Ionic models implementation update: multi-target (CPU+GPU), StarPU interface, and code improvements

Vincent Alba (University of Bordeaux) & Raphaël Colin (University of Strasbourg)

July 05, 2023

3rd Microcard Workshop - Strasbourg

"The project leading to this application has received funding from the European High Performance Computing Joint Undertaking Joint Undertaking (JU) under

Section 1

Computing Ionic Model on Heterogeneous Architecture

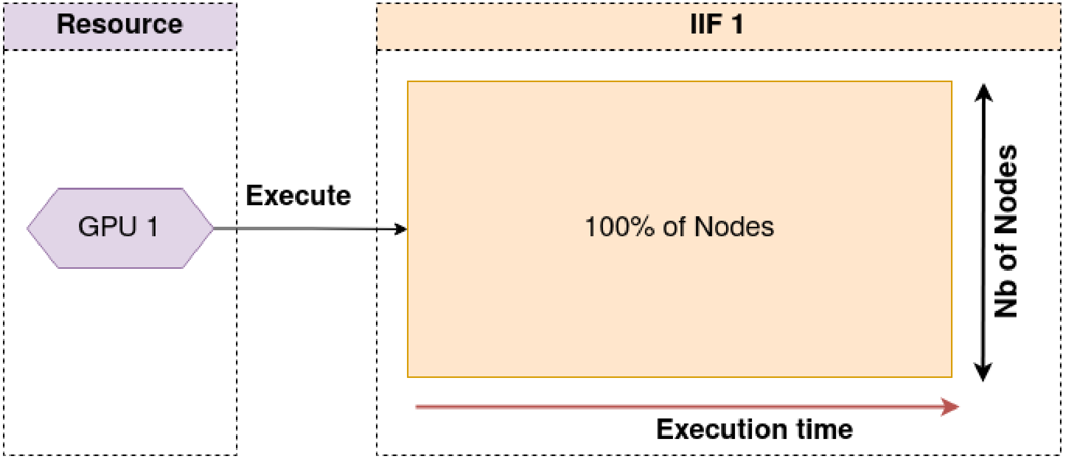
Computing Ionic Model on Heterogeneous Architecture

Bringing openCARP to Exascale

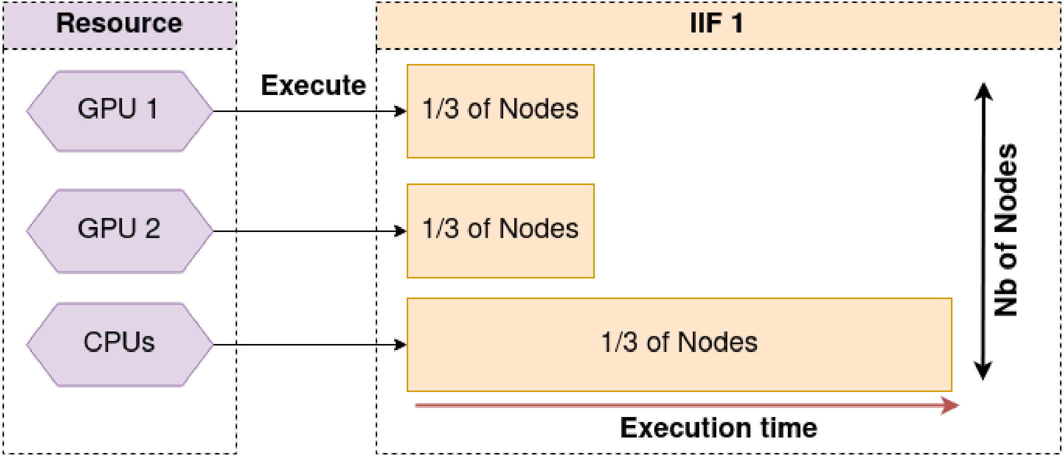
Simulation needs to be adapted to exploit heterogeneous architectures

- What needs to be changed?
- What issues does this raise?

