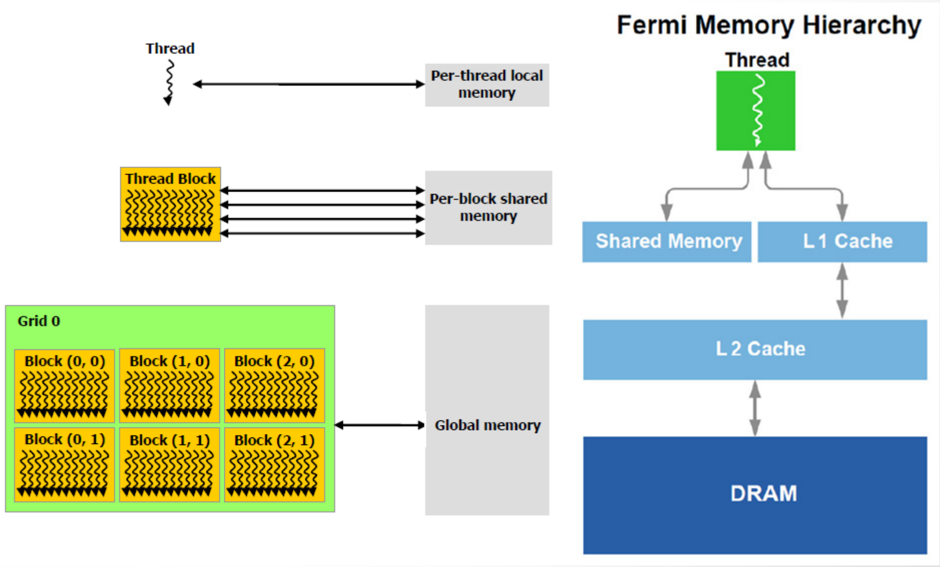
Memory access optimization

Mapping of the LB data structures on the memory hierarchy of the GPU

- Complexity analysis: The LB function uses 6 data structures with different sizes and access latencies/frequencies

Matrix	Size	Number of accesses
PTM	$n \times m$	$n' \times m \times (m-1)$
LM	$n \times \frac{m \times (m-1)}{2}$	$n' \times \frac{m \times (m-1)}{2}$
JM	$n \times \frac{m \times (m-1)}{2}$	$n \times \frac{m \times (m-1)}{2}$
RM	m	$m \times (m-1)$
QM	m	$\frac{m \times (m-1)}{2}$
MM	$m \times (m-1)$	$m \times (m-1)$

- GPU memories have different sizes and access latencies



Memory access optimization (Cont.)

- Memory size issue
  - Nvidia Tesla T10 Processor with 16 KB of shared memory

Nb.Jobs × Nb.machines	JM	LM	PTM	RM, QM
200 × 20	38.000 (38KB)	38.000 (76KB)	4.000 (4KB)	20 (0.04KB)
100 × 20	19.000 (19KB)	19.000 (38KB)	2.000 (2KB)	20 (0.04KB)

- JM and LM do not fit into the shared memory which size is limited

→Which data must be put in the shared memory to get the best performance?
→different explored scenarii.

N. Melab , I. Chakroun, M. Mezmaz and D. Tuyttens. **A GPU-accelerated Branch-and-Bound Algorithm for the Flow-Shop Scheduling Problem.** 14th IEEE International Conference on Cluster Computing, Cluster'12 (2012)".

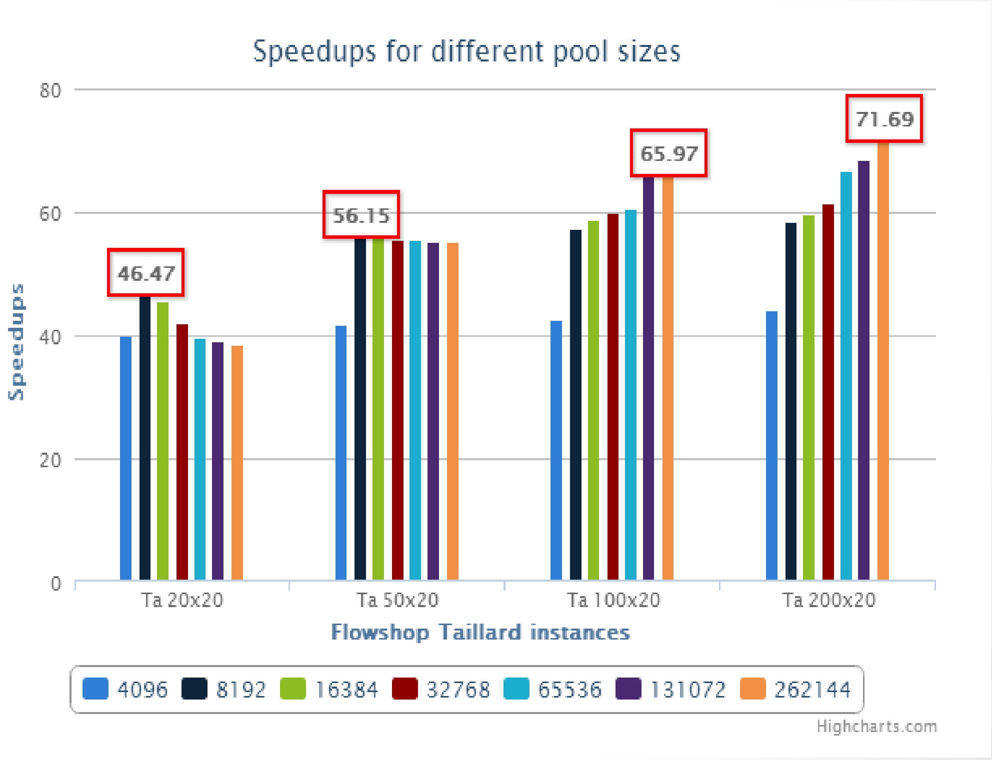
Experimental settings

- Taillard's FSP benchmarks proposed in [Taillard 1993]
  - Optimal solutions of some of these instances are still not known
  - Divided into groups of 10 instances defined by the same N and M
  - Only the instances **where M = 20 and N = 20, 50, 100, 200 are considered**
    - Instances with **M = 5 and 10 are easy to solve**
    - Instances with **500 jobs do not fit in the memory of the GPU**

- Software and hardware platforms
  - C-CUDA 4.0.
  - CPU host: Intel Xeon E5520 quad-core 64-bits server
  - GPU device = Nvidia Tesla C2050
    - 448 CUDA cores, warp size = 32, global memory = 2.8GB, configurable shared memory (16 KB or 48 KB)

Performance evaluation of *GB&B*

- Speedup up to **71.69** is obtained
- Speedup grows with the size of the problem
- The **pool size has a high impact** on the performance of GB&B



→ the pool size has to be tuned dynamically with respect to the problem being solved.

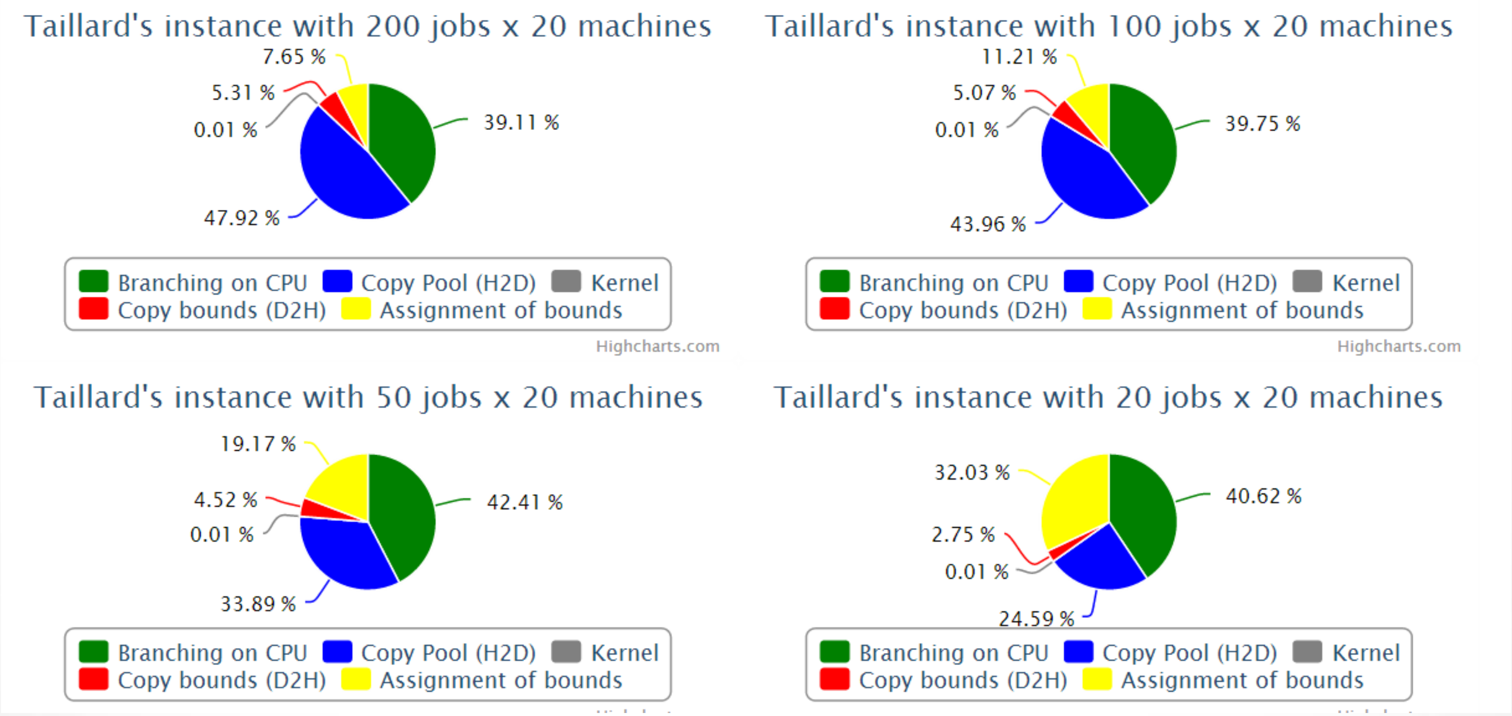
Performances evaluation of *GB&B* (Cont.)

