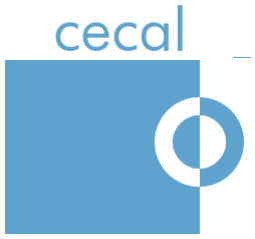
PARALLEL IMPLEMENTATIONS OF THE MINMIN HETEROGENEOUS COMPUTING SCHEDULER IN GPU

MAURO CANABÉ **SERGIO NESMACHNOW**

Centro de Cálculo, Facultad de Ingeniería
Universidad de la República, Uruguay

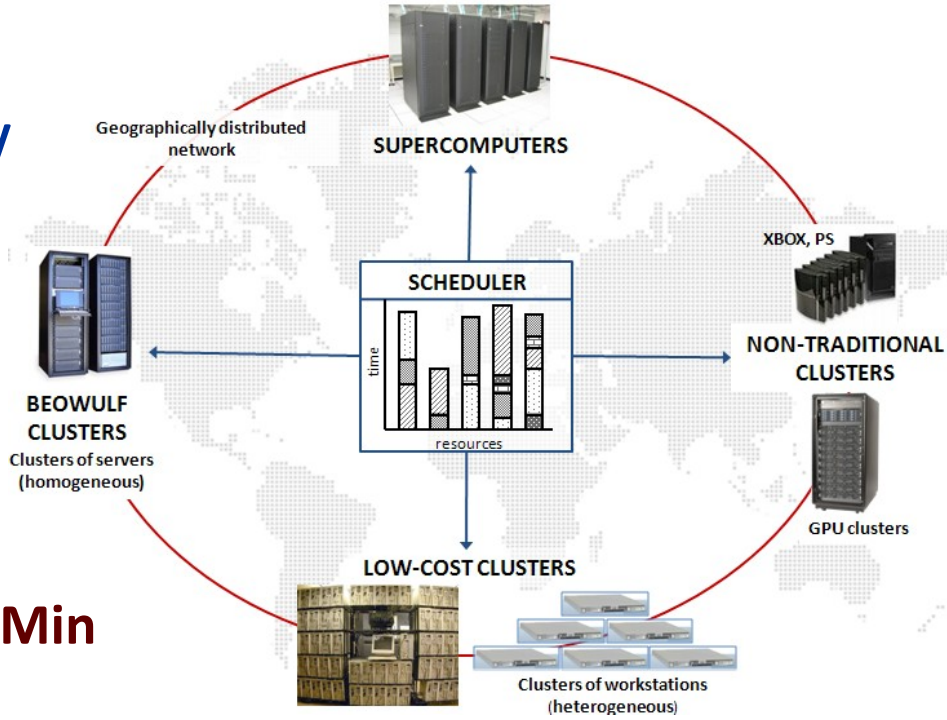


CONTENT

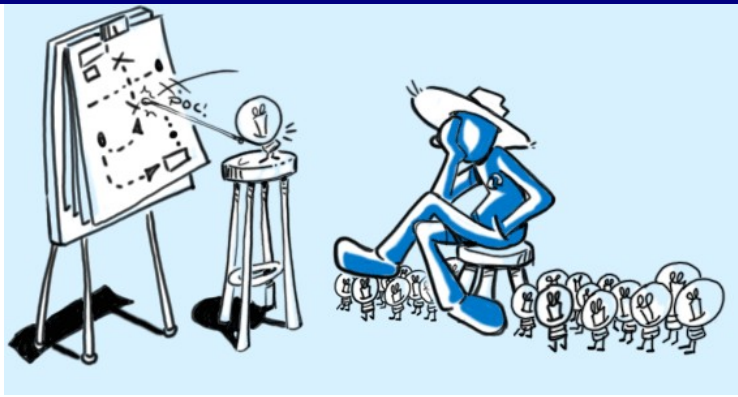
1. Introduction
2. Scheduling in heterogeneous environments
3. MinMin scheduling heuristic
4. GPU Computing
5. MinMin implementation on GPU
6. Experimental analysis
7. Conclusions and Future Work

INTRODUCTION

- Distributed heterogeneous computing (HC)
  - Parallel computing: clusters, grid computing.
- Scheduling
  - Assigning tasks to resources.
  - Several criteria: execution time, utilization, others.
  - NP-hard problem.
- Heterogeneity increases the complexity of the scheduling problem
  - Heterogeneous Computing Scheduling Problem (HCSP).
  - Heuristics to solve it.
- **This work presents parallel implementations in GPU of the MinMin scheduling heuristic for HC.**



SCHEDULING IN HC ENVIRONMENTS



- HC: coordinated set of heterogeneous computing resources.
- Scheduling is crucial to achieve efficiency.

