Investigating Memory Operations Performance in the StarPU Runtime

Lucas Leandro Nesi, Lucas Mello Schnorr

September 5, 2018

Graduate Program in Computer Science (PPGC/UFRGS), Porto Alegre, Brazil

Summary

Introduction

- Programming for HPC is complex
- Using classical paradigms, the programmer has to:
  - Map computation to resources
  - Manage the data and communication
- The task-based paradigm facilitates the programming by abstracting those responsibilities to a runtime
  - Runtime example: StarPU

Problem

- Similar to any other HPC approach, performance analysis of task-based applications is laborious
- Some tools for performance analysis of task-based applications are available with different features
  - Example: StarVZ
- However, those tools lack features for memory management analysis of the runtime and the application

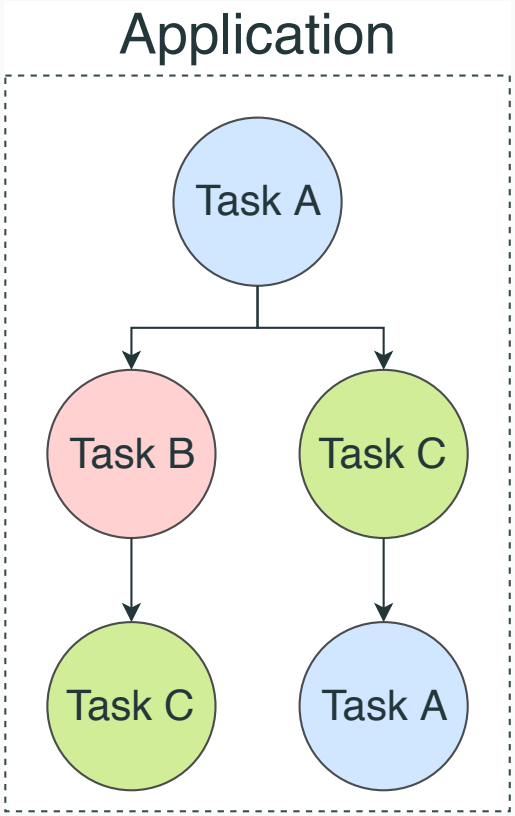
Objectives

Create mechanisms for data management performance analysis of task-based applications running over the StarPU runtime

Basic Concepts

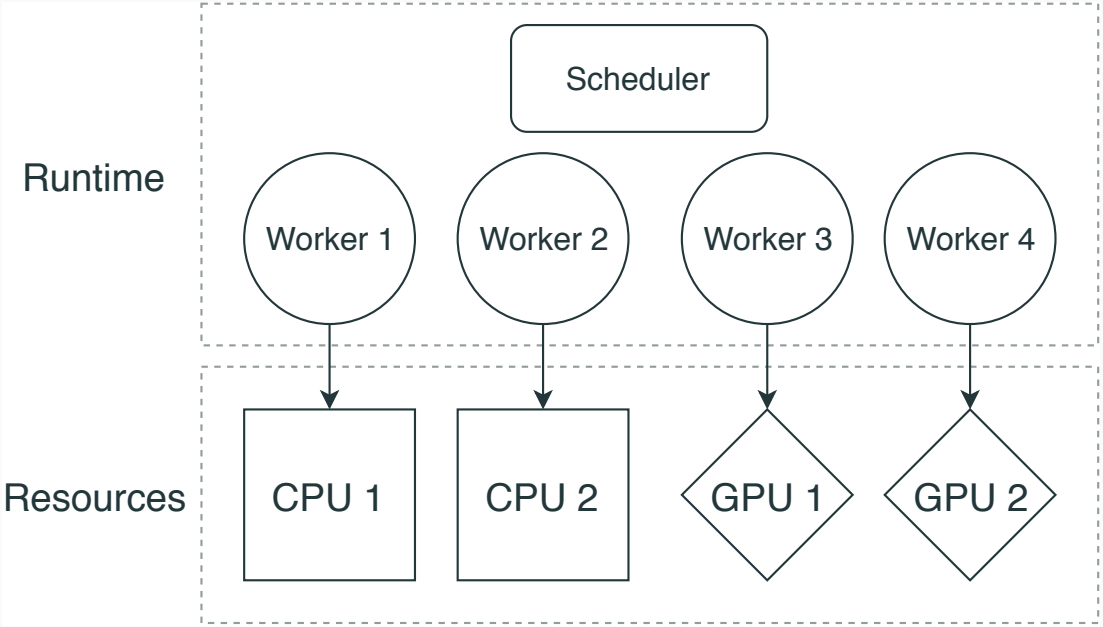
Task-Based Programming Paradigm

- The applications consist of a set of tasks
- The tasks are organized using a DAG (*Directed Acyclic Graph*)
  - The graph links are memory dependencies between tasks
- The applications run over a runtime