- Thread reduction approaches
  - Best reported speedup is **77.46**
  - Divergent branches on average 3 times less
- Data access optimization
  - PTM on shared memory …. enhancement of 19%
  - JM on shared memory …. acceleration of 97.83
  - JM and PTM on shared memory ….. 23% of improvement compared to the scenario with no data access optimization

→ Speedup of 100 is reached for large problem instances

Performance analysis of GB&B

- Time consumption analysis of the different steps …



→ First, the pool size should be dynamically tuned

→Second, the CPU-GPU transfer latency should be minimized

An adaptive selection operator:
Adaptive Selection Heuristic (ASH)

- Calibrates the two parameters …
  - **Maximum number of threads and blocks**

- The ASH heuristic
  - The number of threads per block is **doubled repeatedly …**
  - … until the **maximum number of active threads** allowed on the device is reached
  - A **downwards and an upwards** search around the best pool size found so far

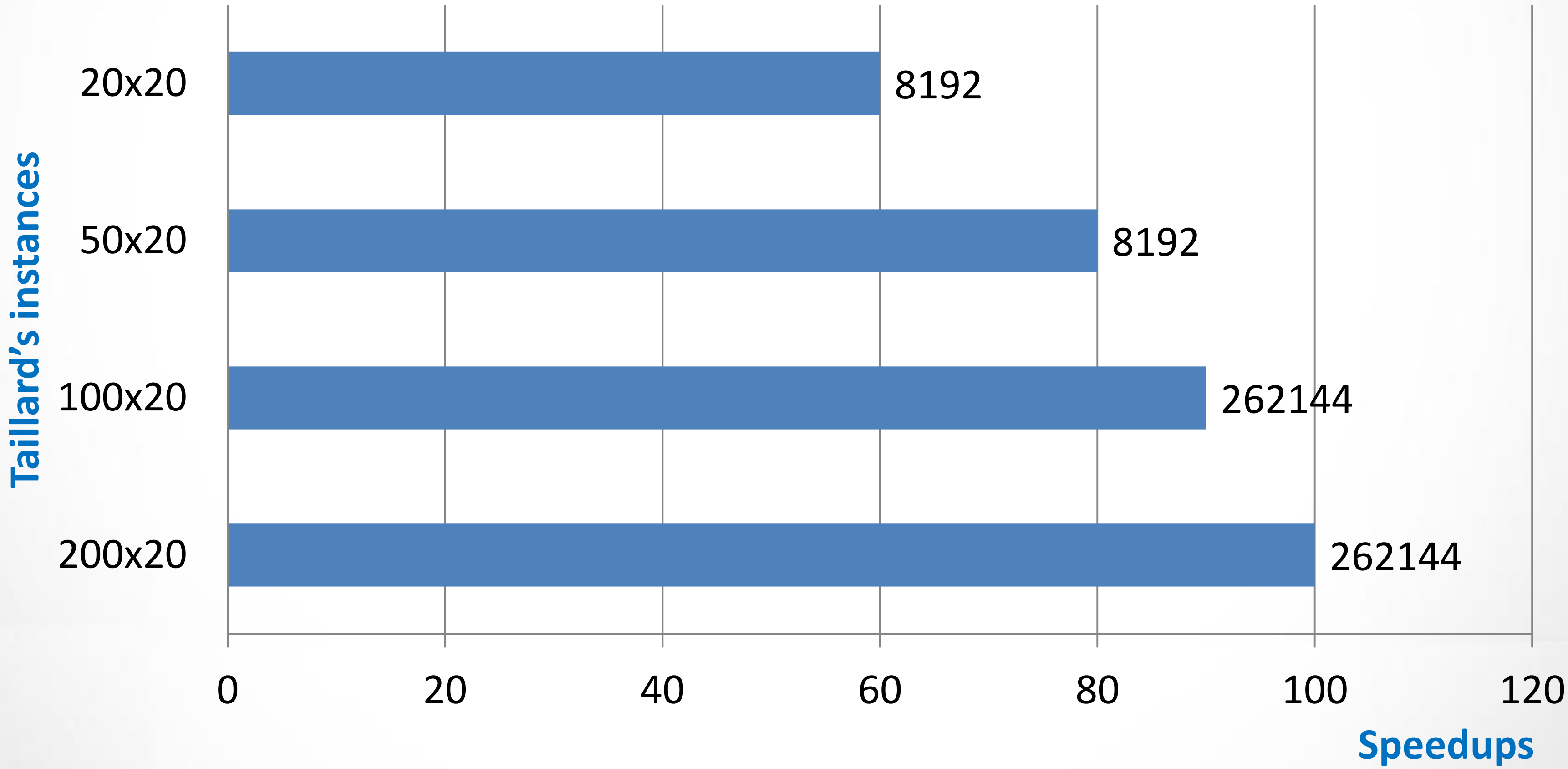
Algorithm 3 Template of the Adaptive Selection Heuristic (ASH).

```
Data: nb_iterations;
Result: best_number_of_threads
max_nb_threads = Detect_GPU_Characteristics();
nb_threads = Use_Cuda_Occupancy_Calculator();
nb_blocks := Get_Number_Of_Multiprocessors();

while not_empty_tree() do
    while pool_size ≤ nb_threads × nb_blocks do
        take_sub_problem();
    end
    Iteration pre-treatment on host side;
    Kernel evaluation on GPU;
    Iteration post-treatment on host side;
    if ( iteration % nb_iterations = 0 ) and ( (nb_threads × nb_blocks) ≤
    max_nb_threads) then
        if Is_best_pool_improved() then
            best_number_of_threads = nb_threads × nb_blocks ;
        end
        nb_blocks := nb_blocks * 2 ;
    end

    else
        Compute_Binary_Search_Around_Best_Pool() ;
    end
    iteration := iteration + 1 ;
end
```

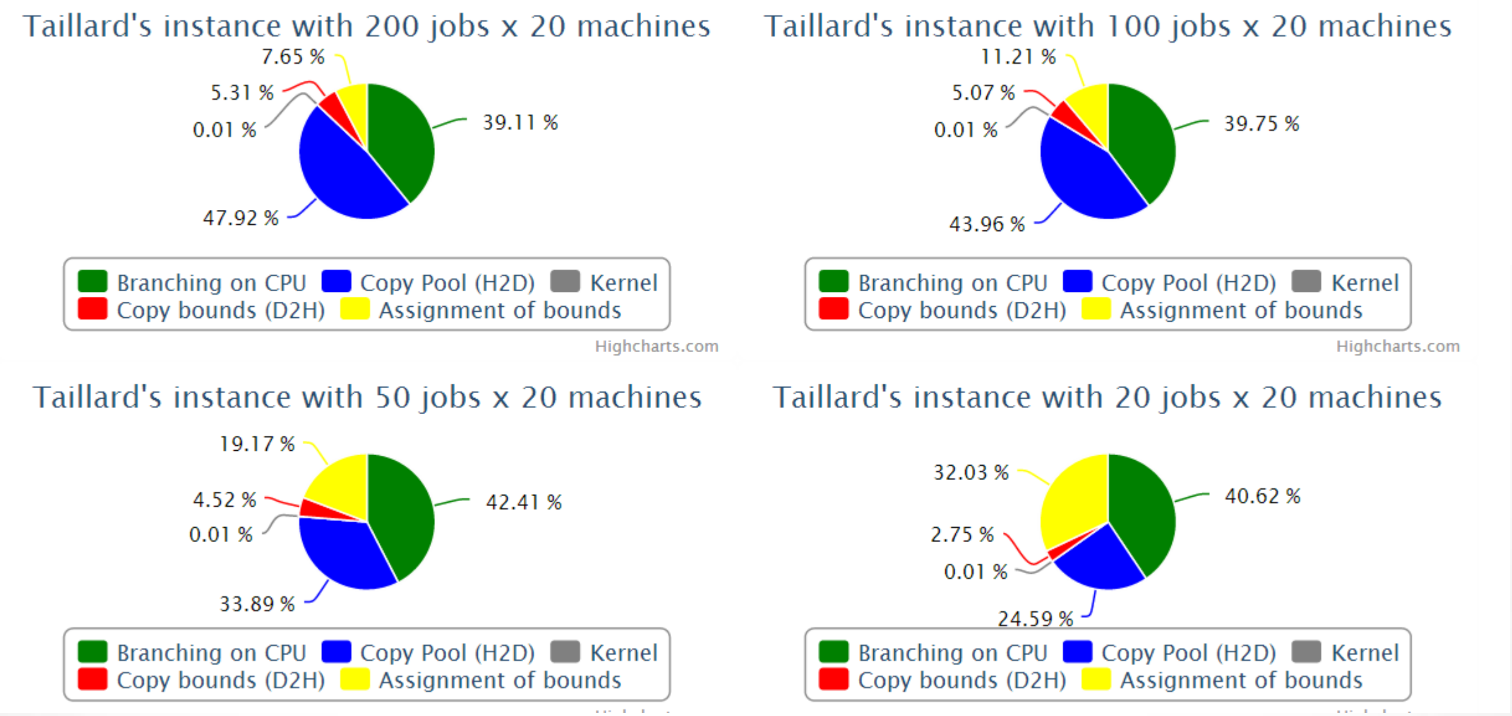
Performance evaluation of the ASH heuristic



Same speedups obtained with the same best pool sizes of the static version

Performance analysis of GB&B

- Time consumption analysis of the different steps …



→ First, the pool size should be dynamically tuned

→Second, the CPU-GPU transfer latency should be minimized

GPU-based parallel tree exploration

- Moving to GPU the branching and pruning operators
- Even if they consume less time than the bounding operator, they allow to reduce the data transfer between CPU and GPU

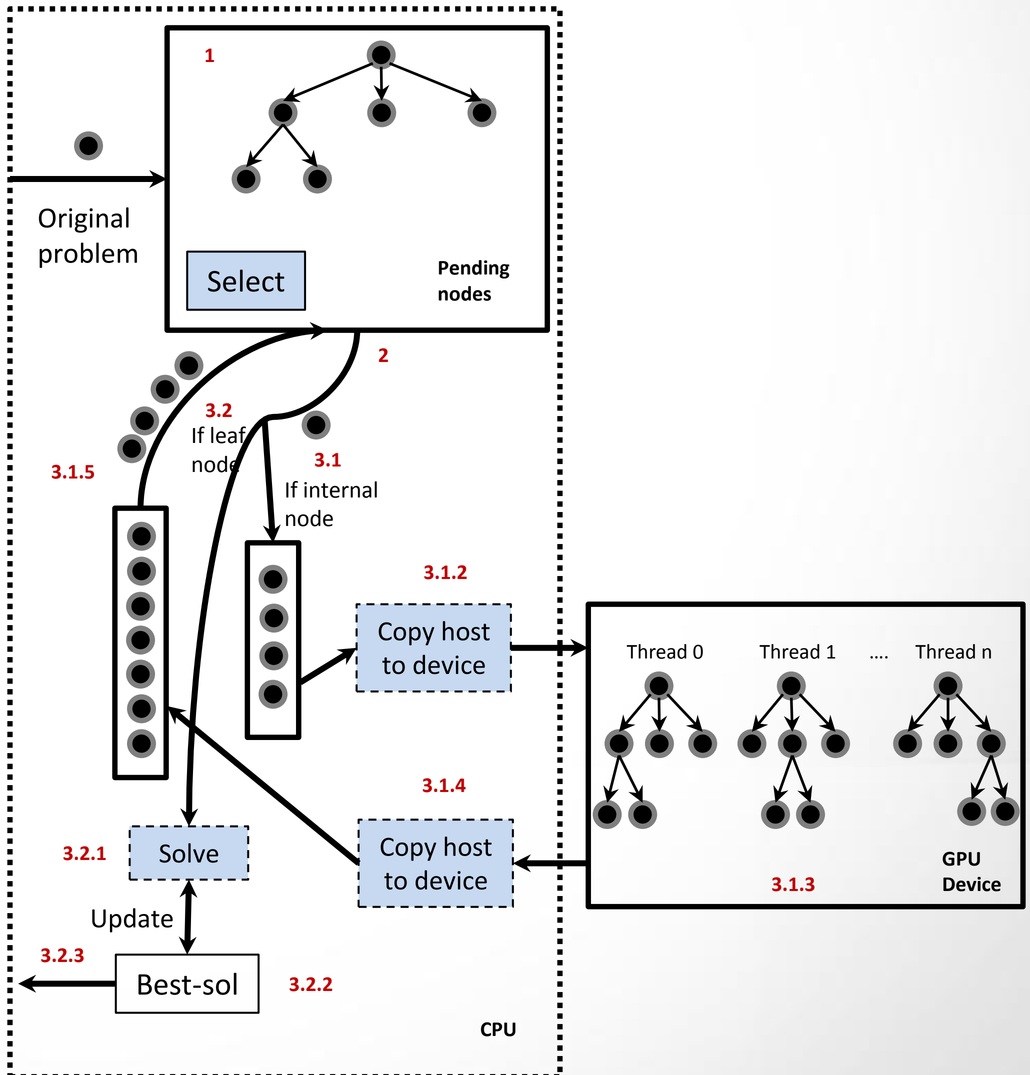
→Higher performances should be achieved

- Two proposed and studied approaches
  - **Multiple-nodes driven approach**
    - Each thread performs in parallel B&B operators on multiple tree nodes
  - **Single-node driven approach**
    - Consecutive data-parallel kernels where threads compute in parallel the same amount of work on a single tree node

I. Chakroun and N. Melab. Operator-level GPU-accelerated Branch and Bound algorithms. **International Conference on Computational Science, ICCS 2013.** Barcelona, Spain, June 5-7, 2013.