- HCSP formulation:
  - Set of heterogeneous machines $P = \{m1, m2, ..., mM\}$.
  - Set of tasks $T = \{t1, t2, ..., tN\}$ to be executed on P.
  - Execution time function $ET: P \times T \rightarrow R$.
  - Time required to execute task t_i in machine m_j.
  - Goal: find a schedule (function $f: T^N \rightarrow S^M$) which minimizes makespan:

$$makespan = \max_{m_j \in P} \sum_{\substack{t_i \in T: \\ f(t_i)=m_j}} ET(t_i, m_j)$$

- HCSP is NP-hard, dimension of the HCSP, search space is $O(M^N)$

MINMIN SCHEDULING HEURISTIC

- Greedily picks the task that can be completed the soonest.

while tasks left to assign do
for each task to assign, each machine do
evaluate MCT (task, machine)
end for
assign the selected task to the selected machine
end while

Starts with a set U of all unmapped tasks

Calculates the MCT for each task in U for each machine

Assigns the task with the minimum overall MCT to the best machine

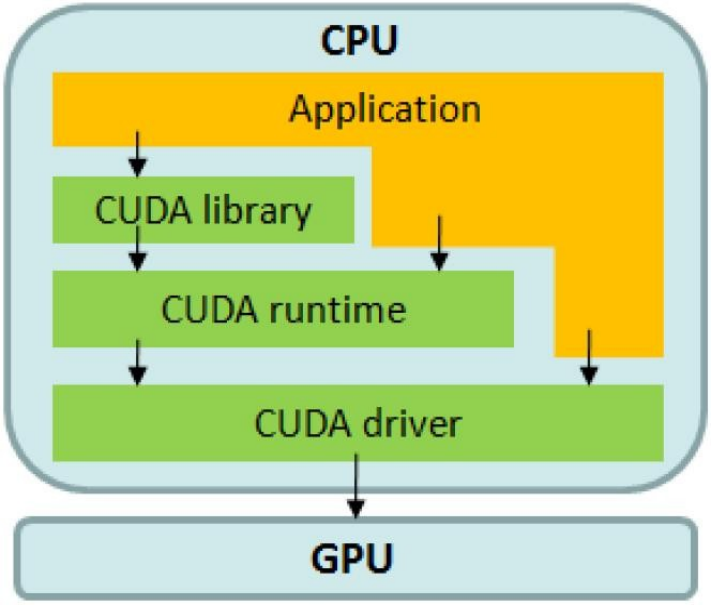
The mapped task is removed from U, and the process is repeated until all tasks are mapped

- This procedure leads to balanced schedules and also allows finding smaller makespan values than other heuristics, since more tasks are expected to be assigned to the machines that can complete them the earliest.

