Task-to-Accelerator Mapping for Predictable OpenMP Applications

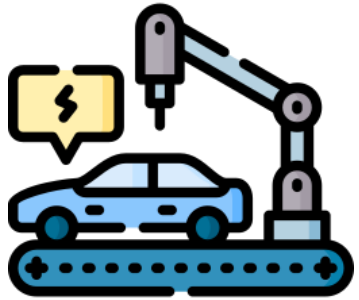
Mohammad Samadi Tiago Carvalho Luis Miguel Pinho Sara Royuela

SoftCPS Software Technologies for Cyber-Physical Systems Laboratory
ISEP Instituto Superior de Engenharia do Porto
INESCTEC
UPC Universitat Politècnica de Catalunya BarcelonaTech
BSC Barcelona Supercomputing Center Centro Nacional de Supercomputación

AEiC 2026

Introduction

Modern Critical Systems



Requirements

Hardware Capabilities

- Guaranteeing predictability
- Reducing response time and WCRT
- Reducing response time variability
- Enhancing high-performance capability

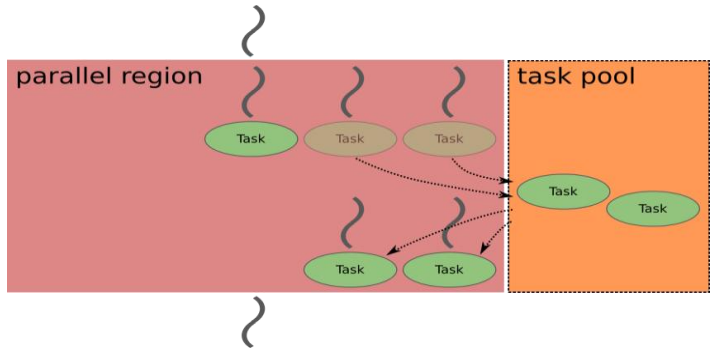
Modern Platform (e.g., NVIDIA DGX H200)

Major Challenges in Literature

- Many approaches are designed for single-GPU systems and fail to scale efficiently to multi-GPU environments.
- Others rely on complex runtime methods that may improve throughput but further reduce predictability.
- Blocking scheduling approaches reduce resource utilization and increase response time by preventing work-conserving behavior.

Real-Time Computing With OpenMP

Availability of portable, performant, and easily programmable parallel computing paradigms

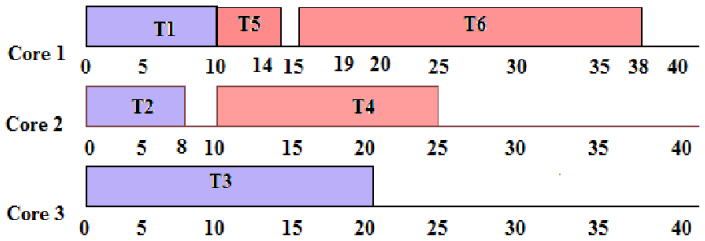


Why **OpenMP**?

OpenMP provides HPC capabilities in heterogeneous and highly parallel systems

OpenMP tasking allows real-time-systems-compatibility, enabling predictability

Need for real-time HPC behavior in the task-to-accelerator mapping



Needs

Improving work-conservation of the mapping

Improving load-balancing of the queues

Reducing contention on queues

Minimizing application response time

Significant Challenges With OpenMP

- GPU scheduling for OpenMP often focus on single-GPU systems or require extensive user intervention to exploit multi-GPU systems efficiently.
- Asynchronous target tasks are needed to avoid the busy-wait forced by synchronous execution on the GPU that reduces system efficiency.
- Most OpenMP implementations do not consider application and system-wide information to implement a mapping that minimizes response time and enhances predictability.