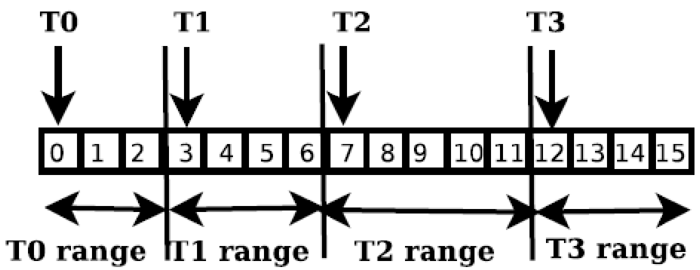
Multiple nodes-driven approach

- Divide the search space into **disjoint sub spaces**
- To **each thread is assigned a node** from the selected nodes.
  - **Mapping strategy (next slide)**
- Each thread builds its **local search tree by applying the branching, bounding and pruning operators** to its assigned node
- Resulting nodes are moved back to CPU. Other nodes are **deleted on the device memory**



Multiple nodes-driven approach (Cont.)

- Mapping strategy
  - Thread i branches the node i of the pool, thread i+1 branches the nodes i+1, and so on.
  - Each thread writes the nodes it generates in an allocated range: **position of thread i depends on the number of children of the thread i-1**
- Challenging issues
  - Uncoalesced memory accesses
  - Thread divergence due to the **high irregular nature of the tree**

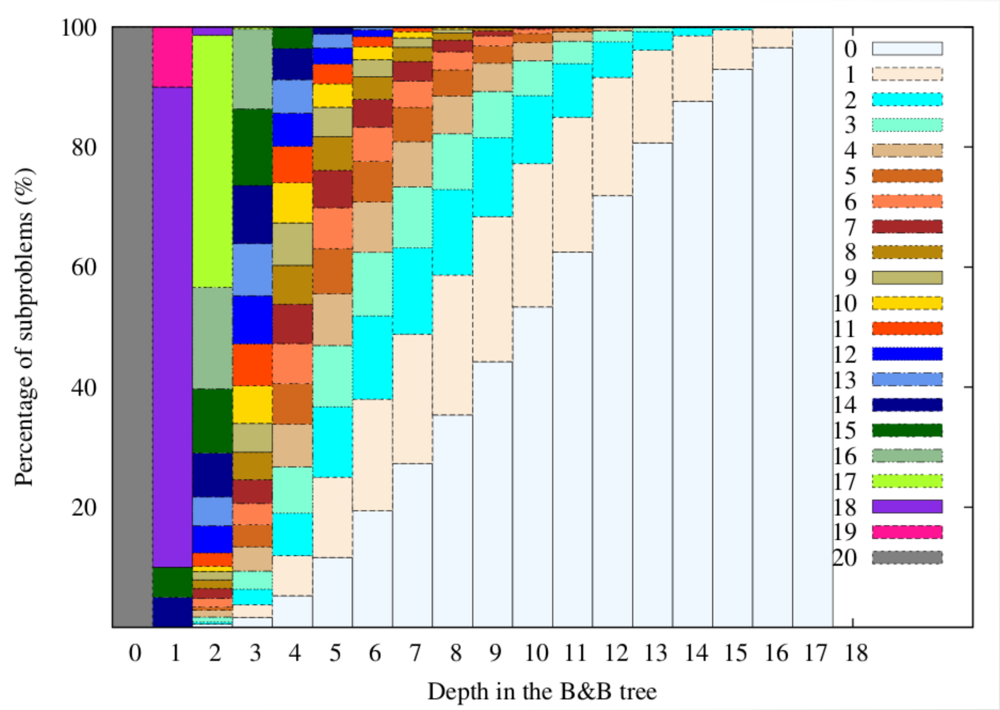


Example of uncoalesced access in the multiple-nodes driven approach

How much irregularity in the B&B? Intra-instance irregularity

- At the same level (depth 10) …
  - 53% of nodes have 0 children, 23% have 1 child, 11% have 2 children, etc.
- At different levels
  - 41% of nodes have 17 children at depth 2 *vs.* 1% at depth 3

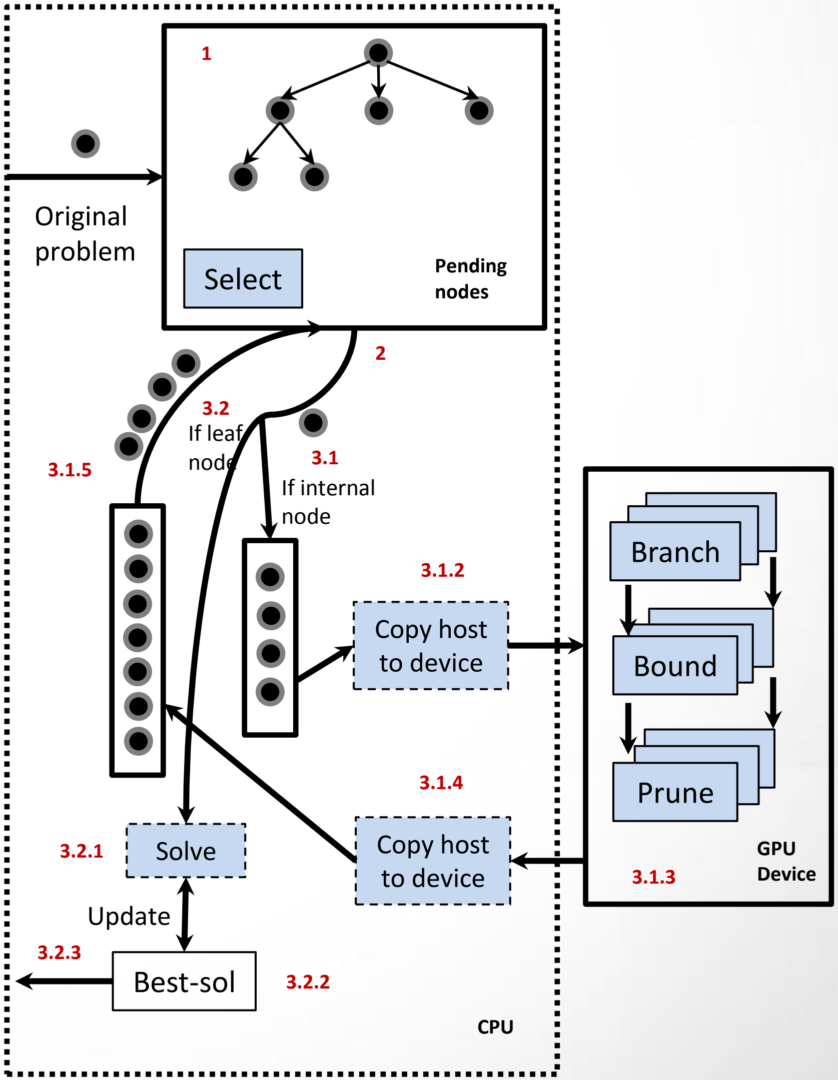
Structure of the search tree for the instance Ta023: 20 jobs on 20 machines



→ Solution: single node-driven approach

Single node-driven approach

- **The same amount of work on each tree node**
- Branching kernel
  - Each thread **generates a unique child** and inserts it into a global pool
- Bounding kernel
  - The pool is **kept in the global memory** and used by the bounding kernel
  - Each thread **assigns a lower bound to a unique node**
- Pruning kernel
  - The evaluated pool is **kept in the device memory** and used by the pruning kernel

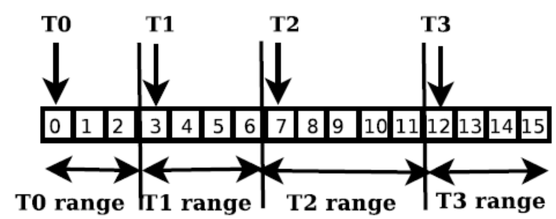


Single node-driven approach (Cont.)

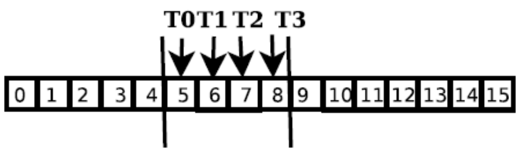
- Mapping strategy
  - Thread i writes the generated node i in the position i
- All threads execute **exactly the same flow of instructions**

➔Prevents from thread divergence

- **The approach prevents from the uncoalesced accesses** to the global memory
  - Memory accesses constitute a contiguous range of addresses.