GPU COMPUTING

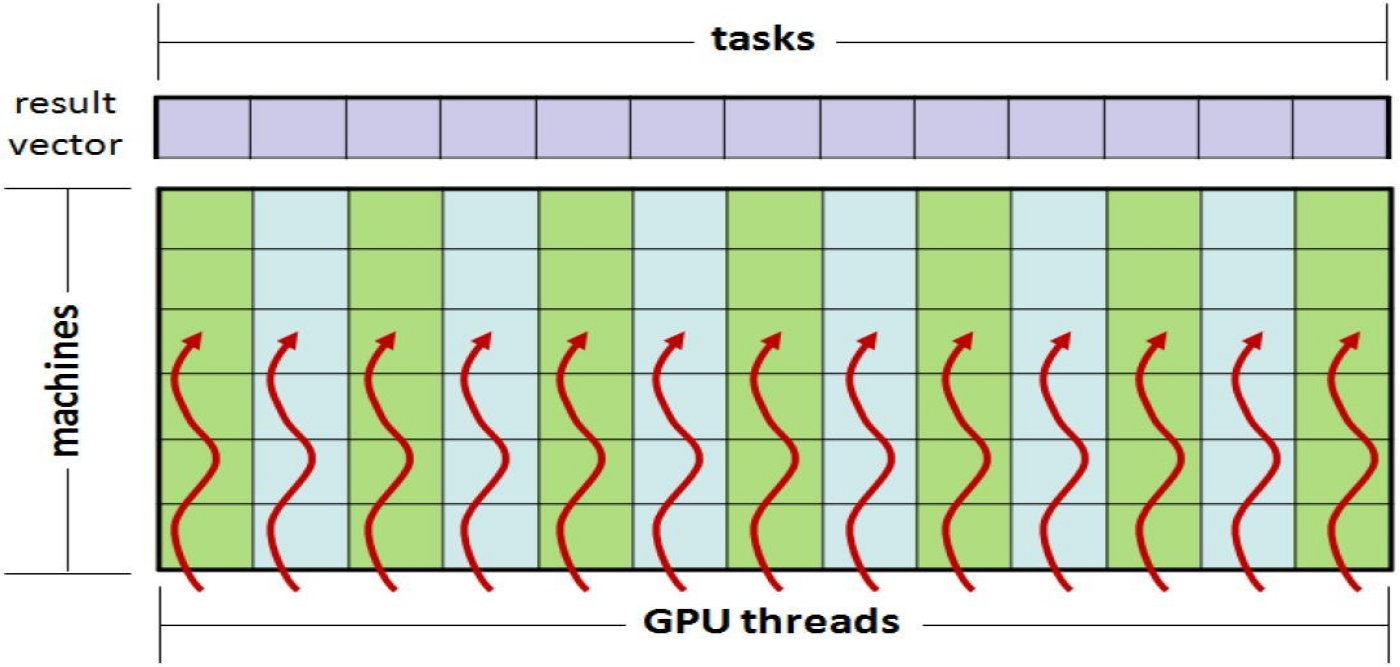
- GPUs: originally designed to exclusively perform graphic processing.
- Nowadays, GPUs have a large computing power and are used as a powerful parallel architecture to efficiently execute applications.
- Initially programmed using low-level mechanism (BIOS interruption services or graphic APIs such as OpenGL and DirectX).
- In 2007, NVIDIA introduced CUDA, a software architecture for managing the GPU as a parallel computing device without using a graphic API.

- A large number of threads run in parallel, each one executing simultaneously a copy of a procedure on a different data set.
- The GPU has its own DRAM. Data are copied from the GPU-DRAM to the RAM of the host using optimized calls to the CUDA API.



MINMIN IMPLEMENTATION ON GPU

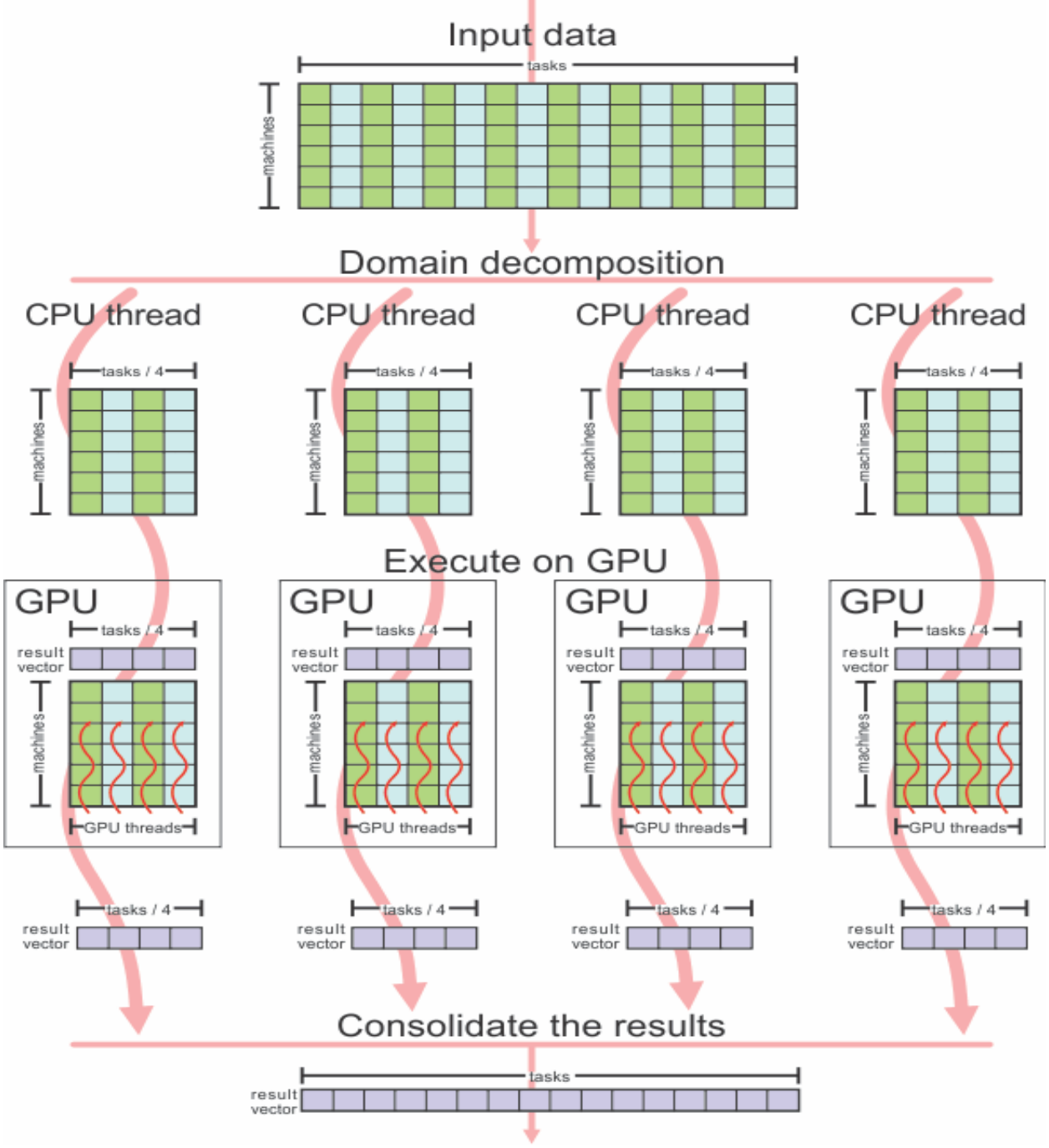
- Parallel MinMin using one GPU
  - For each unassigned task, the MCT evaluation for all machines is performed in parallel on the GPU, building a vector that stores the identifier of the task, the best MCT value found, and the machine to get that value.
  - The indicators in the vector are processed in the reduction phase to obtain the best MCT value overall, and the best pair (task, machine) is assigned.
    - This vector processing is also performed using the GPU.