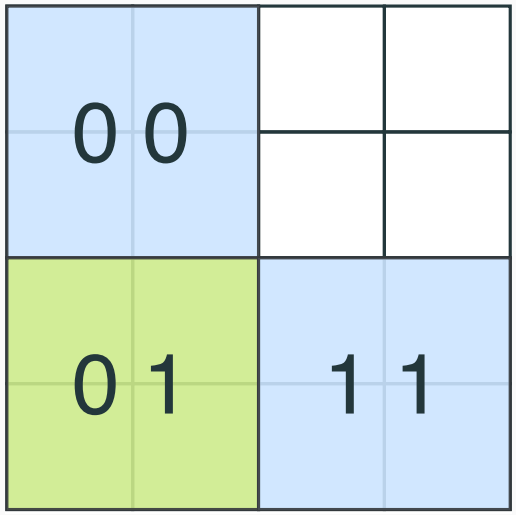
Task-Based Programming Paradigm

- The runtime creates workers for system resources
- The tasks are submitted by the application to the runtime
- The runtime will schedule the tasks to workers based on scheduler heuristics, examples are:
  - `LWS` (Local work stealing)
  - `DM` (deque model)



StarPU - Memory Blocks

- Data is organized into memory blocks
  - Example: The matrix is divided into memory blocks

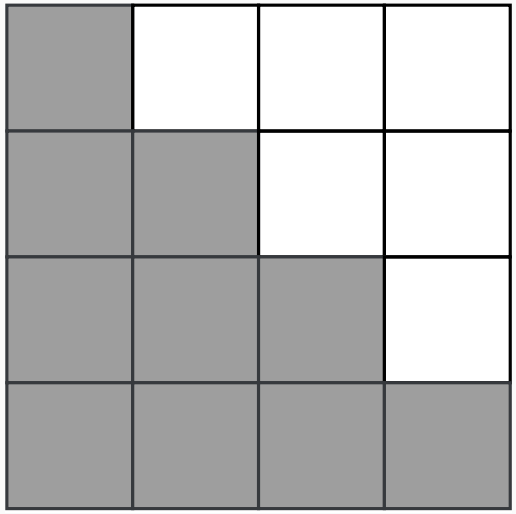


Data Management

- Each resource memory has a entity called memory manager
- The memory block is the minimal runtime memory unit:
  - Tasks uses memory blocks as inputs
  - The DAG dependencies are reuse of memory blocks
  - Blocks are transferred between resources
- Coherence between resources: `MSI` System:
  - Three possible states for each block: `Modified`, `Shared`, `Invalid`
  - Each memory block has one state per resource memory

Application example: Chameleon/MORSE

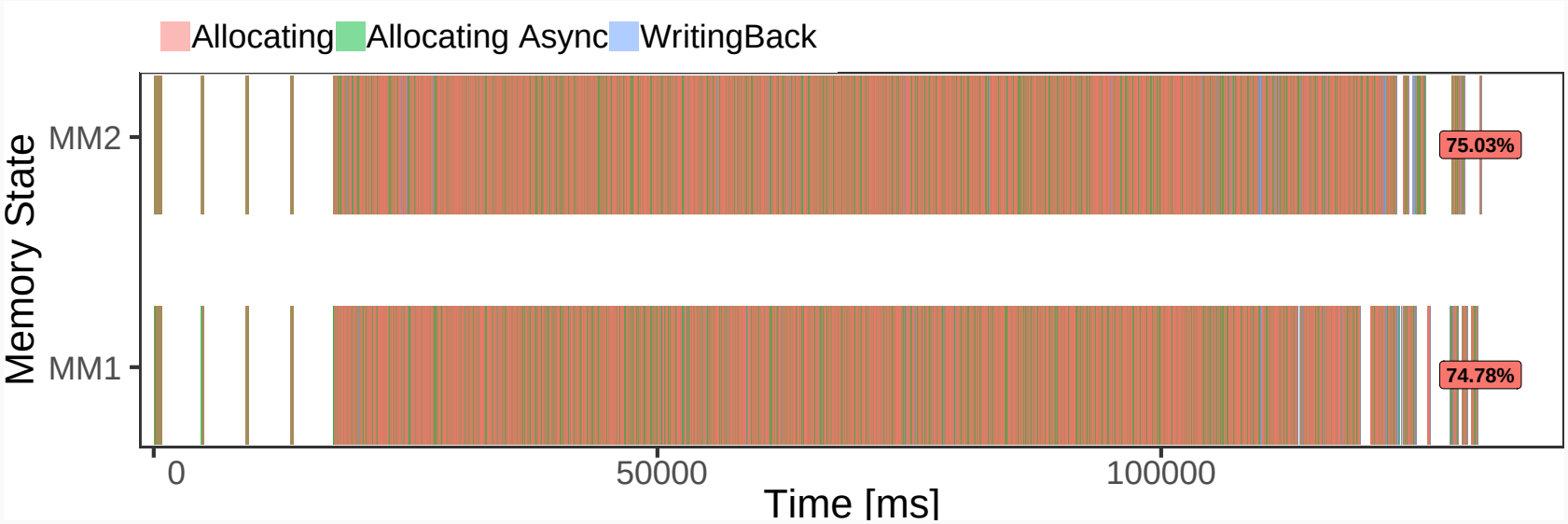
- Solver for dense linear algebra using task-based paradigm
- One of the the possible operations: Cholesky factorization
  - Uses a triangular matrix
- Four tasks: `dportf`, `dsyrk`, `dtrsm`, `dgemm`
- Compute from lower memory blocks coordinates to higher ones