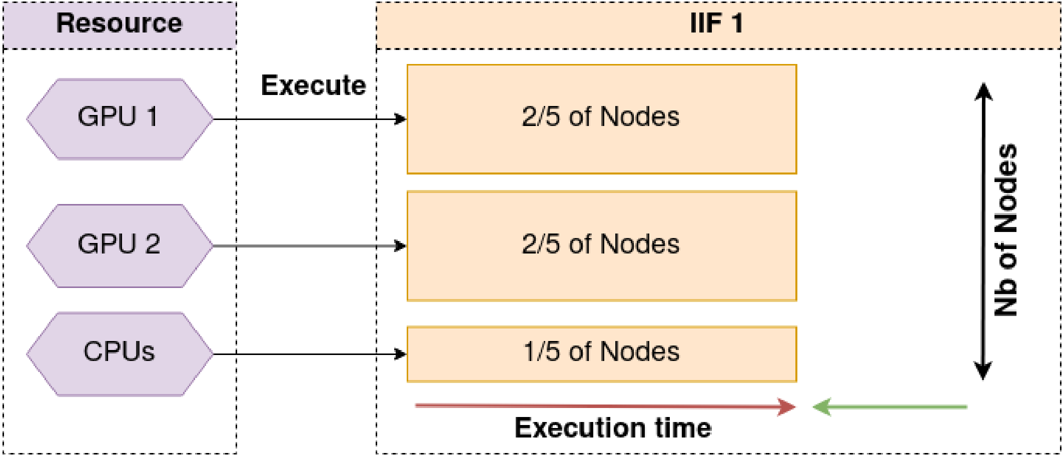
Problematic : Base version



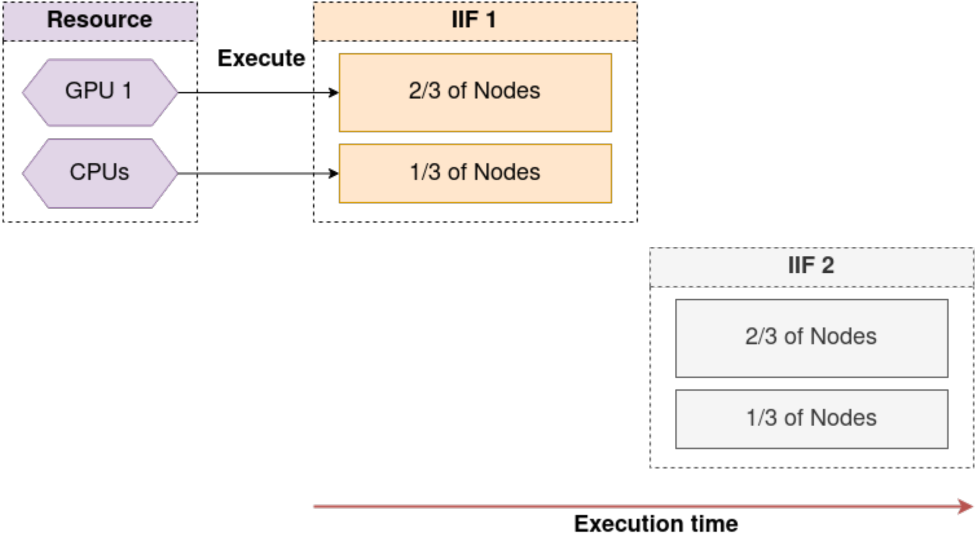
Problematic : Heterogeneous version



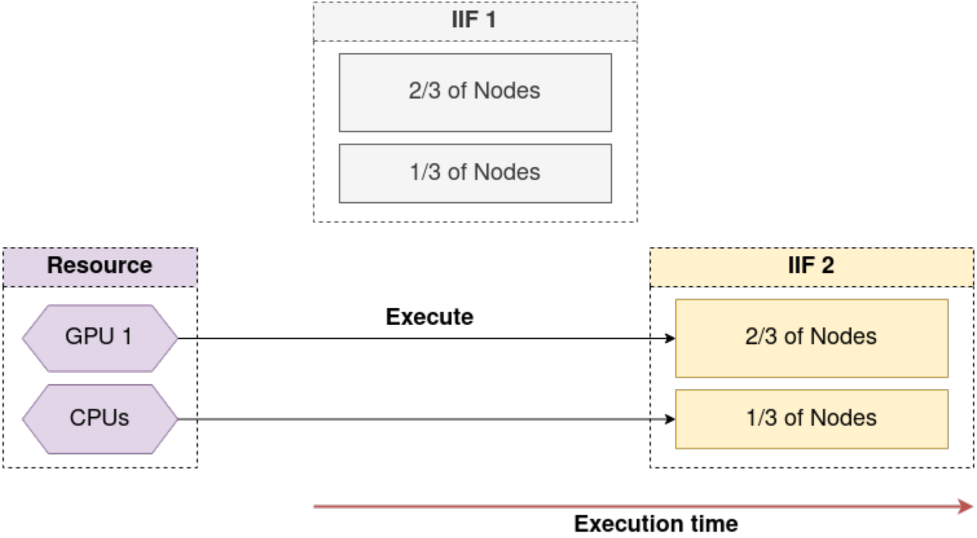
Problematic : Load Balancing



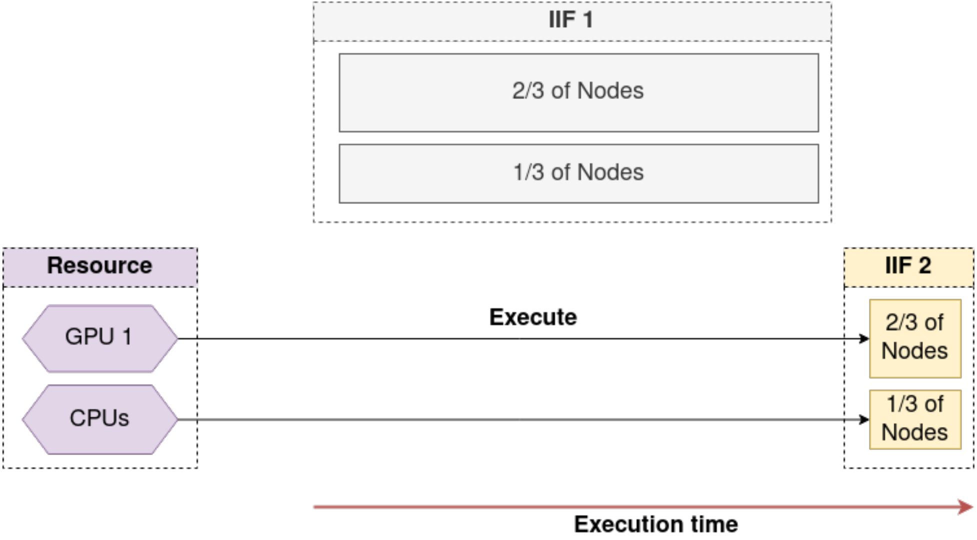
Problematic : Multi IIF



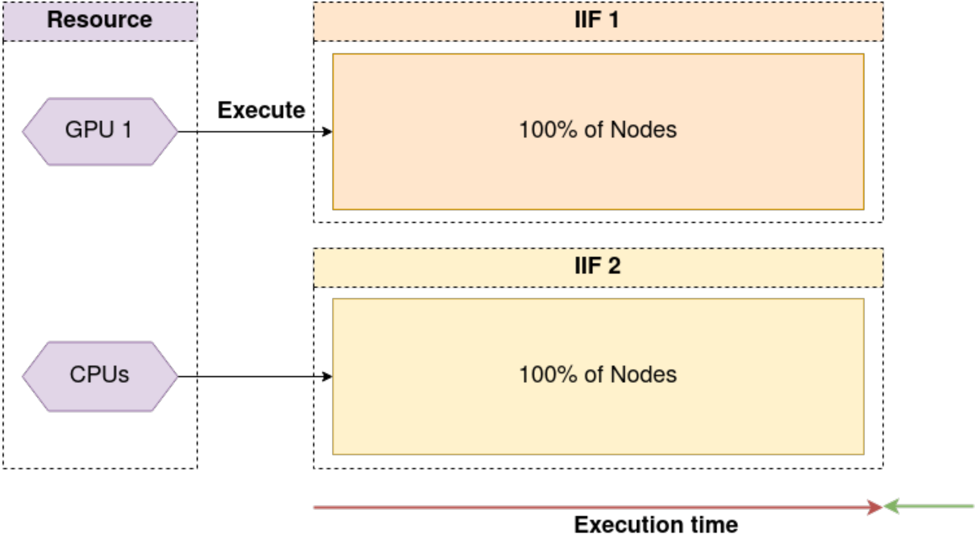
Problematic : Multi IIF



Problematic : Multi IIF



Problematic : Multi IIF



About Bench

Important

In this presentation, we focus on the ionic models computation

- The implementations presented here focus on bench