Example of uncoalesced access in the multiple-nodes driven approach

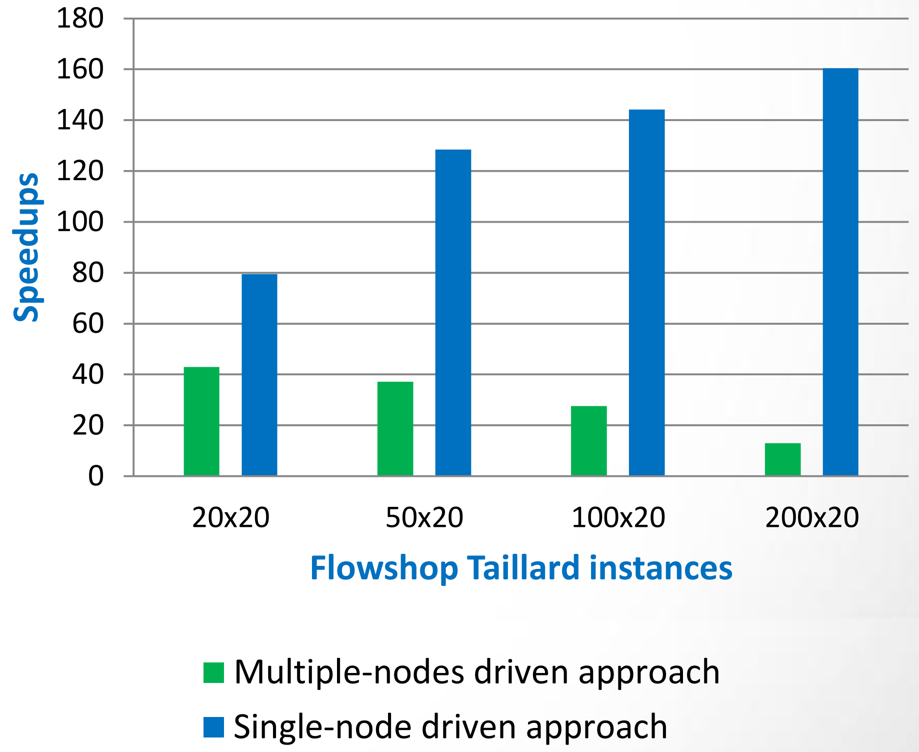


Example of a contiguous and coalesced access in the single-node driven approach

Speedups obtained with different GPU-based approaches

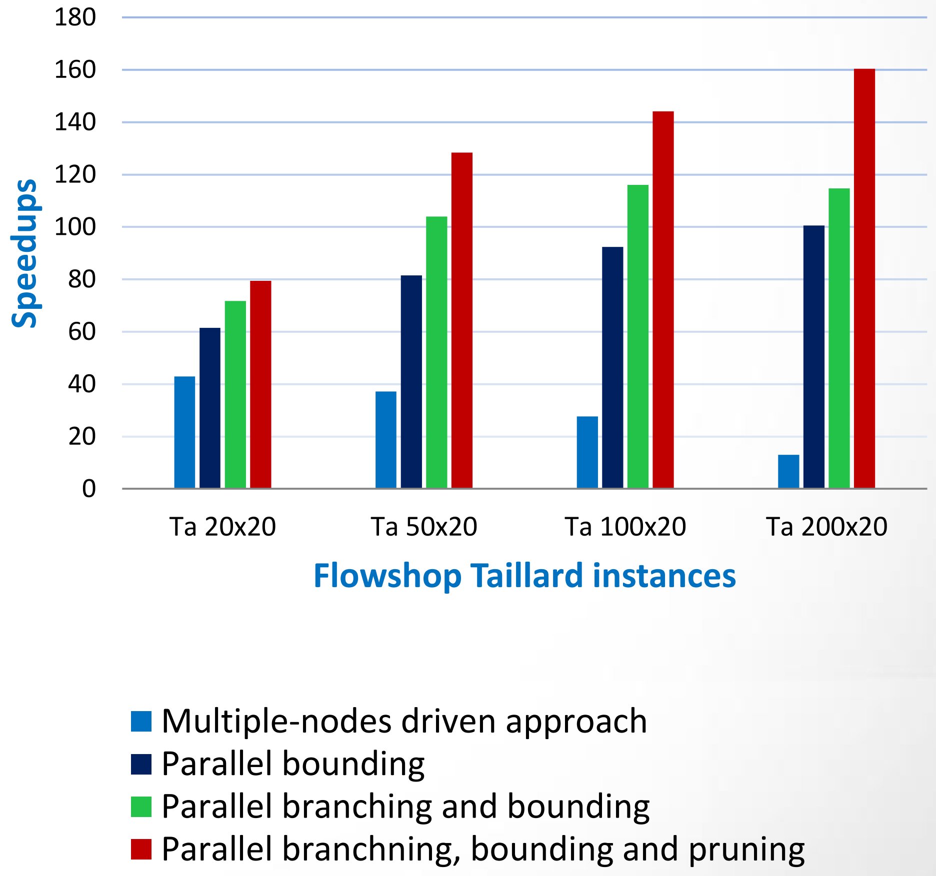
- **GPU-based parallel tree exploration** using the single node-driven approach …
  - … allows **further speedups (up to 160.41)** than the GPU-accelerated B&B based on parallel bounding **(up to 100.48)**.

- The single node-driven approach is **more efficient** than the multiple-nodes driven approach, especially for large instances.



Speedups obtained with different GPU-based approaches (Cont.)

- **Only the bounding operator** is on GPU: GB&B
- **Branching and bounding on GPU**
  - From 14% to 20% of improvements compared to GB&B
- **Bounding, branching and pruning on GPU**
  - Further enhancement from 10% to 29%



Comparison for the different approaches in terms of data transfer

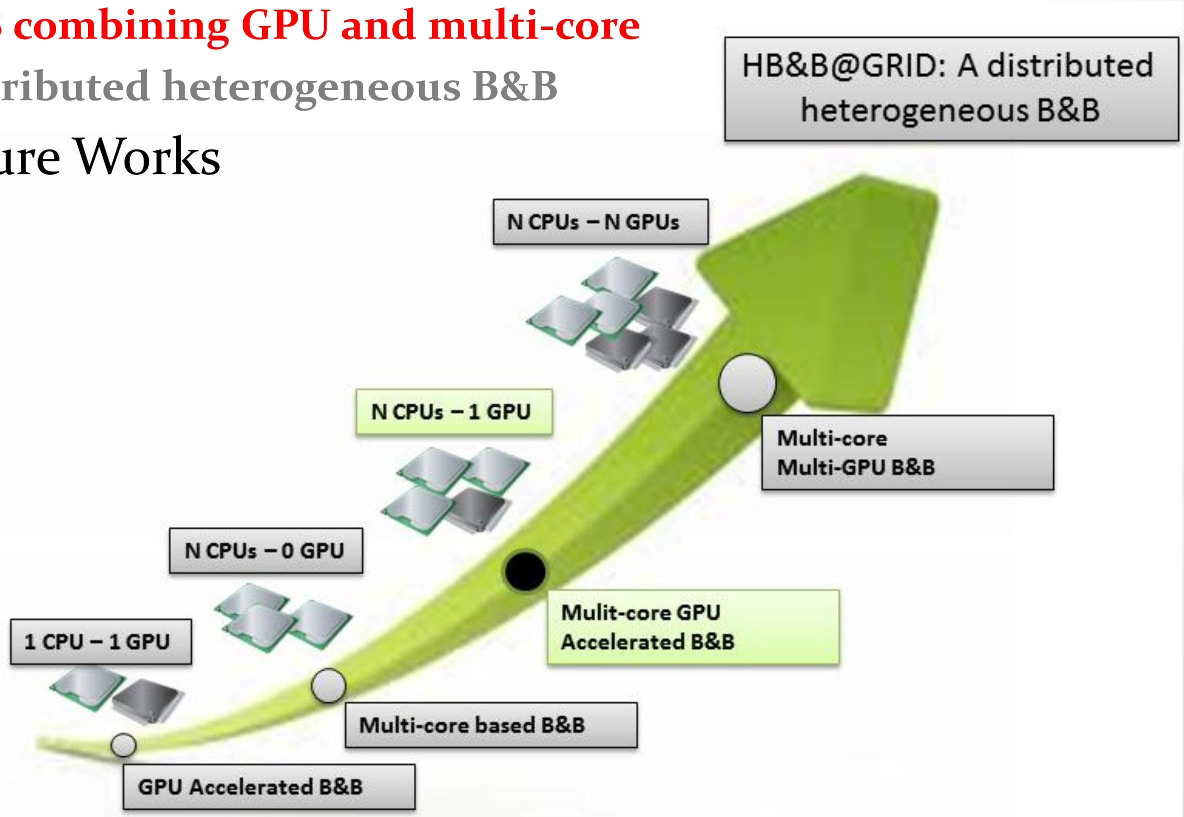
(Nb. jobs × Nb. machines)	Bounding Only	Branching and Bounding	Branching, Bounding and Pruning
200×20	181.29 MB	180.91 MB	98.42 MB
100×20	101.39 MB	100.45 MB	65.45 MB
50×20	1865.90 KB	1860.96 KB	840.10 KB
20×20	916.72 KB	926.26 KB	384.18 KB

- Performing branching, bounding and pruning operators on GPU **reduce by 50% the average amount of exchanged data**
- **Low Latency GPU-accelerated B&B (LL-GB&B)**
  - hides the latency induced by data transfers

→ Further transfer latency minimization by judiciously using multiple CPU cores available on nowadays resources

Outline

- Context and objectives
- **Contributions**
  - **GPU-accelerated parallel B&B - Application to FSP**
  - **Heterogeneous B&B combining GPU and multi-core**
  - **HB&B@GRID: a distributed heterogeneous B&B**
- Conclusions and Future Works



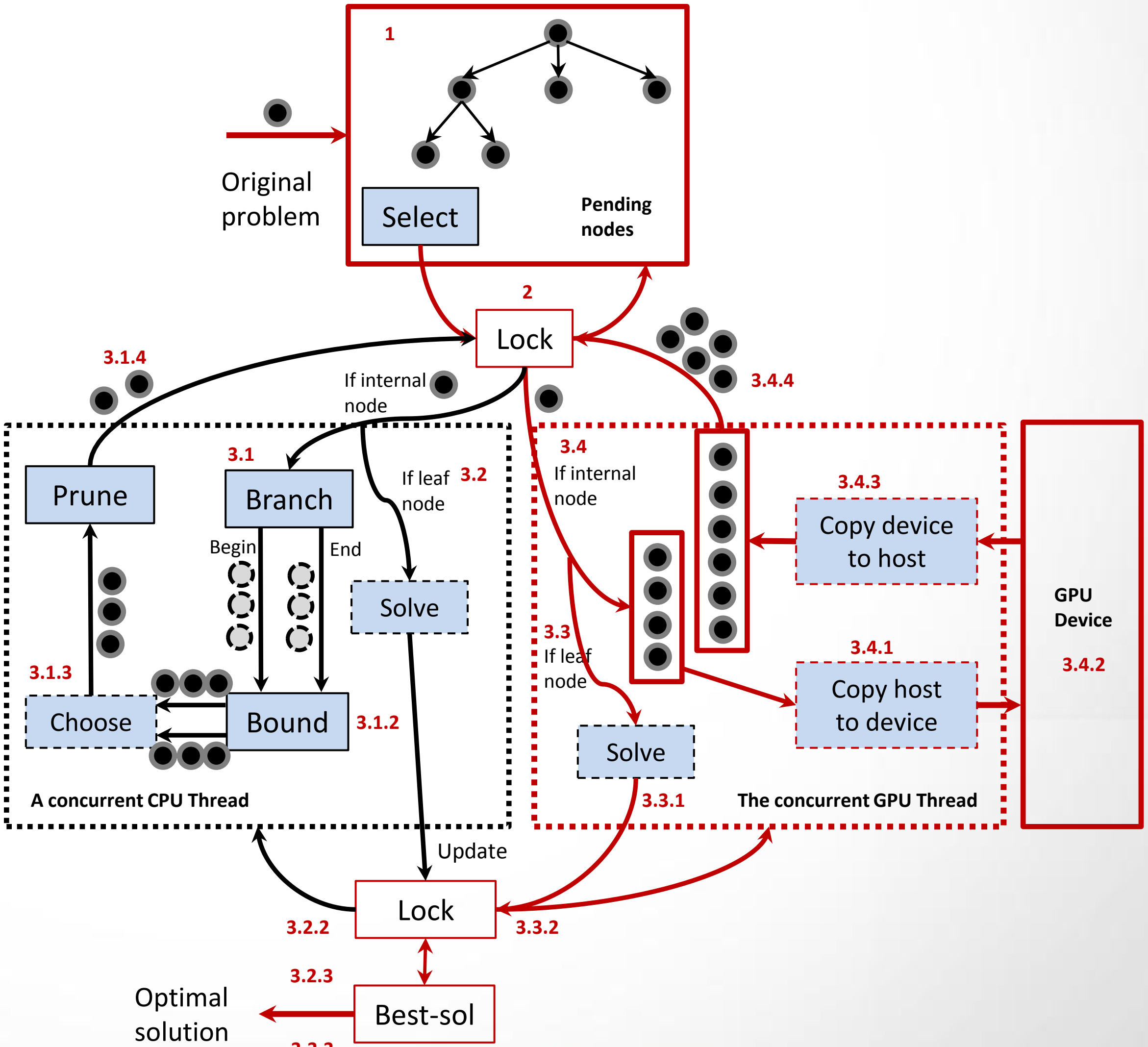
ConcuRrent multi-core Low-Latency GPU-accelerated B&B (*RLL-GB&B*)

- Challenges related to **computation and data partitioning**
- Concurrent GPU thread + Concurrent CPU thread
- **Concurrent GPU thread**
  - The idea: thread **with highest priority** exploits the computing power of the GPU
- Access to shared variable is handled using locks