Analyzing Data Management

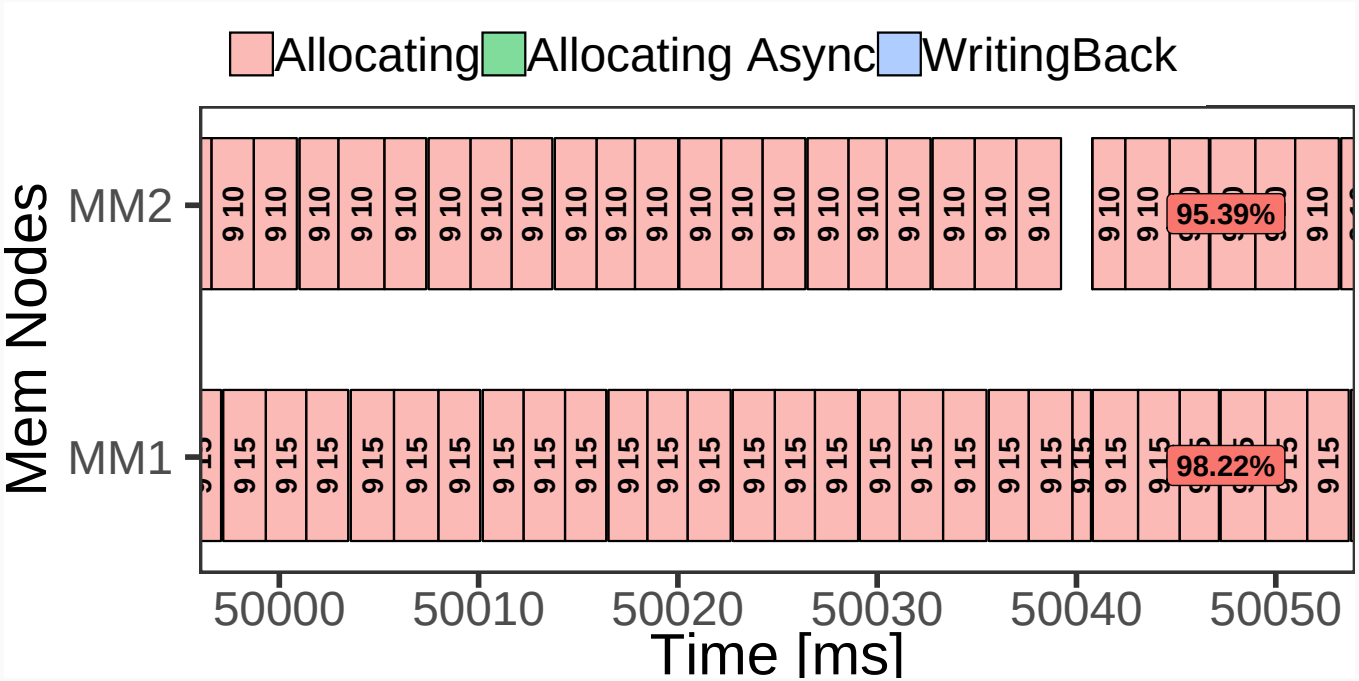
Memory Nodes States

- Uses the traditional Gantt Chart for memory managers states
- X is time in ms and Y is the Memory Managers