Main Contributions of This Work

- A set of lightweight heuristic algorithms for efficient task-to-accelerator mapping on heterogeneous platforms.
- A suspension-aware mechanism that increases CPU availability and resolves the busy-waiting problem in synchronous procedures.
- An evaluation of the proposal on OpenMP-compatible synthetic graphs under different settings.
- An assessment of the proposal using graphs representative of kernels of real-world applications under different configurations.

OpenMP

OpenMP Accelerator Model

```
#pragma omp task depend(out: var1)
 {/* host task code */}
 #pragma omp target nowait depend(in: var1) \\
       map(to: var2) map(from: var3)
 {/* target task code */}
 #pragma omp taskwait
```

— The target task depends on the host task.

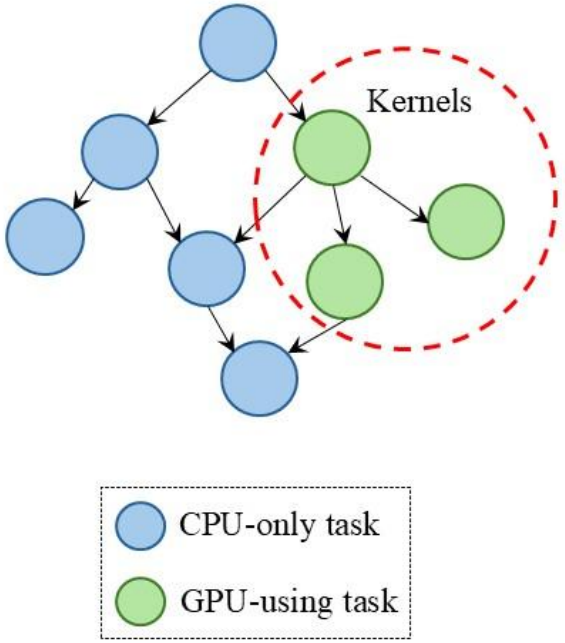
— The target task defines the data that has to be moved to and from the device using the *map* clause with the modifiers *to* and *from*, correspondingly.

— The target task is defined as asynchronous using the *nowait* clause.