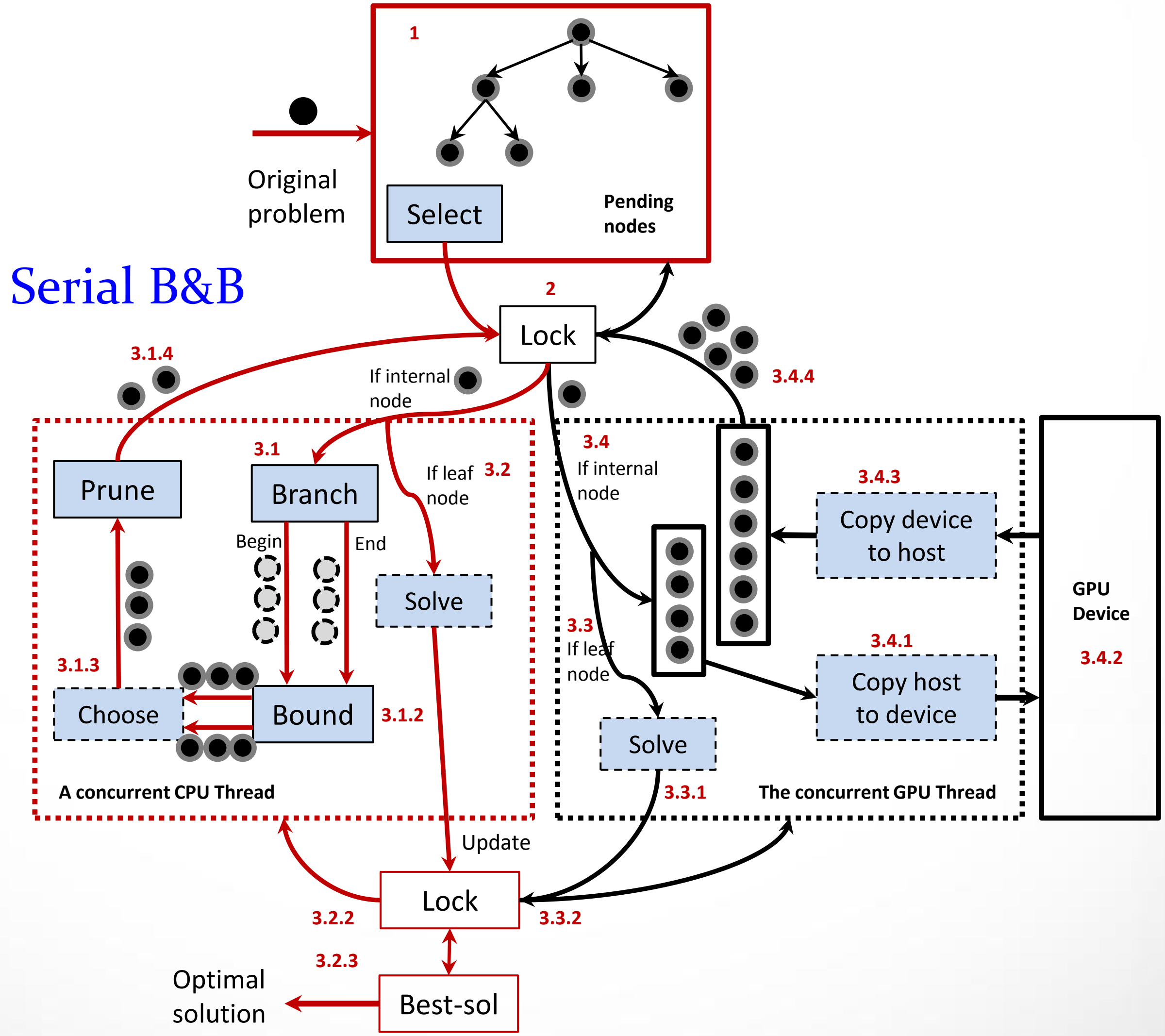
I. Chakroun, N. Melab, M. Mezmaz and D. Tuyttens. Combining multi-core and GPU computing for solving combinatorial optimization problems. **Journal of Parallel and Distributed Computing (JPDC)** – Elsevier.

RLL-GB&B approach: Concurrent GPU thread

- A tree node is a leaf ...
  - the best solution is improved ➔ **updated**
  - the subproblem **is deleted**
- Is an internal node ...
  - inserted in the pool to be off-loaded to the GPU
- Once the best pool size is reached (using ASH) ...
  - the **LL-GB&B** is executed

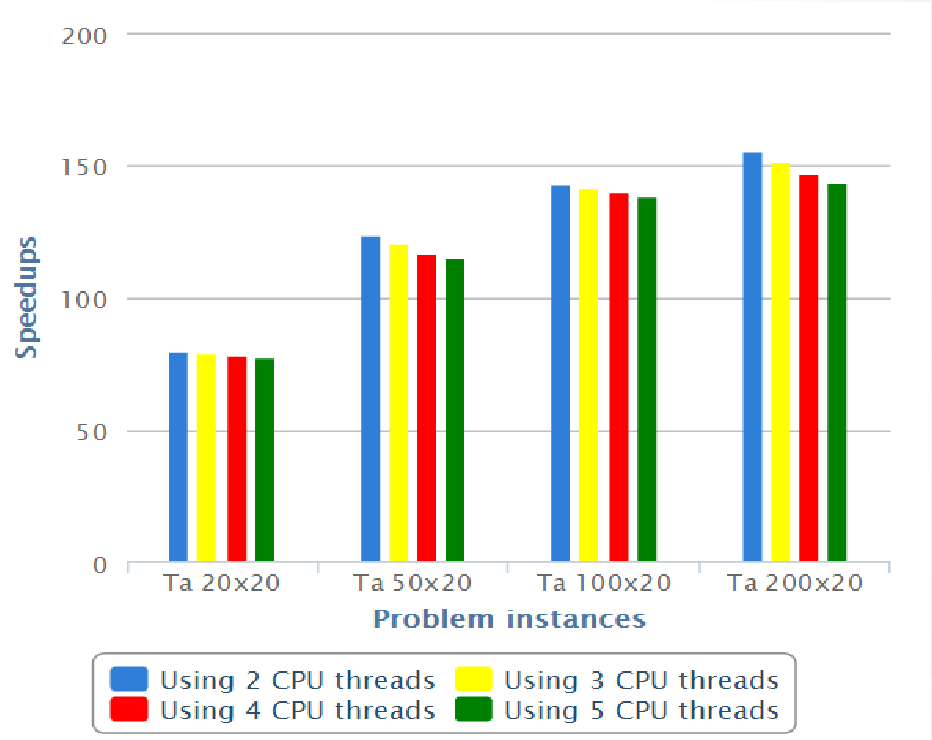


RLL-GB&B approach: Concurrent CPU thread



Performances of the RLL-GB&B approach

- The **more** the number of cores is the **worst** the speedup is compared to LL-GB&B
- Average normalized waiting times
  - … the GPU thread is forced to wait for the lock if it has the highest priority
  - … waiting time increases according to the number of concurrent CPU threads



- Under-utilization of the GPU by the concurrent GPU thread

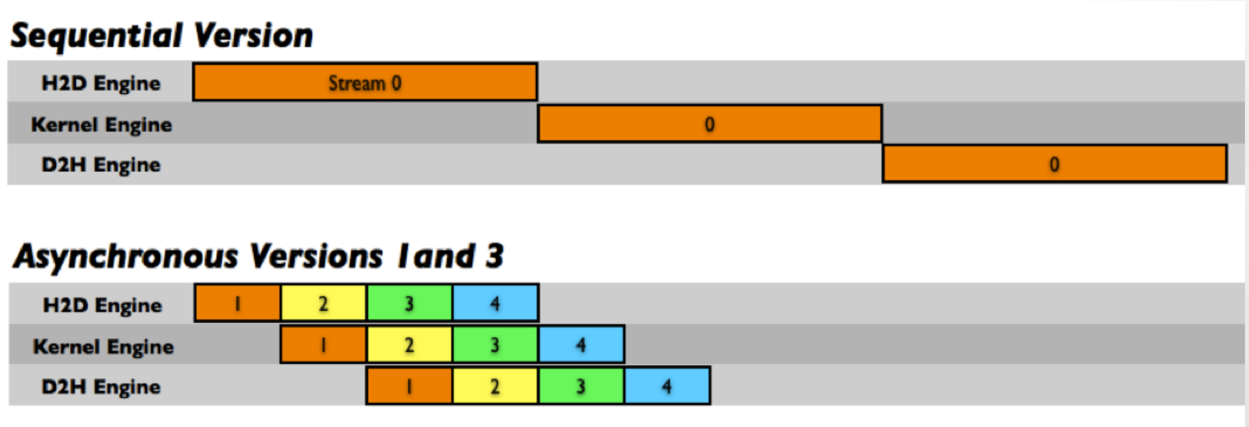
➔ Cooperative approach

Idea: CPU threads prepare data for GPU using Cuda streaming + GPU (only) explores the tree

CooPerative multi-core Low Latency GPU-accelerated B&B (*PLL-GB&B*)

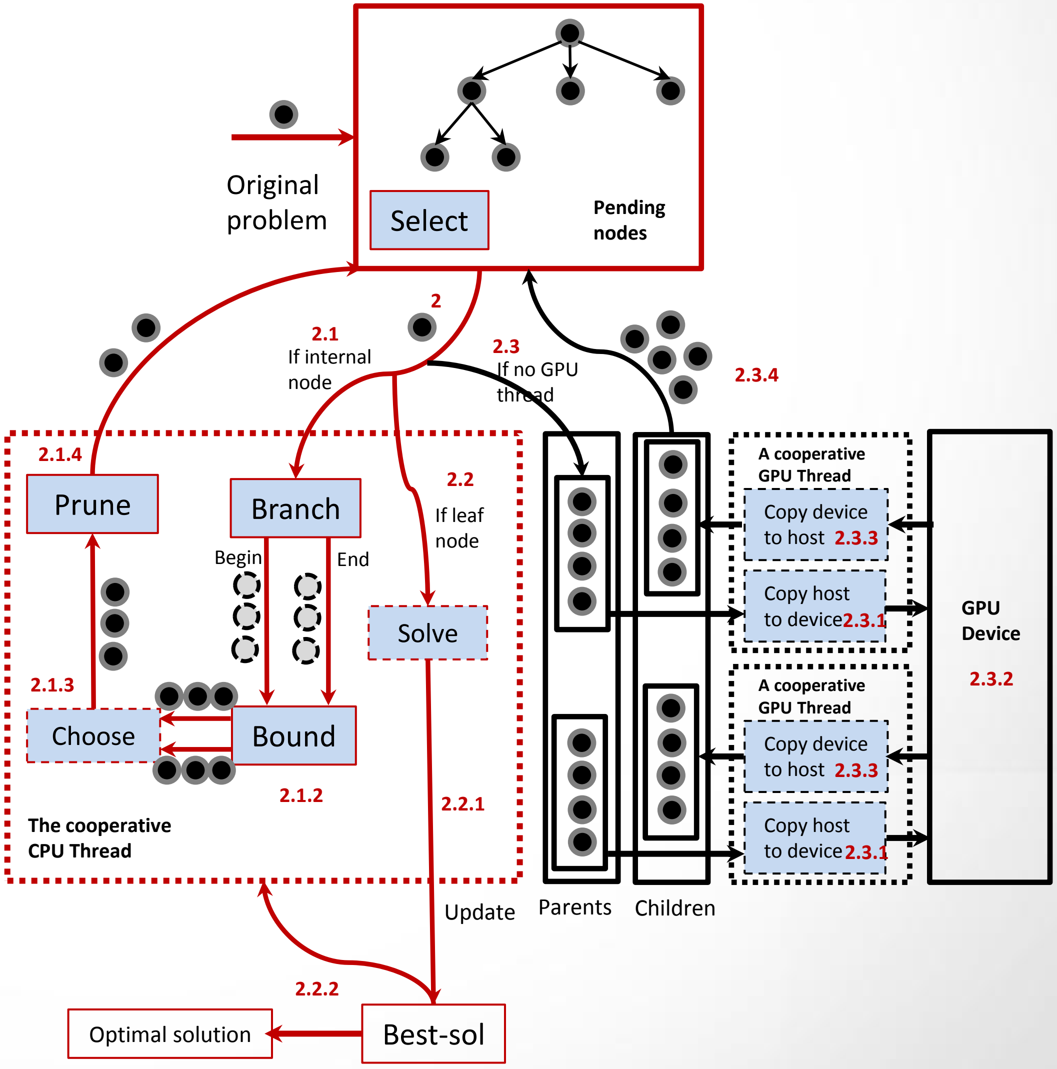
- **Avoids synchronization issues** and further **minimizes the CPU-to-GPU data transfer latency**
- Hides this latency by **executing transfers asynchronously with kernel calls**