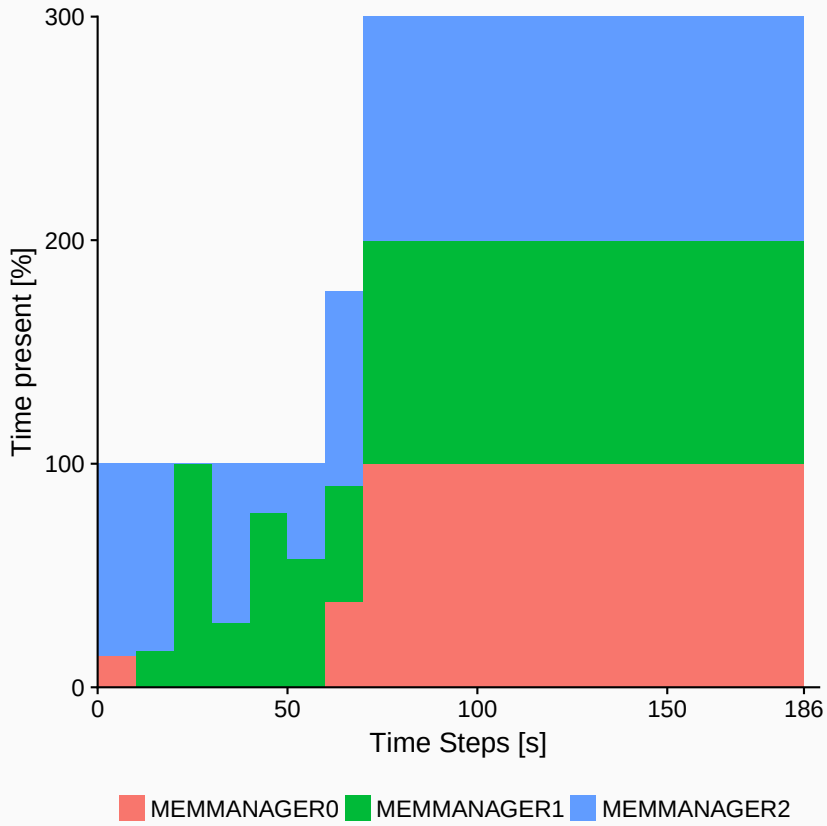
Memory Nodes States - Zoom

- If zoomed it shows the coordinates of the states' related to memory blocks



Memory Block Residency

- Show the residency of a memory block on the memory managers during the execution
- X is the time divided into intervals, and Y is the time that the memory block was present on the memory manager



