January-July 2020

Automatic configuration of the Heteroprio scheduler

Research internship

Academic tutor

Julien NARBOUX

IGG team

Author

Clément FLINT

Supervisor

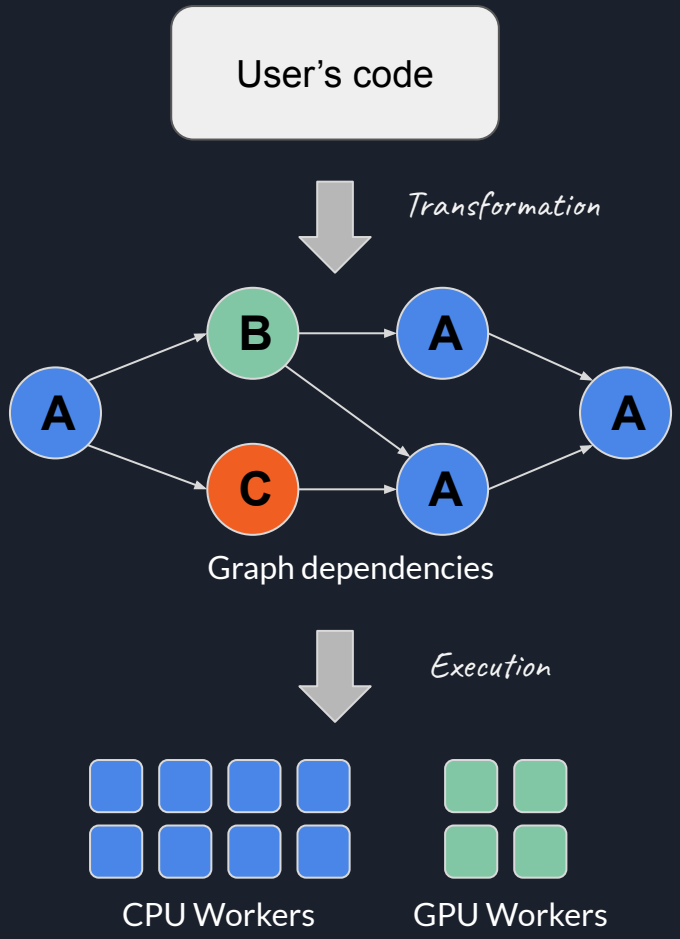
Bérenger BRAMAS

CAMUS team

Context

StarPU

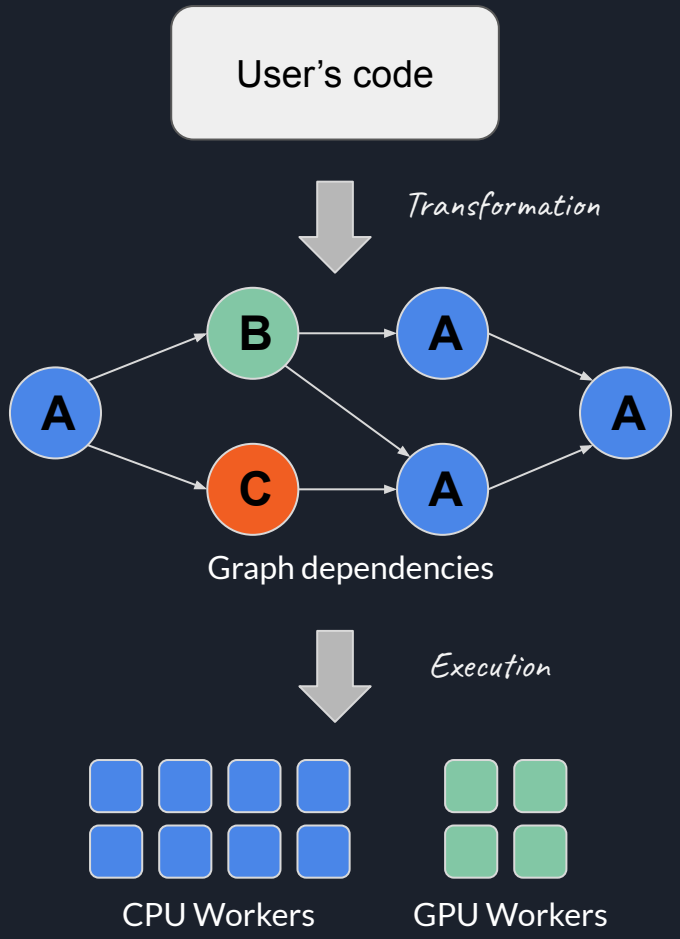
- Task programming library for hybrid architectures
- The computation has to be divided into ***tasks***
- Automatically handles ***task dependencies***
- The workload is dynamically assigned to the workers (e.g. ***CPU*** and ***GPU***)
- The choice of worker for a task is taken by the ***scheduler***



Context

StarPU - scheduler

- A worker can usually choose between multiple ready tasks
- Finding the best scheduling is known to be ***NP-complete*** in the general case
- Different strategies can be adopted to perform efficient scheduling
- Multiple schedulers available in StarPU : eager, lws, ***heteroprio*** ...