Overlapping and interleaving data transfers and kernel calls



PLL-GB&B: Cooperative CPU thread

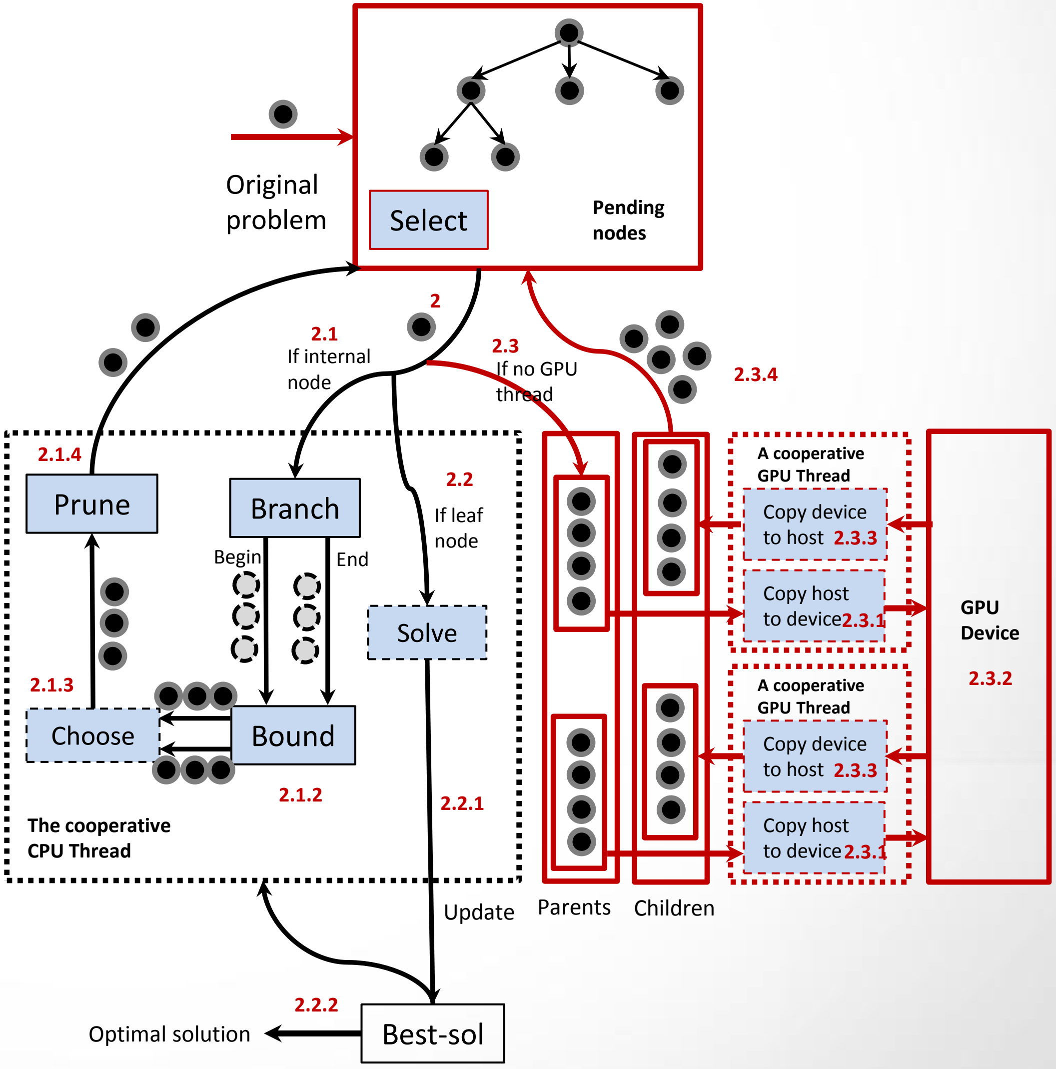
- **Selects the pool of nodes** to be off-loaded to the GPU.
- Creates the **collaborative GPU threads.**
- **Explores** some pending subproblems while the collaborative GPU threads **are busy.**
- **Inserts** subproblems returned by collaborative GPU threads.



PLL-GB&B: Cooperative GPU thread

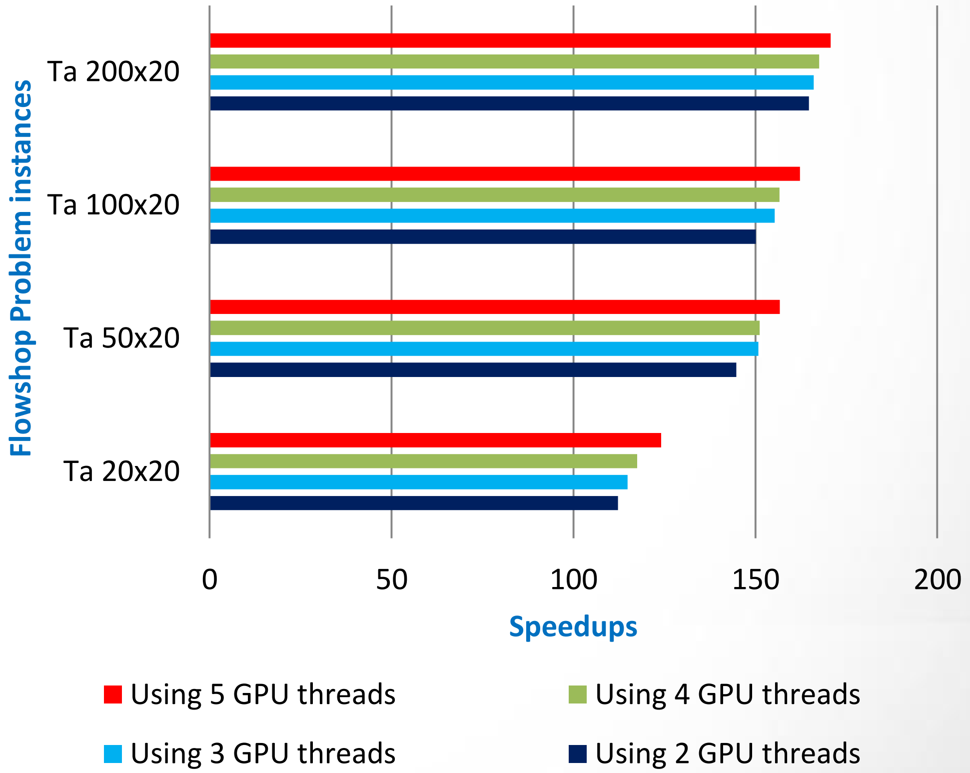
- **A CUDA-stream of ordered operations** associated with collaborative GPU threads

- Each cooperative GPU thread **handles a part of the pool** of selected nodes, by performing …
  - asynchronous transfers to the GPU device
  - calls to branching, bounding and pruning kernels
  - copies of results back to the CPU host



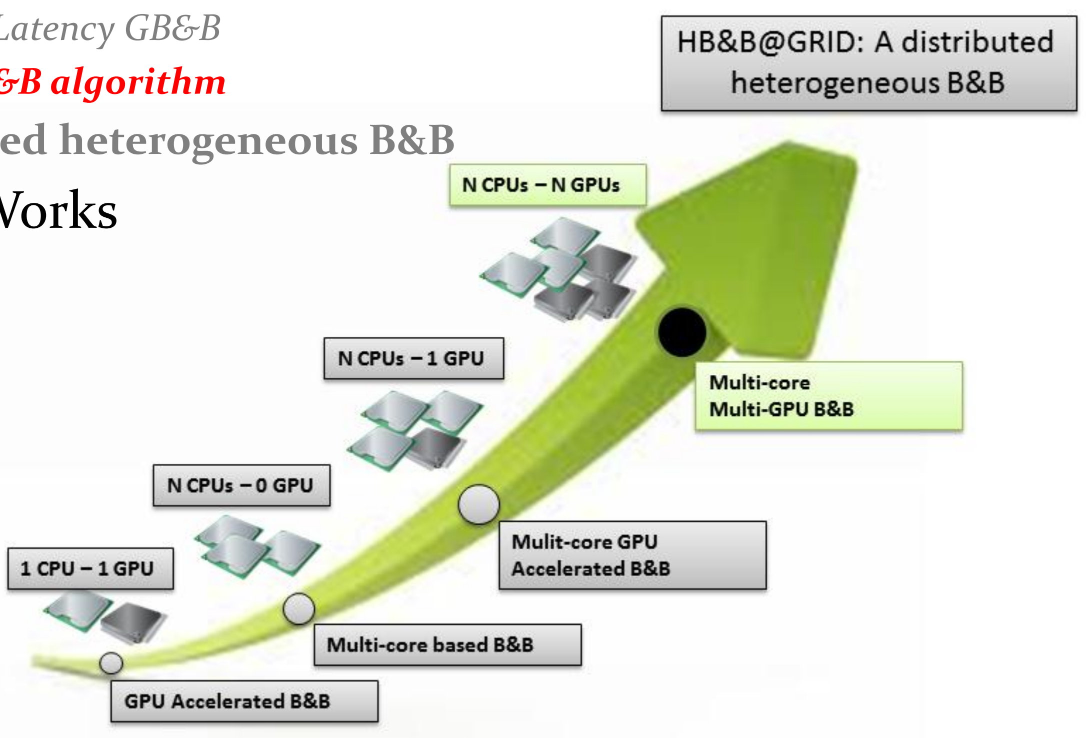
Performances of the PLL-GB&B approach

- Speedups increase according to the instance size and to the number of cooperative GPU threads
- Acceleration up to **170** compared to a serial B&B
- Enhancement up to 36% compared to the LL-GB&B approach



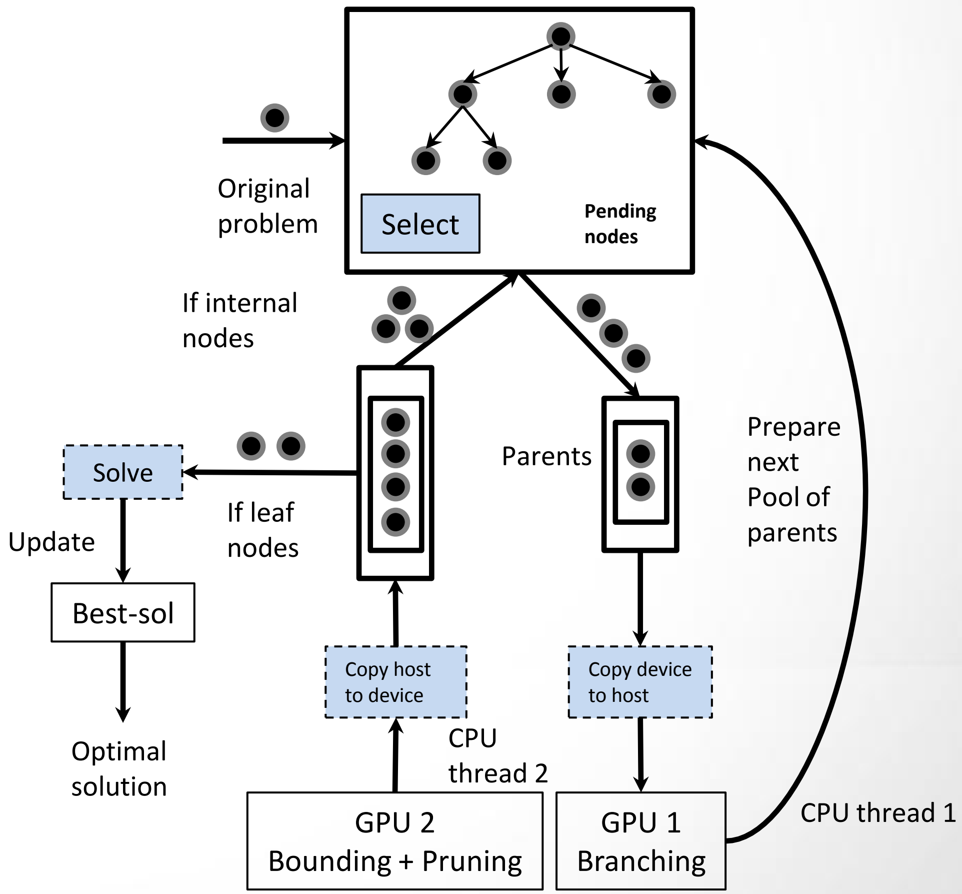
Outline

- Context and objectives
- **Contributions**
  - **GPU-accelerated parallel B&B - Application to FSP**
  - **Heterogeneous B&B combining GPU and multi-core**
    - *ConcuRrent multi-core Low-Latency GB&B*
    - *CooPerative multi-core Low Latency GB&B*
    - ***Low Latency Multi-GPU B&B algorithm***
  - **HB&B@GRID: a distributed heterogeneous B&B**
- Conclusions and Future Works