MINMIN IMPLEMENTATION ON GPU

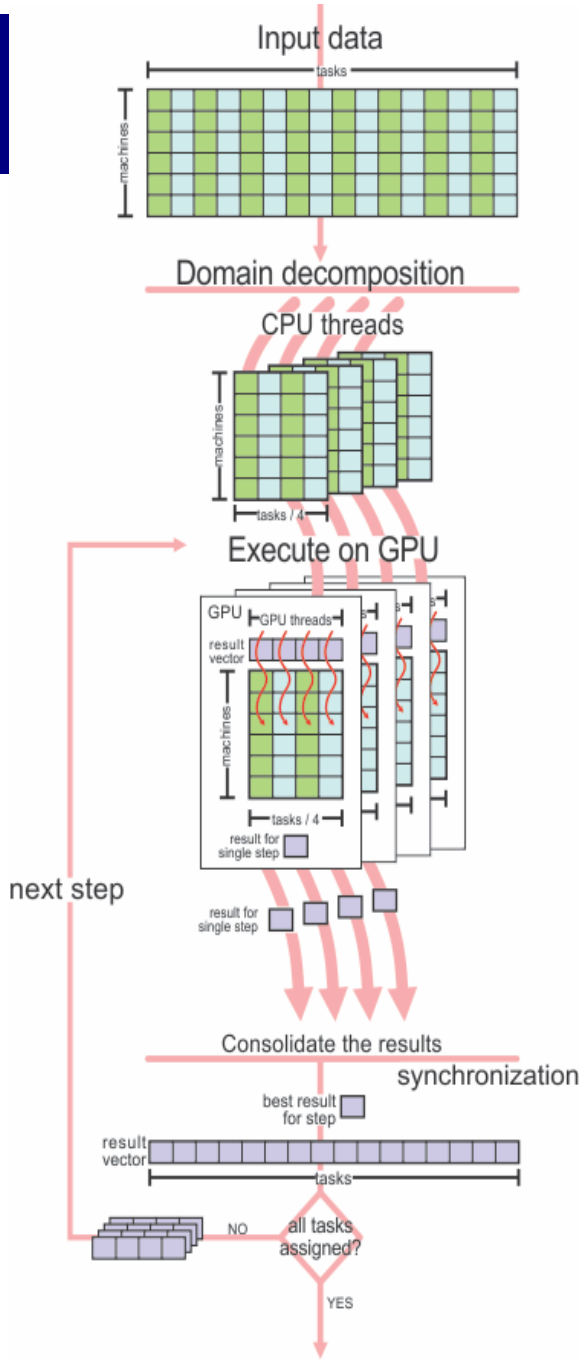
- Parallel MinMin in four GPUs with domain decomposition using pthreads
  - Master-slave multithreading approach implemented with POSIX threads (pthreads): executes the same algorithm on four GPUs independently.
  - The domain partition strategy splits the domain (i.e. the set of tasks) into N equally-sized parts, so that each task belongs to only one subset
  - Each GPU performs the MinMin algorithm on a subset of the tasks input data on all machines, and a master process consolidate the results after each GPU finishes its task.