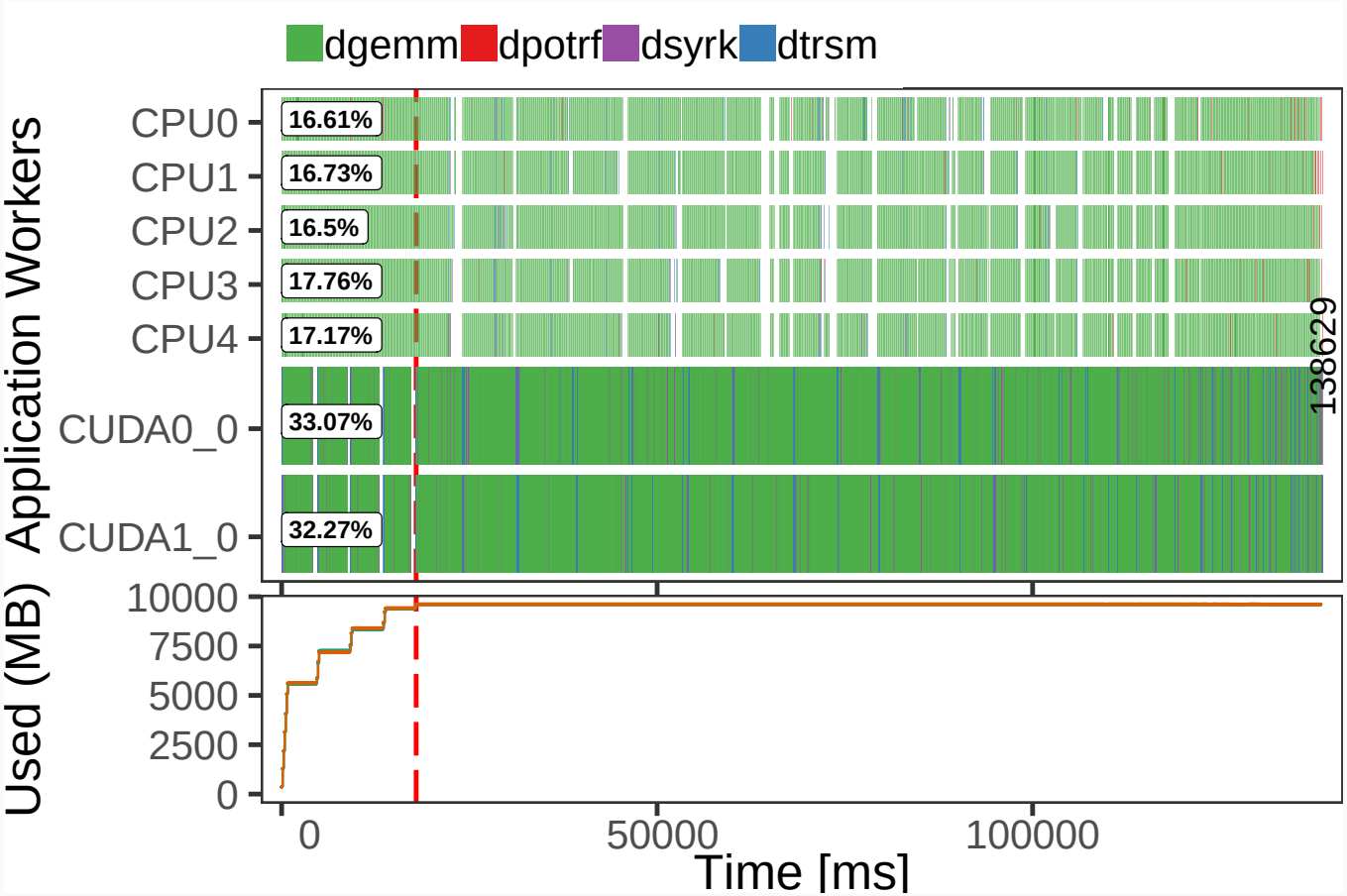
Experiments

Experiments

- Test Case: Chameleon/Morse Cholesky Factorization
  - Block size: 960x960
  - 60x60 tiles.
  - `DMDA` Scheduler
  - Machine: `Tupi`
- Preliminary testes suggested performance problems
  - Unknown reasons.
- Using the earlier methodology to investigate