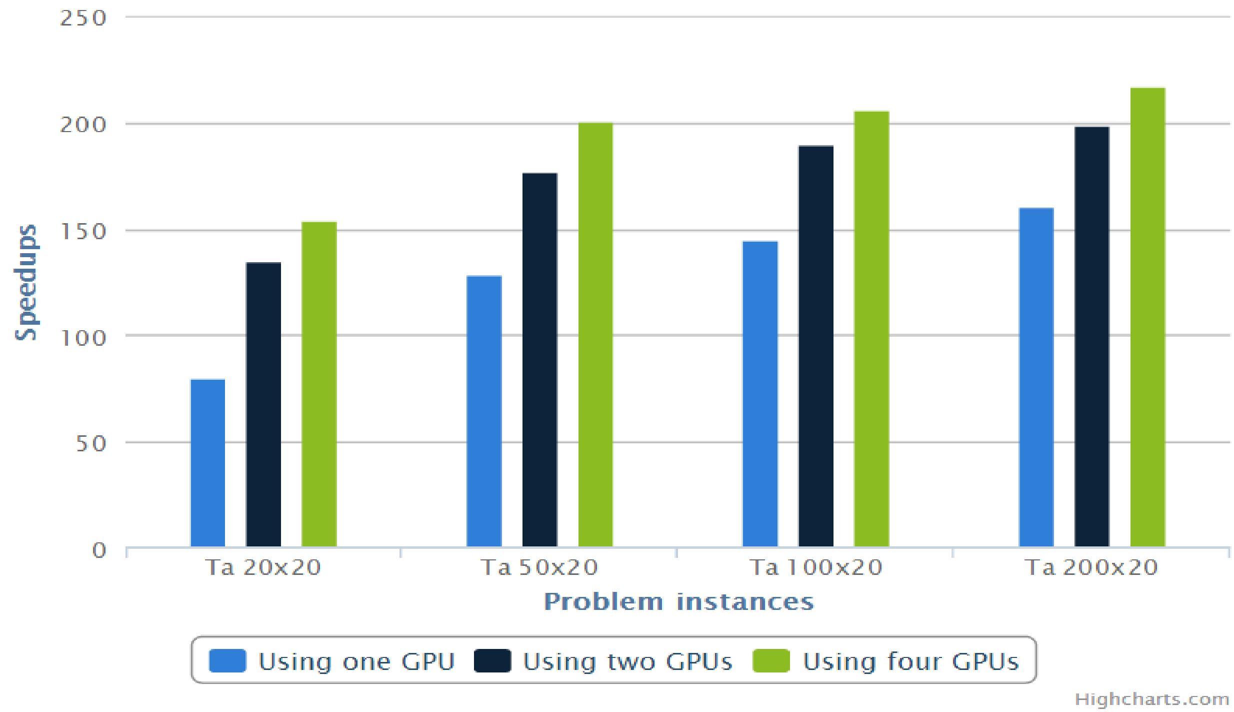


Low Latency Multi-GPU B&B algorithm (*LL-MultiGB&B*)

- The branching kernel is launched by CPU thread 1 and executed on GPU 1
  - The resulting pool is moved to the memory of GPU 2 using the **peer-to-peer access mechanism**
- The first CPU thread **prepares the pool of the next nodes to be explored**
- CPU thread 2 launches the bounding and pruning kernels on GPU 2
  - Applied on the result of branching (sent by GPU 1)



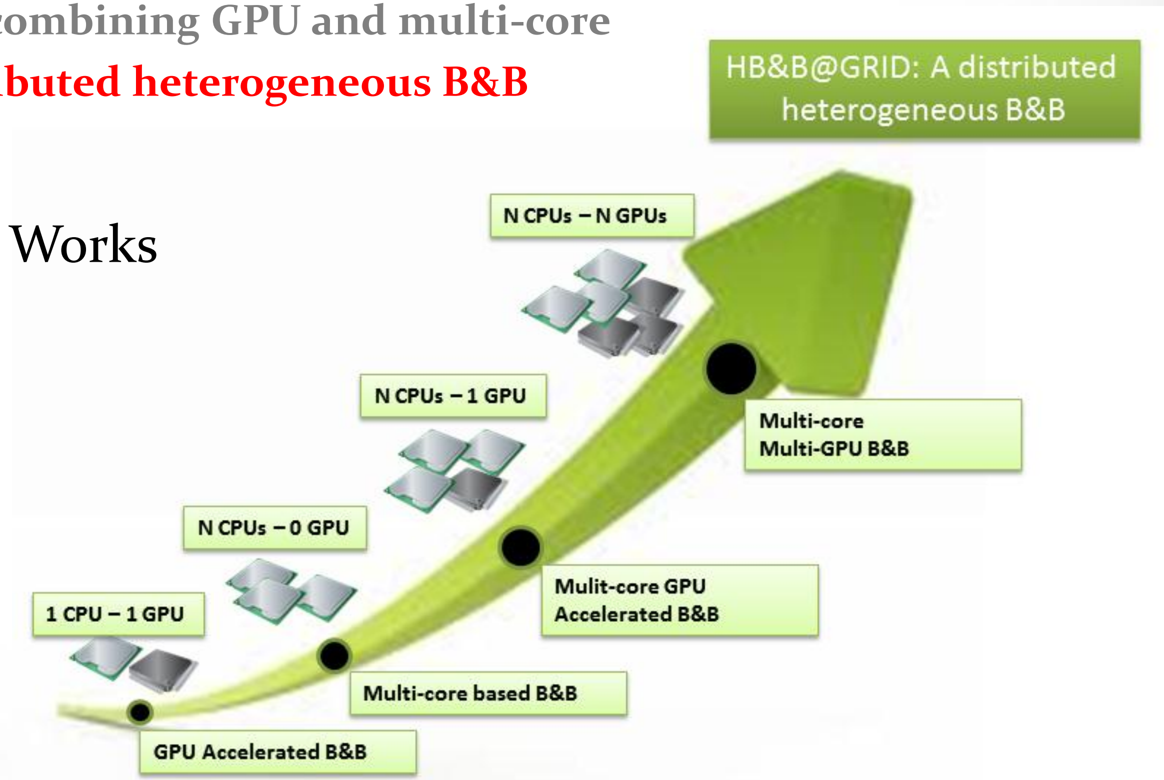
Speedups using single/multiple GPUs



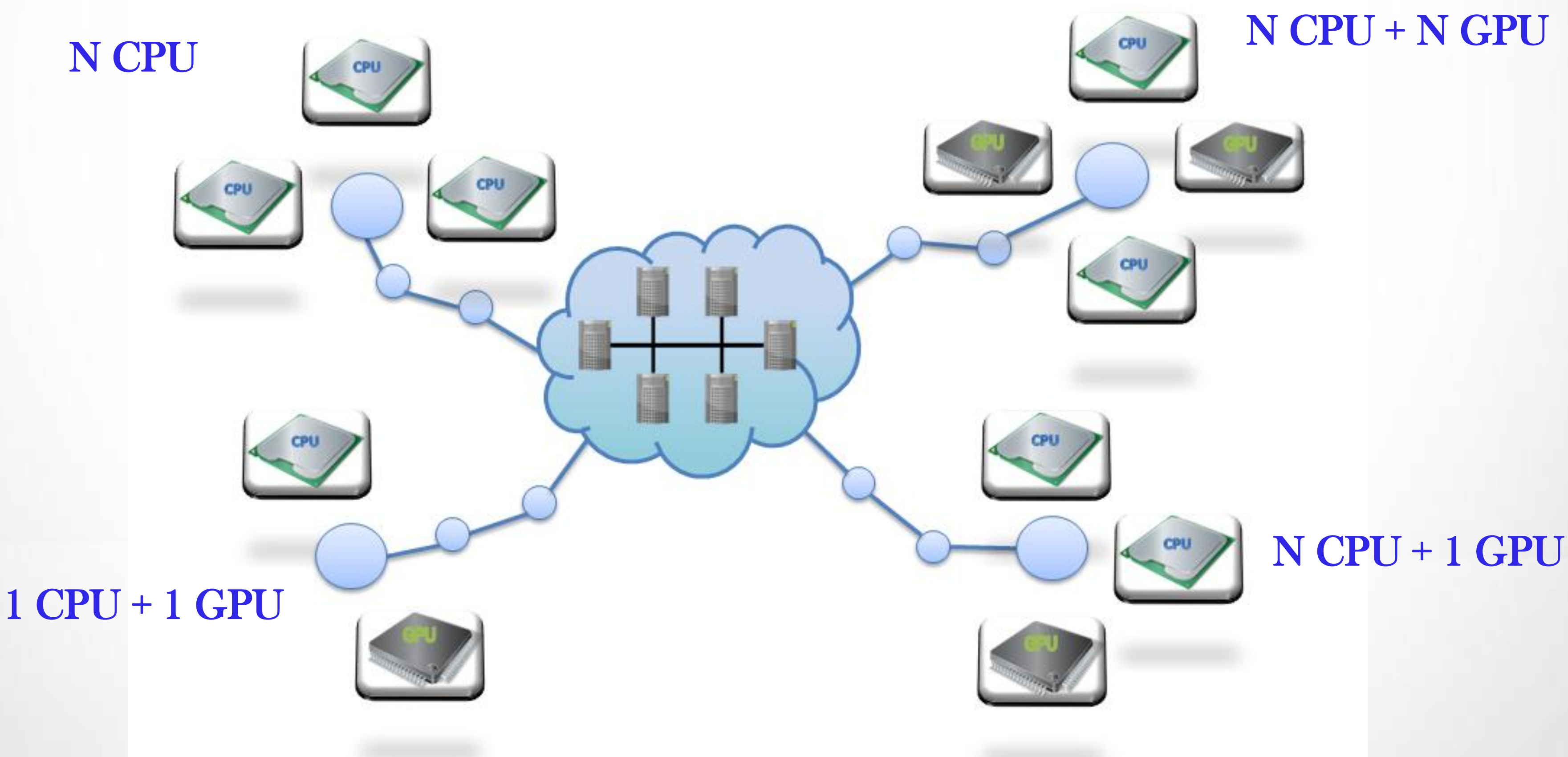
- Speedup up to **217** with 4 GPUs for the (200 x 20) problem instances

Outline

- Context and objectives
- **Contributions**
  - **GPU-accelerated parallel B&B - Application to FSP**
  - **Heterogeneous B&B combining GPU and multi-core**
  - **HB&B@GRID: a distributed heterogeneous B&B**
    - ***The B&B meta-algorithm***
    - ***The B&B@Grid approach***
- Conclusion and Future Works



A large-scale adaptive heterogeneous multi-core GPU-accelerated B&B algorithm



Overall design of the adaptive heterogeneous B&B (*HB&B@GRID*)

- Combining **two hierarchical levels of parallelism**
  - B&B@GRID master-workers approach [*Mezmaz et al.*, 2007]
    - … splits the B&B tree among multiple nodes
  - B&B meta-algorithm
    - … dynamically selects the parallel B&B to be deployed according to the underlying configuration