- Parallel MinMin in four GPUs with domain decomposition using OpenMP
  - Only differs in how the threads are handled: automatically managed and synchronized using OpenMP directives included in the implementation.
  - The code for loading input data, dumping the results, performing the domain partition, and implementing the GPU kernel are identical.


Input data
tasks
machines
Domain decomposition
CPU thread
CPU thread
CPU thread
CPU thread
tasks / 4
machines
Execute on GPU
GPU
result vector
GPU threads
Consolidate the results

MINMIN IMPLEMENTATION ON GPU

- Parallel synchronous MinMin in four GPUs and CPU
  - Also applies a domain decomposition, but it follows an hybrid approach.
  - In each iteration, each GPU performs a single step of the MinMin algorithm. Then a master process in CPU assesses the result computed by each GPU and select the one with the best MCT. Finally, the information of the selected assignment is updated in each GPU.



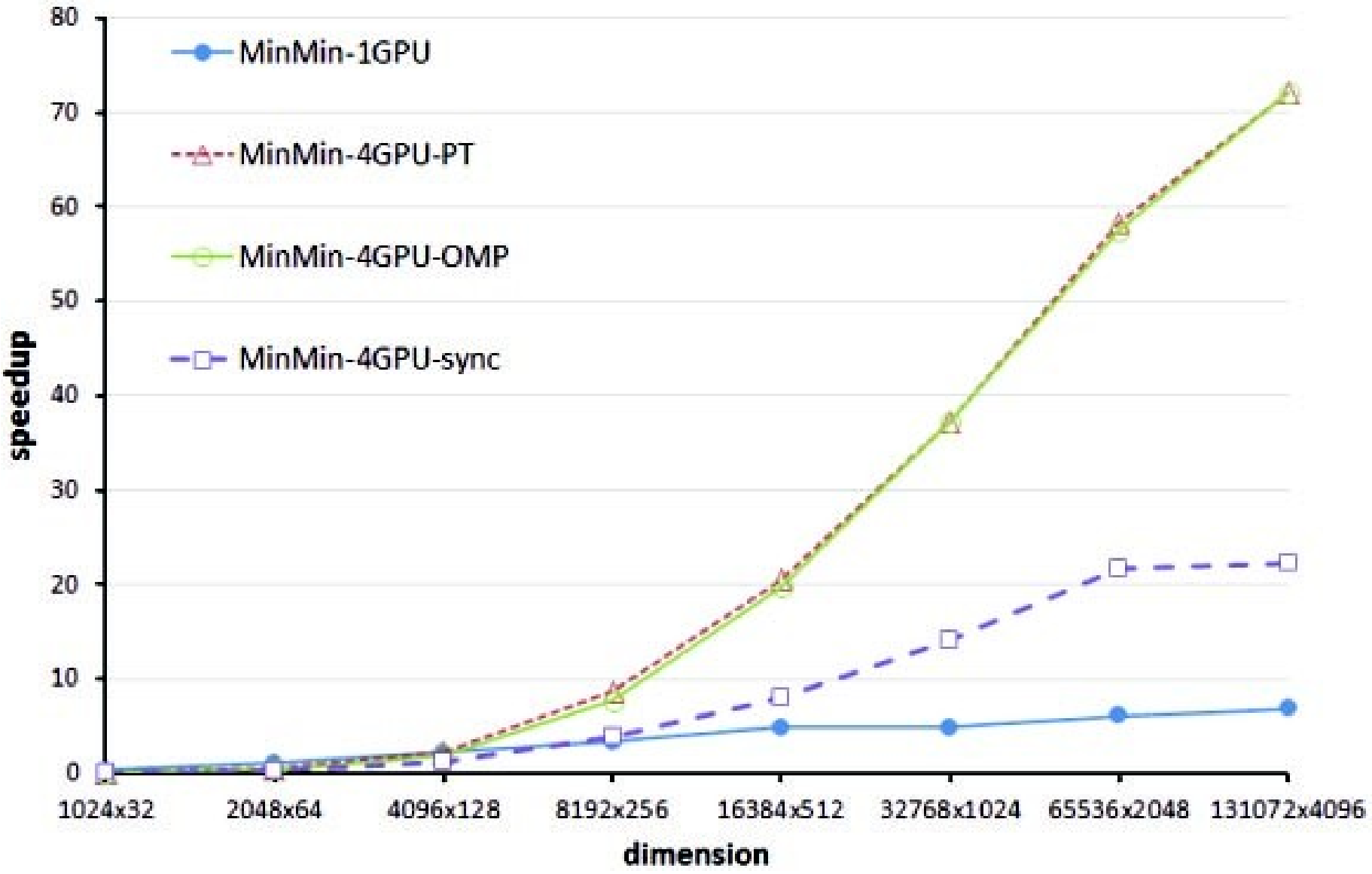
EXPERIMENTAL ANALYSIS

- Experimental platform
  - Dell PowerEdge, QuadCore E5530 at 2.27 GHz, 48 GB RAM, CentOS Linux 5.4.
  - Tesla C1060 Computing Processor (240 processor cores at 1.33 GHz, 4 GB).
- Test instances
  - 24 instances, with dimensions (tasks×machines): 1024×32, 2048×64, 4096×128 8192×256, 16384×512, 32768×1024, **65536×2048**, and **131072×4096**.
  - Far more larger than previously tackled instances.
  - 24 HCSP instances for each dimension, considering the combinations of heterogeneity and consistency according to the ETC model by Ali et al (2000).
- Metrics:
  - Makespan and execution time.

EXPERIMENTAL RESULTS

dimension	MinMin	MinMin-1GPU			MinMin-4GPU-PT		
	$t(s)$	$t(s)$	GAP	$speedup$	$t(s)$	GAP	$speedup$
1024×32	0.07	0.23	0.10%	0.31	0.88	20.00%	0.08
2048×64	0.39	0.37	-0.16%	1.06	0.95	28.33%	0.41
4096×128	2.25	1.02	0.13%	2.20	1.19	20.66%	1.89