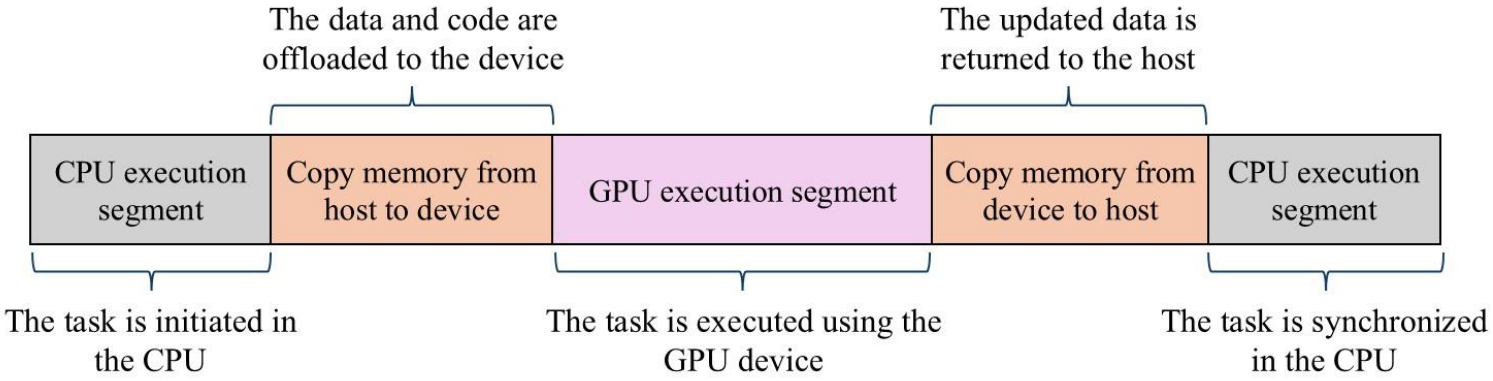
Proposed Mapping

System Model

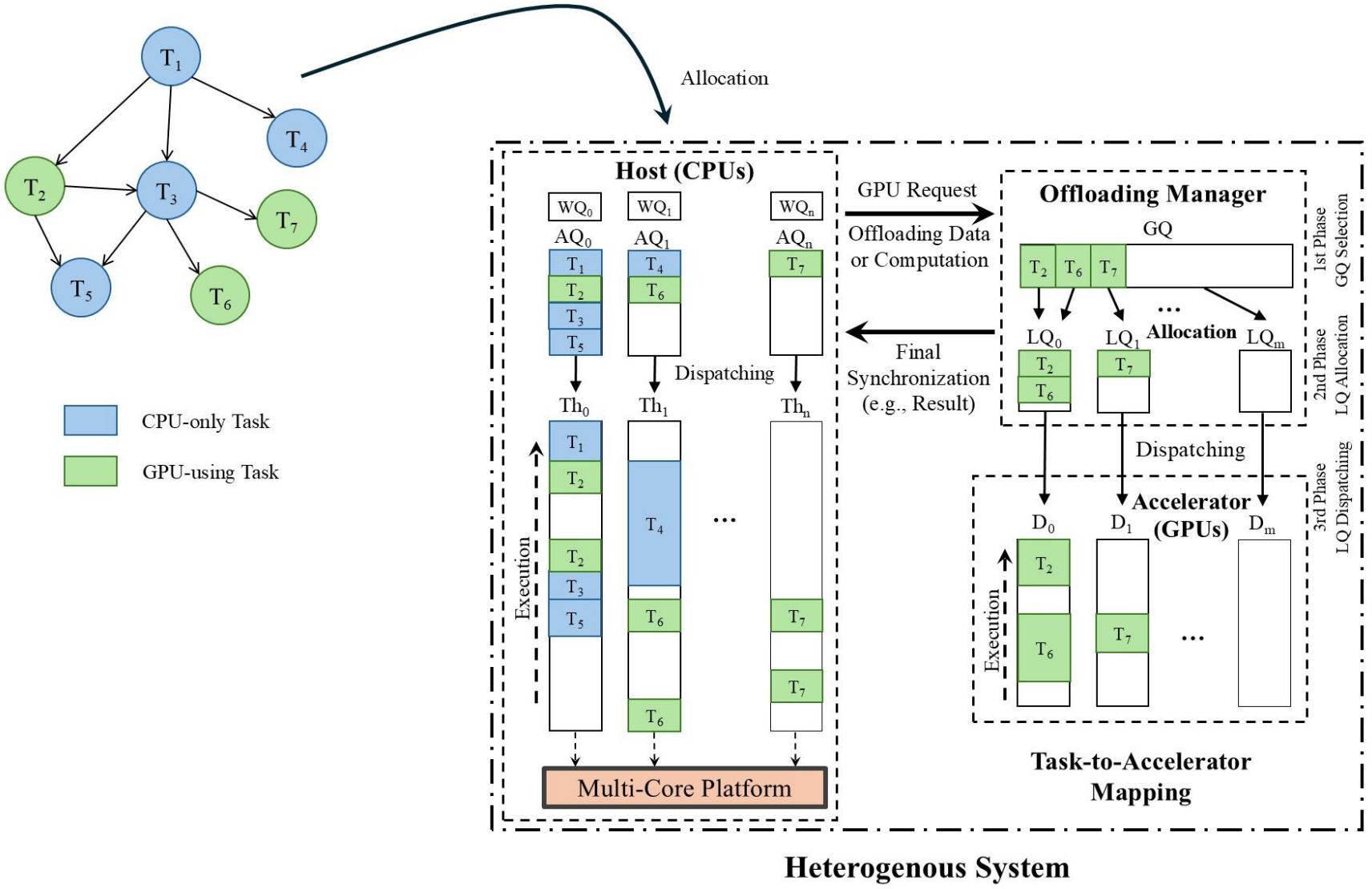
Target Application:
a heterogeneous program
represented as a computation graph



Main Parts of OpenMP Target Tasks
(those potentially using the GPU)



Methodology



Selection, Dispatching, and Allocation Algorithms

GQ Selection and LQ Dispatching

LET	Least Execution Time
MNAOT	Maximum Number of All Outgoing Tasks
SWSM	Weighted-Sum Model

LET creates nested tasks sooner.