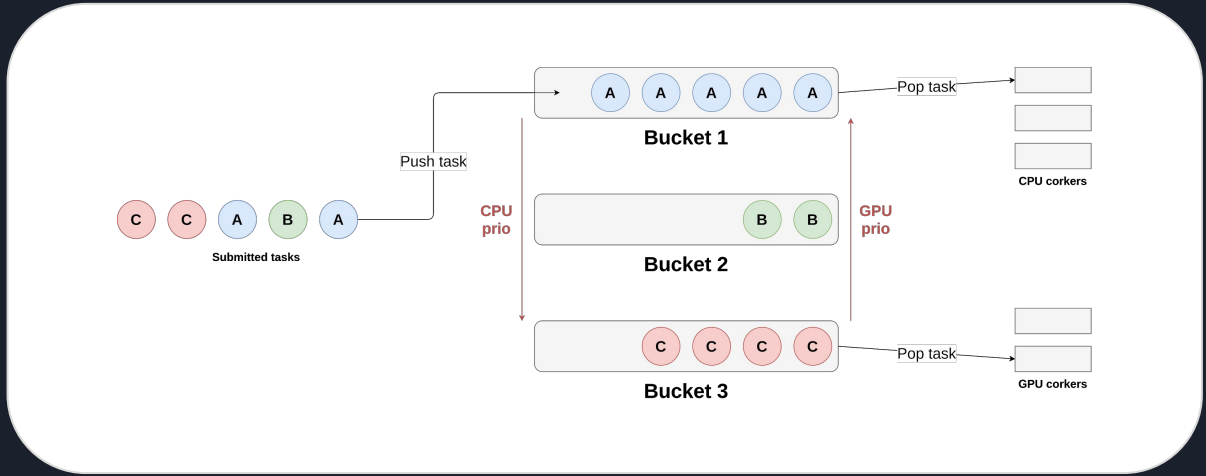


Context

Heteroprio

- One of the base ***StarPU*** schedulers
- Efficient for handling CPU/GPU workload
- Is based on CPU and GPU ***user-defined priorities***

Context



Heteroprio :

Schematic of Heteroprio's functioning

- New tasks are added to "buckets"
- Workers retrieve tasks from buckets depending on their priority

Context

Heteroprio

Pros

- Highly efficient when tasks have a clear processor affinity
- Low latency

Cons

- The CPU and GPU priorities have to be set manually
- The scheduling efficiency entirely depends on the priorities

Context

Possible improvement for Heteroprio : assign priorities ***automatically***.

Indeed, setting priorities manually :

- requires the user to study the execution times on each architecture
- implies that the user has a good understanding of Heteroprio's inner mechanism

Goal of this project : create a version of Heteroprio that requires ***no user interaction***

Methodology

Challenges

- Extract local graph features
- Extract relevant quantities (e.g. estimated execution times)
- Generate CPU and GPU priorities based on heuristics

Methodology

Typical used quantities

- Number of successors
- Normalized Out-Degree
- CPU-GPU time difference
- Normalized Released Time

$$NOD(v_i) = \sum_{v_j \in succ(v_i)} \frac{1}{ID(v_j))}$$

Normalized Out-Degree

$$NRT_{arch}(v_i) = \sum_{v_j \in succ(v_i)} \frac{P(exec(v_j, arch)) \cdot w_j^{arch}}{ID(v_j))}$$

Normalized Released Time

Methodology

$$\mathrm{NOD/s}_{arch}(v_i) = \frac{NOD(v_i)}{w_i^{arch}}$$

score of the NOD/second heuristic

Example of a heuristic

Normalized Out-Degree (NOD) per second

- Measures the speed at which successors are released

We have found **27** other heuristics

Methodology

First step : Create heuristics for generating efficient priorities

Methodology for validating our heuristics

- Generation of a dataset, 32 random application graphs
- Creation of control priorities, generated by an iterative optimization algorithm
- We evaluate the efficiency of a heuristic by comparing its makespans on a simulator against the control priorities' ones