The B&B meta-algorithm

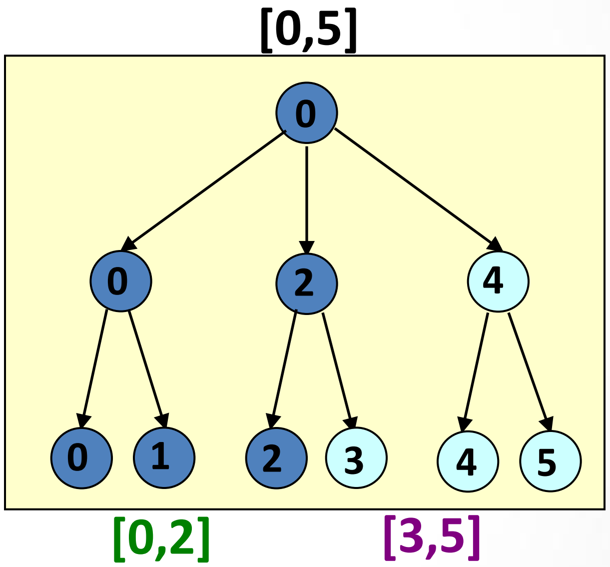
- The meta-algorithm **detects the number of supplied CPU cores and GPU devices**
  - **LL-GB&B:** single CPU + single GPU
  - **MC-B&B:** multi-core CPU without GPUs
  - **PLL-GB&B:** multi-core CPU + single GPU
  - **LL-multiGB&B:** multi-core CPU + multiple GPUs

Algorithm 15 Template of the proposed meta-algorithm.

```
max_nb_devices = Detect_GPU_Characteristics();
max_nb_cores = Get_CPU_Characteristics();

if max_nb_devices = 0 then
  | Run_MCB&B_algorithm();
end
else if max_nb_devices = 1 && max_nb_cores < 2 then
  | Run_LLGB&B_algorithm();
end
else if max_nb_devices = 1 && max_nb_cores >= 2 then
  | Run_PLLGB&B_algorithm();
end
else if max_nb_devices > 1 then
  | Run_LL-MultiGB&B_algorithm();
end
```

The B&B@Grid approach

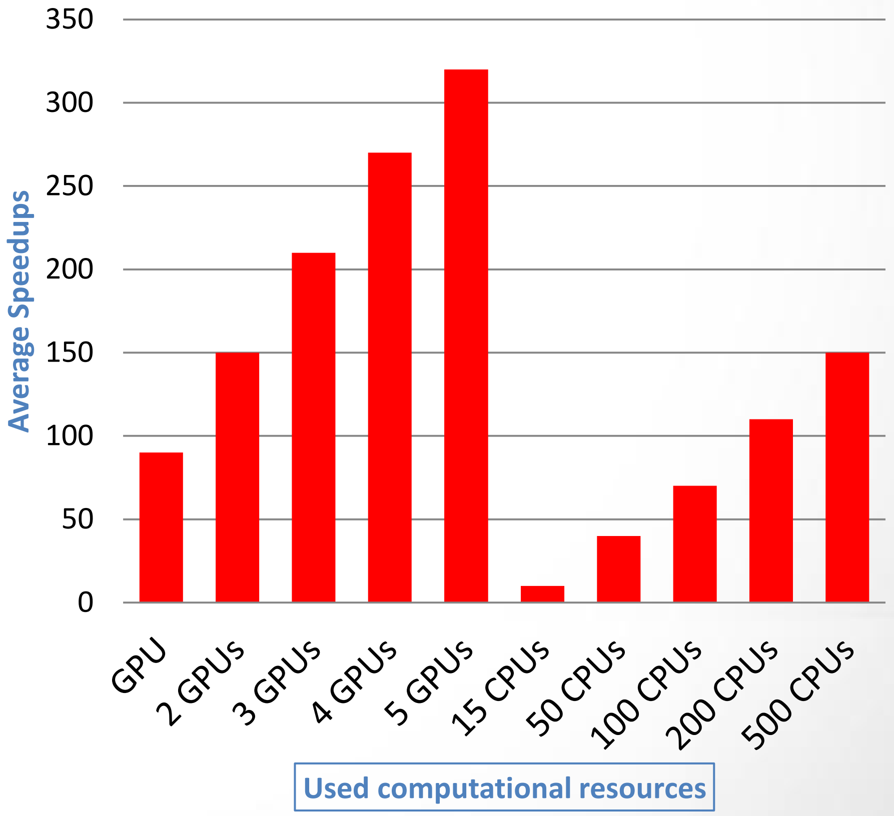


- The approach uses a special description ...
  - Each node is assigned **a number**
  - Work unit (collection of nodes) = **an interval**

- The approach is **Dispatcher-Worker** based on the **work stealing** paradigm
  - Dispatcher: maintains a pool of work units (intervals) and the global solution found so far
  - Worker: performs B&B on a given interval and updates the global solution

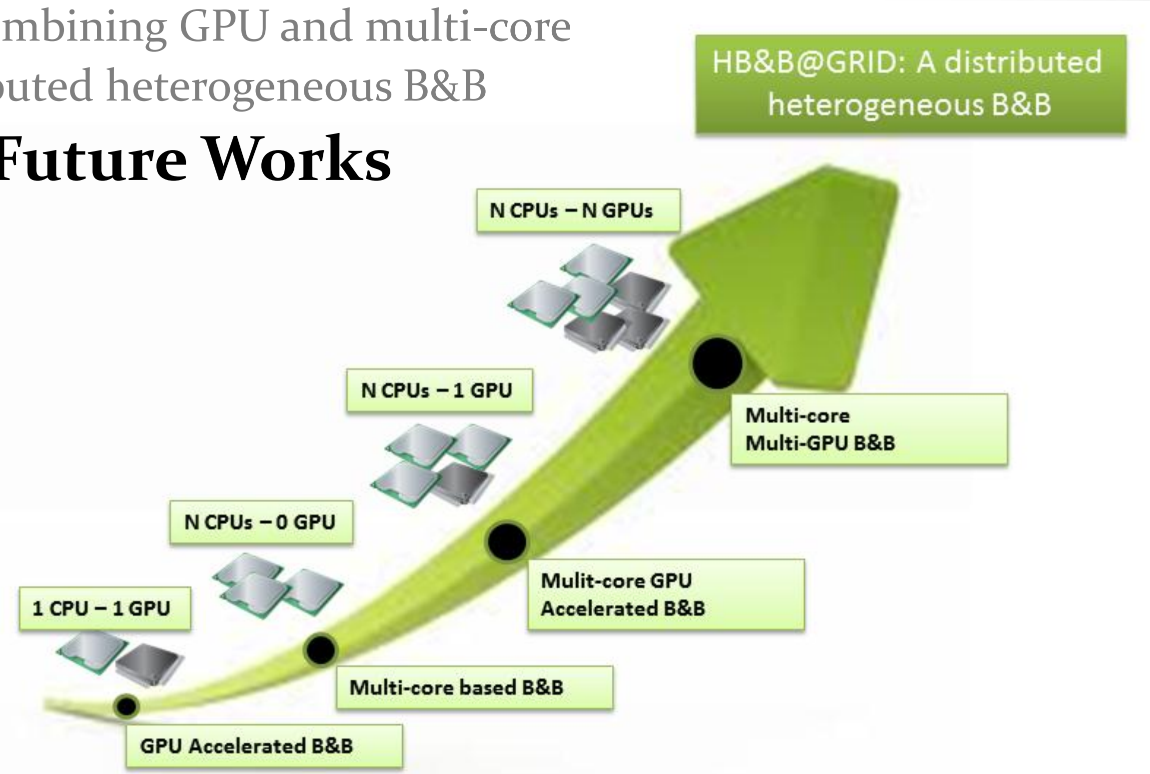
The HB&B@GRID approach: experimental results

- For a same computational power, the GPU-based B&B **is more efficient** than a distributed CPU-based B&B
- Using **3 distant GPUs** is **5 times faster** than **50 CPUs**
- Using **5 GPU devices** allows accelerations **twice higher** than those obtained using **500 CPU cores**



Outline

- Context and objectives
- Contributions
  - GPU-accelerated parallel B&B - Application to FSP
  - Heterogeneous B&B combining GPU and multi-core
  - HB&B@GRID: a distributed heterogeneous B&B
- **Conclusions and Future Works**



Conclusions and general insights

- GPU-accelerated parallel B&B
  - Parallel bounding on GPU using branch refactoring and shared memory allows high speedups (**up to ~100** in our work) …
    - The computation applied on tree nodes (e.g. bounding function) should be **shared-data intensive** and contain conditional instructions with **long branches**
  - … is more efficient if combined with GPU-based tree exploration (**accelerations up to ~160**)
    - The overhead induced by CPU-GPU data transfer is minimized by …
    - **Auto-adapting the size of the pool** off-loaded to the GPU and FSP instance to be solved
    - Limiting the **granularity** of each thread to a **single tree node**

Conclusions and general insights (Cont.)

- Heterogeneous B&B combining multi-GPU and multi-core
  - **Higher speedups (up to 217** in this work) could be obtained …
  - using the **cooperative low latency data/work partitioning** …
  - … based on the CUDA **data streaming** to further reduce the cost of CPU-GPU data transfer
  - … and the **P2P memory access** between GPUs
- Large scale B&B (multi-GPU+multi-core+grid computing)
  - Heterogeneous B&B meta-algorithm + B&B@Grid
  - **Auto-adaptive** to the target execution hardware configuration
  - Solving very large problem instances
    - **Proof of concept** on (20 x 20) FSP instances (**4,5h - 108h**)

Future work

- Extending HB&B@Grid with **GPU-level check-pointing** …
  - … for solving (50 x 20) FSP instances (years of computation!)
- Considering other …
  - bounding functions (library): **adaptive selection of bounding operator**
  - exact tree-based methods (**B&X**), problems
  - **Ph.D thesis of Rudi Leroy (Maison de la Simulation)**
- Investigating other perspectives …
  - in-house to **energy-aware cloud**-based HB&B
  - adapting to multi and many-core evolution with **more advanced features** (e.g. Kepler GPU - Nvidia GPUDirect, MIC, …)

International Publications

International journals (3 accepted + 1 submitted)

- I. Chakroun, N. Melab, M. Mezmaz and D. Tuyttens. *Combining multi-core and GPU computing for solving combinatorial optimization problems.* **Journal of Parallel and Distributed Computing (JPDC)** - Elsevier.
- I. Chakroun, M.Mezmaz, N. Melab, and A.Bendjoudi. *Reducing thread divergence in a GPU-accelerated branch-and-bound algorithm*. **Concurrency and Computation: Practice and Experience** vol 25, N° 8, pages 1121-1136, 2013 - John Wiley & Sons.
- N. Melab, I. Chakroun, and A. Bendjoudi. *GPU-accelerated Bounding for Branch-and-Bound applied to a Permutation Problem using Data Access Optimization.* **Concurrency and Computation: Practice and Experience** John Wiley & Sons.
- I. Chakroun and N. Melab. *Towards an heterogeneous and adaptive parallel Branch-and-Bound algorithm.* **Journal of Computer and System Sciences** - Elsevier (Submitted).

International conferences (4)

- Elsevier Intl. Conf. e on Computational Science (**Elsevier ICCS'13**)
- 14th IEEE Intl. Conf. on Cluster Computing (**IEEE CLUSTER'12**)
- 14th IEEE Intl. Conf. on High Performance Computing and Communications (**IEEE HPCC'12**)
- 9th Intl. Conf. on Parallel Processing and Applied Mathematics (**LNCS, PPAM'11**)

Book Chapters (1)

- I. Chakroun and N. Melab. *GPU-accelerated Tree-based Exact Optimization Methods Designing scientific applications on GPUs.* **CRC Press, Taylor & Francis G**roup.