8192×256	15.67	4.77	0.04%	3.28	2.03	19.90%	7.73
16384×512	119.74	24.83	-0.46%	4.82	6.08	20.45%	19.69
32768×1023	848.62	176.85	-0.11%	4.80	22.77	16.33%	37.28
65536×2048	6352.94	1049.34	0.11%	6.05	110.44	20.14%	57.52
131072×4096	49764.76	7253.88	0.04%	6.86	690.67	17.64%	72.03
dimension	**MinMin**	**MinMin-4GPU-OMP**			**MinMin-4GPU-sync**		
	$t(s)$	$t(s)$	GAP	$speedup$	$t(s)$	GAP	$speedup$
1024×32	0.07	0.83	20.00%	0.09	1.03	-0.07%	0.07
2048×64	0.39	0.89	28.33%	0.44	1.26	0.07%	0.31
4096×128	2.25	1.01	20.66%	2.21	1.95	-0.17%	1.15
8192×256	15.67	1.82	19.90%	8.62	4.16	0.05%	3.77
16384×512	119.74	5.84	20.45%	20.51	14.83	-0.28%	8.07
32768×1023	848.62	22.79	16.33%	37.23	60.16	-0.16%	14.11
65536×2048	6352.94	108.93	20.14%	58.32	292.75	-0.16%	21.70
131072×4096	49764.76	690.85	17.64%	72.05	2236.72	0.40%	22.25

SPEEDUP ANALYSIS



CONCLUSIONS

- Four MinMin implementations in CPU using CUDA, following a simple domain decomposition approach that allows scaling up to solve very large dimension problem instances (131072 tasks and 4096 machines).
- Experimental results demonstrated that the parallel MinMin in GPU provide significant accelerations over the time required by the sequential implementations for large HCSP instances.
- The speedup values raised up to a maximum of 72.05 for the parallel asynchronous MinMin implementation on four GPUs using OpenMP threads.
- The parallel synchronous MinMin using four GPUs computed exactly the same solution than the sequential MinMin, but improving the execution time in a factor of up to 22.25.

FUTURE WORK

- Improving the proposed GPU implementations, mainly by studying the management of the memory accessed by the threads.
- Make the implementation of other heuristics.
- This implementations for complement the efficient heuristic local search methods implemented on GPU.

THANKS FOR YOUR ATTENTION

FACULTAD DE INGENIERÍA, UNIVERSIDAD DE LA REPÚBLICA, URUGUAY