MNAOT makes most successors ready for execution sooner.

SWSM combines LET and MNAOT.

$$SWSM_i = ET_w * ET_i + NAOT_w * \frac{1}{NAOT_i}$$

LQ Allocation

LTET	Least Total Execution Time
MNJ	Minimum Number of Jobs
AWSM	Weighted-Sum Model

MNJ improves the load balancing of queues based on the number of tasks.

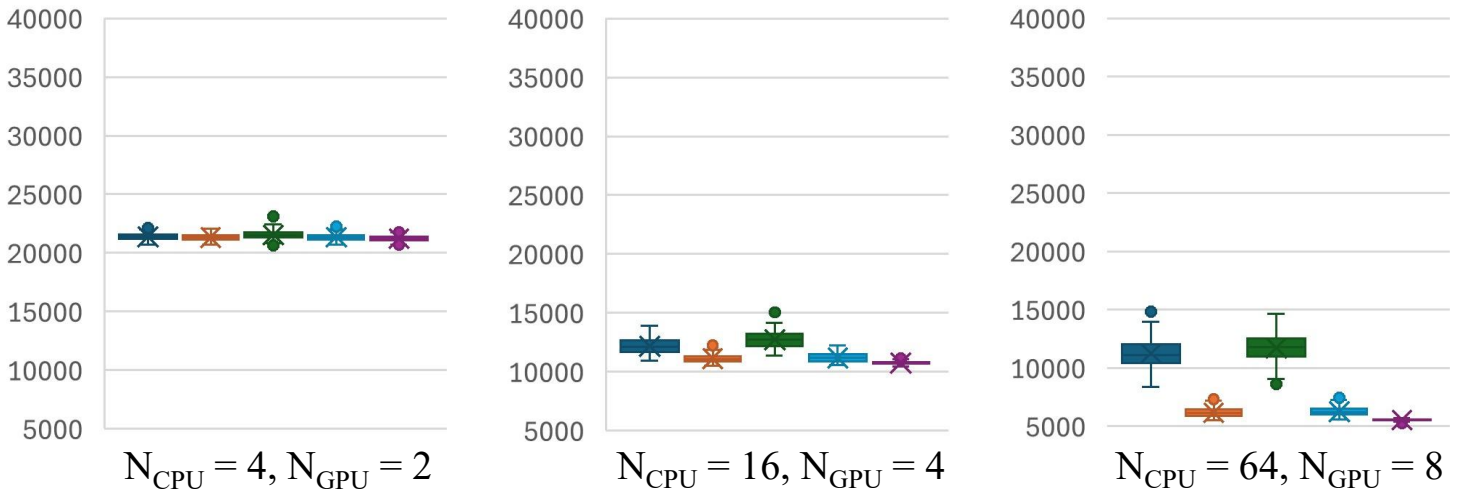
LTET enhances the load balancing of queues based on task execution time.

AWSM combines MNJ and LTET.

$$AWSM_i = TET_w * TET_i + NJ_w * NJ_i$$

Evaluation

Representative Results: Synthetic Graphs



$N_{CPU} = 4, N_{GPU} = 2$ | $N_{CPU} = 16, N_{GPU} = 4$ | $N_{CPU} = 64, N_{GPU} = 8$

System setup
- Highly-parallel graphs
- Task execution time: [1, 50] *ms*
- Running with different CPUs and GPUs

Baseline | O-KGLP | TET-BF-TET | TET-WF-TET | Proposed

Proposed: LET for GQ selection
LTET for LQ allocation
MNAOT for LQ dispatching

The proposed method reduces WCRT and response time variability, as the algorithms:

— Select the task with the shortest execution time from GQ to execute dependant task(s) earlier, improving the work-conservation of the mapping.