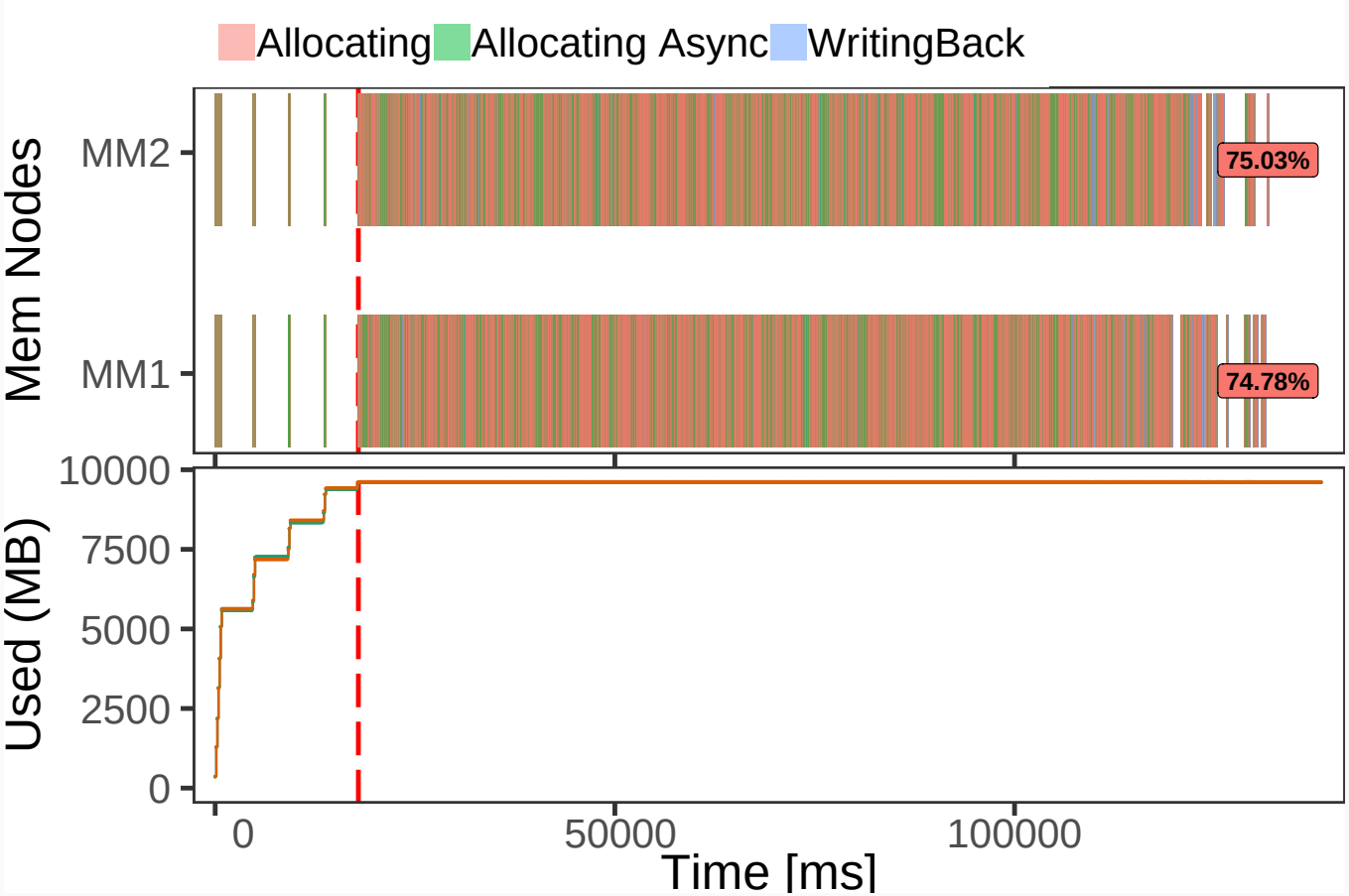
Application Workers

- GPUs have high idle times after used memory reaches a plateau



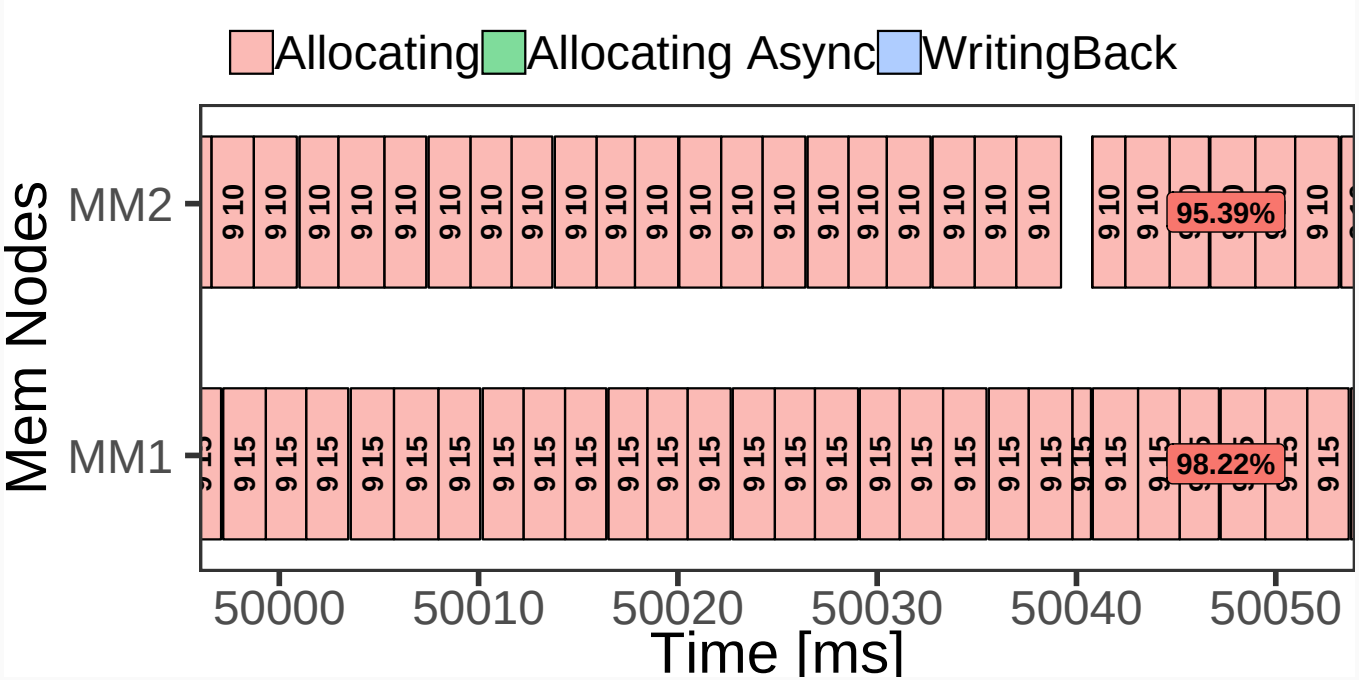
Memory Managers

- GPUs' Memory Managers with many allocation's states



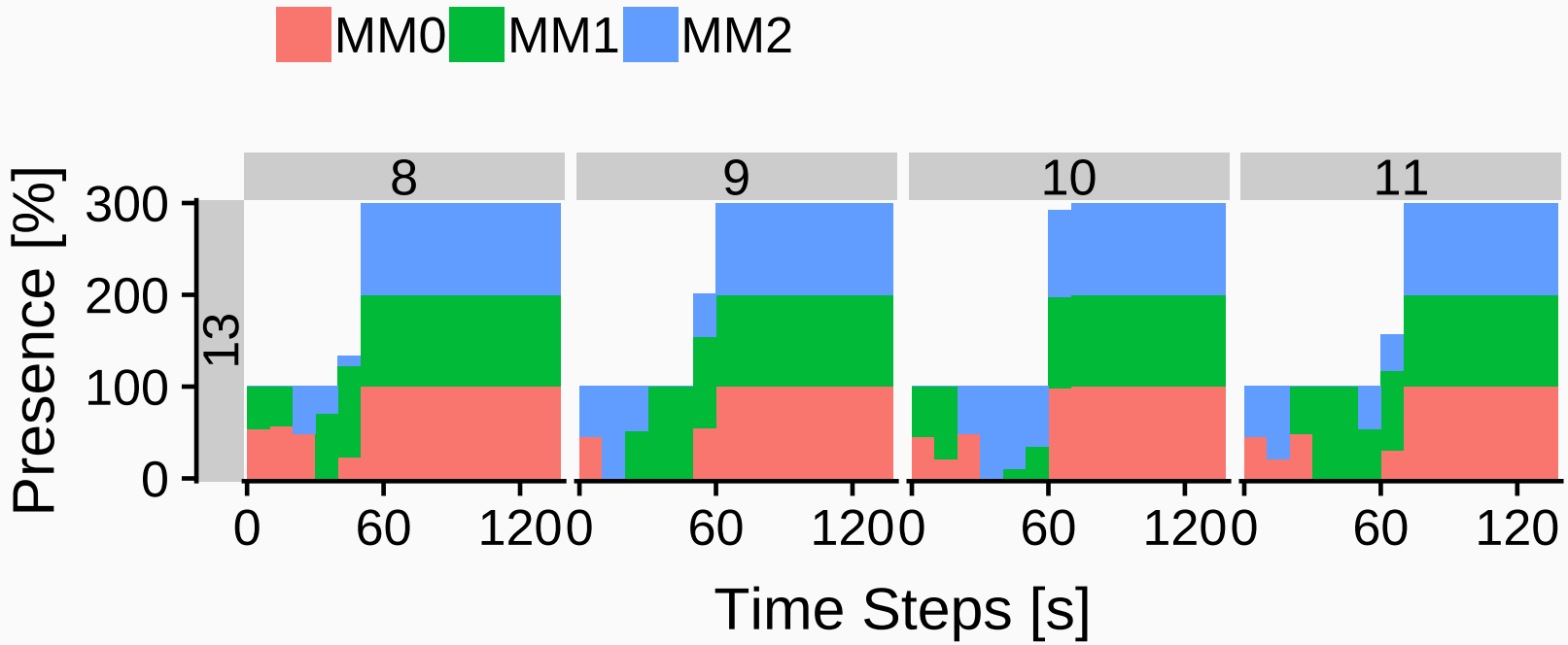
Zoom Memory Nodes

- Repeated allocations for the same memory blocks
- Inspecting StarPU code: this could happen if the allocations fail



Residency of Blocks

- Memory Blocks remain on resources' memory even if they aren't needed anymore



Code Inspection

At this point:

- We think that the allocations are falling because the memory is full
  - `nvidia-smi` confirms it
- However, if StarPU identify that the memory is full, it *should* free unused blocks.
  - So StarPU believes it has free memory!
- We inspect the execution using `gdb` and identify a disparity between the internal StarPU GPUs used memory values and the real ones