Methodology

Second step : implement the heuristics in Heteroprio

Modify Heteroprio so that it be able to exploit data and use heuristics

Methodology

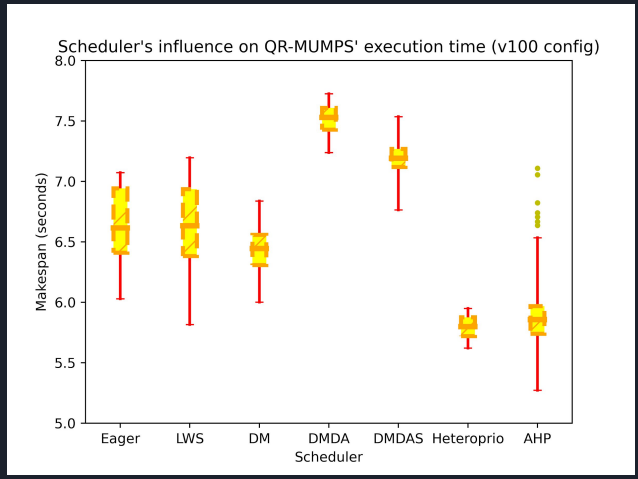
Finally, we tested our scheduler on **QR-MUMPS**, an application that uses **StarPU**

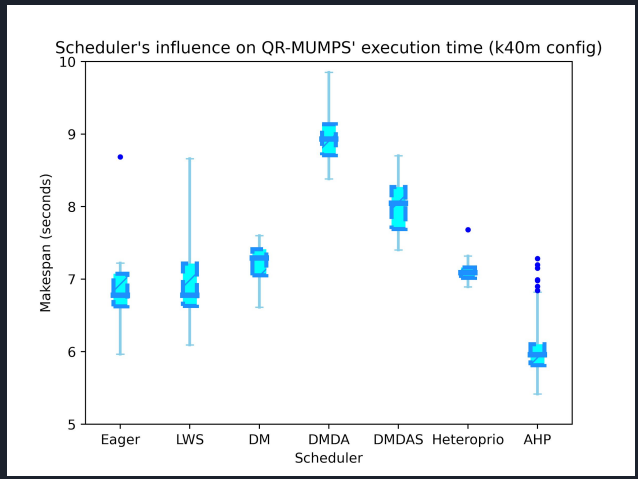
- Execution on the PlaFRIM cluster
- 2 different GPU architectures (v100, k40m)
- Measured variable : makespan of a QR factorization
- Parameters : configuration, scheduler (Heteroprio vs others) and handmade vs automated priorities

Results

Comparison of the schedulers

- AHP = Heteroprio with automated priorities
- AHP is **equivalent** to Heteroprio on the v100 architecture
- AHP is significantly **superior** to Heteroprio on the k40m configuration





Discussions

According to our tests :

- Using automated priorities does not hurt Heteroprio performances
- In some cases, automated priorities can even be significantly faster than the optimized static ones

Discussions

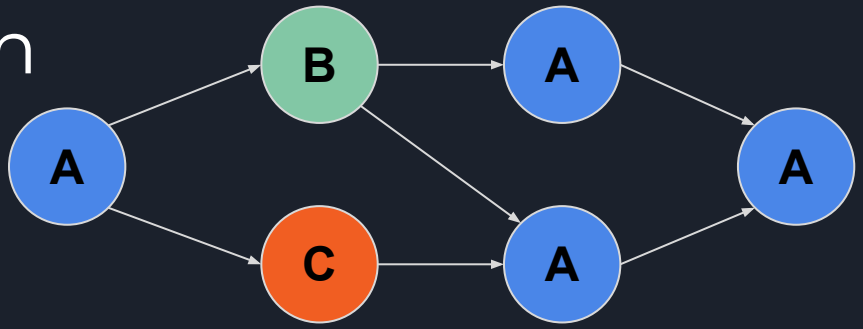
We have demonstrated that automated priorities can perform as well, or better than handmade ones on QR-MUMPS, but :

- Does our method perform well on other applications than QR-MUMPS ?
- Are there realistic edge cases where the scheduling is particularly bad ?

We are planning on conducting a test on a larger set of applications.

Thank you for your attention

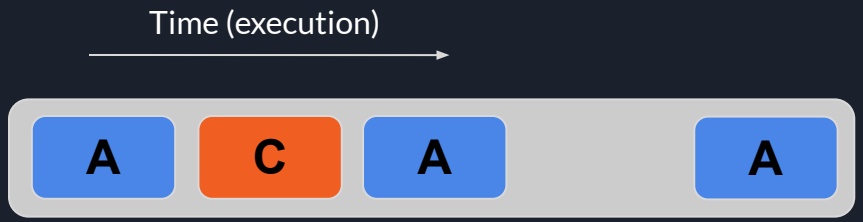
Example execution



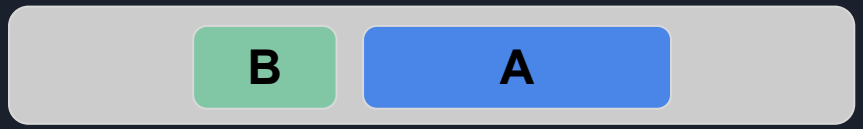
Dependency graph

Task type	CPU time	GPU time
A	1s	2s
B	2s	1s
C	1s	1s

Execution time of each task



CPU worker



GPU worker

CPU priorities : **A > C > B**

GPU priorities : **B > C > A**