— Choose local queues with the shortest total execution time, improving the load balancing of the system.

— Select tasks with the largest number of all outgoing tasks from LQs, leading to an increase in the number of ready tasks.

The method's effectiveness increases with more computing elements, revealing numerous parallel tasks in this configuration, and therefore, an optimal selection of local queues and tasks.

Simulation Setup: Real-World Applications

Axpy

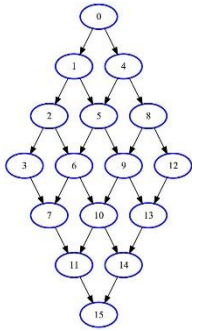
Embarrassingly parallel.
Basic linear algebra operations (a*x***y*).

Number of tasks: 109
Number of data dependencies: 0

Dotp

Embarrassingly parallel.
Basic linear algebra operations (Σ*x***y*).

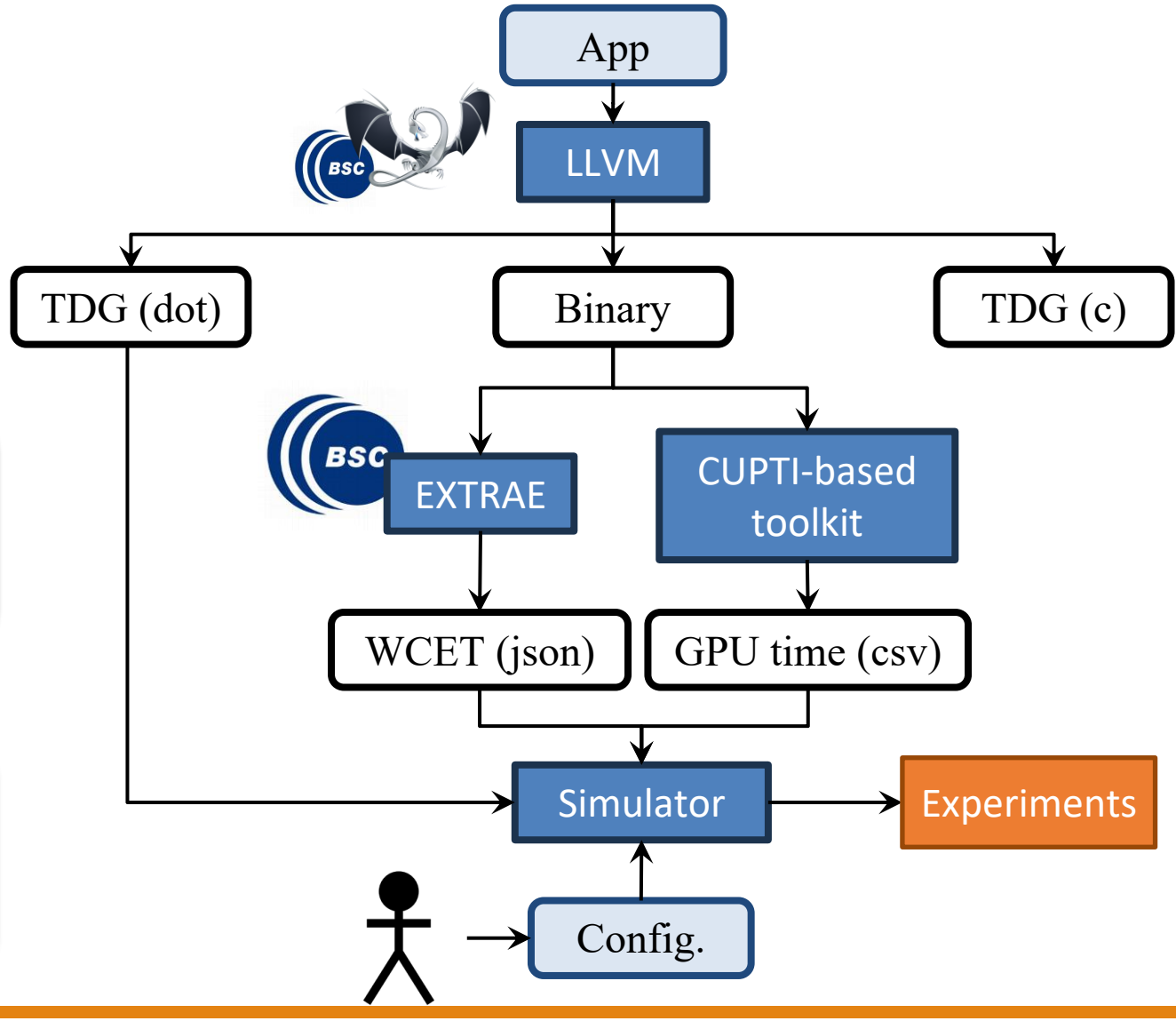
Number of tasks: 155
Number of data dependencies: 0