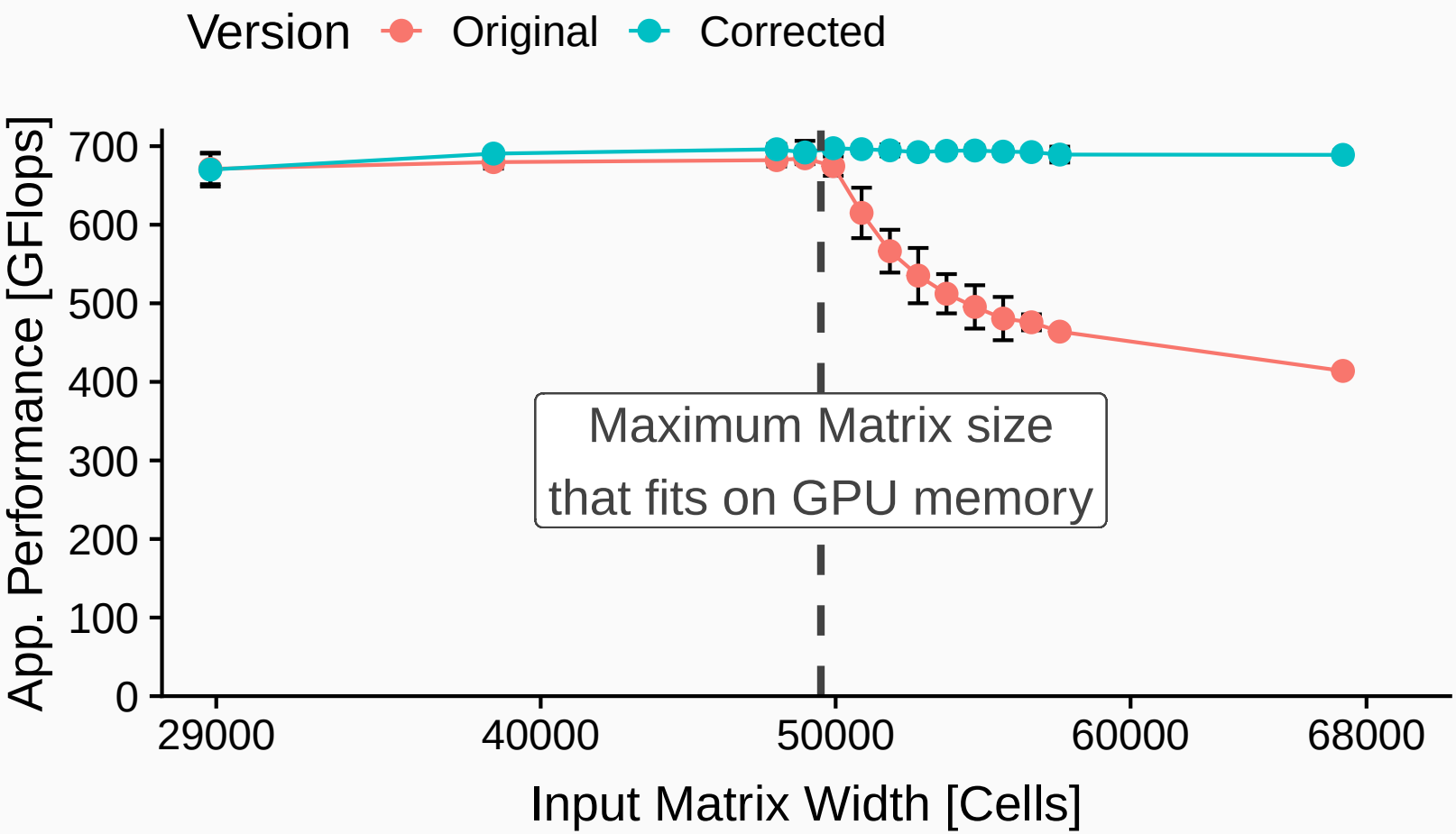
Problem Cause

- StarPU internally register the used memory by the values passed to `cudaMalloc`
- However, `cudaMalloc` can reserve more memory than the requested
  - Page size of the device; in our case: 2MB
  - Causing a deviance of 1.8GB for the matrix
- We propose a patch for correcting it

Patch Experiments

- 10 executions comparing the original and the corrected version.
- Using the earlier Cholesky parameters:
  - Block size: 960
  - `tupi` machine
  - `DMDA` Scheduler
- Varying the size of the matrix
- 99% of confidence

Patch Results



Conclusion

Conclusion

- Memory Manager is a important role in StarPU's Applications
- The proposed visualizations give extra information on the memory management decisions
  - Application independent
- We solved a runtime problem that was discovered using the new features
  - Performance gains of 66%
  - Application Performance sustain independent of matrix size

Investigating Memory Operations Performance in the StarPU Runtime

Lucas Leandro Nesi, Lucas Mello Schnorr

September 5, 2018

Graduate Program in Computer Science (PPGC/UFRGS), Porto Alegre, Brazil