Heat

Wavefront parallelism based on stencil.
Heat diffusion based on the Gauss-Seidel.

Number of tasks: {128, 220}
Number of data dependencies: {336, 400}



Representative Results: Real-World Applications

	Axpy (num_tasks = 109)			Dotp (num_tasks = 155)		
N_{CPU}, N_{GPU}	8, 4	16, 4	32, 8	8, 4	16, 4	32, 8
Baseline	49.70876	49.11773	26.75326	44.714827	44.406787	23.697888
O-KGLP	49.70783	49.18778	26.30887	44.752739	44.8702	23.256691
TET-BF-TET	48.57896	48.12031	25.64134	44.582951	44.572147	22.904986
TET-WF-TET	48.57653	48.07965	25.6247	44.582	44.559589	22.892224
LET-LTET-LET	48.57644	48.08009	25.62136	44.582	44.559589	22.892216
LET-LTET-MNAOT	48.57644	48.08009	25.62136	44.582	44.559589	22.892216
LET-LTET-SWSM	48.57644	48.08009	25.62136	44.582	44.559589	22.892216
LET-MNJ-LET	48.57653	48.07965	25.6247	44.582	44.559589	22.892224
LET-MNJ-MNAOT	48.57653	48.07965	25.62208	44.582	44.559589	22.892224
LET-MNJ-SWSM	48.57653	48.07965	25.6247	44.582	44.559589	22.892224
LET-AWSM-LET	48.57644	48.08009	25.62136	44.582	44.559589	22.892216
LET-AWSM-MNAOT	48.57644	48.08009	25.62136	44.582	44.559589	22.892216
LET-AWSM-SWSM	48.57644	48.08009	25.62136	44.582	44.559589	22.892216

— Most of the proposed algorithms perform efficiently than those in the literature, in most cases.

— LET-X-X performs more efficiently than MNAOT-X-X and SWSM-X-X, as selecting tasks with the shortest execution time improves the mapping.

Future Work

Future Work

- Performance evaluation of the proposed mapping using experimental results obtained for real-world OpenMP applications, running on different hardware platforms from embedded and high-performance computing domains, including supercomputing nodes with large computing resources.
- Execution of tasks using multiple GPUs in the same transaction, addressing memory management optimization.

Thank You!

Mohammad Samadi

MMASA@ISEP.IPP.PT

Find the code to this project on GitHub.

SOFTCPS SOFTWARE TECHNOLOGIES FOR CYBER-PHYSICAL SYSTEMS LABORATORY

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

INESCTEC

UPC UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

BSC Barcelona Supercomputing Center Centro Nacional de Supercomputación