- We plan to extend some of those implementations to the rest of openCARP

Bench

- Simpler than the full openCARP simulator
- Only uses one IIF/region
  - No opportunity for scheduling
  - We can still do load balancing

Task-Based programming with StarPU

StarPU

- Task-based programming library for hybrid architectures
- Handles low-level issues for task-based programs transparently
  - Task dependencies
  - Heterogeneous scheduling
  - Optimized data transfer

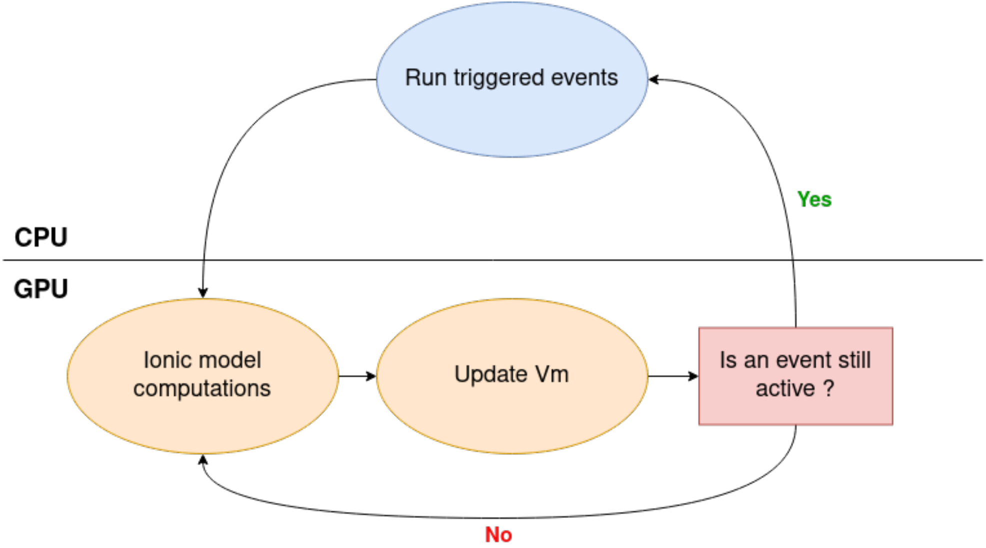
Granularity Issue

Issues

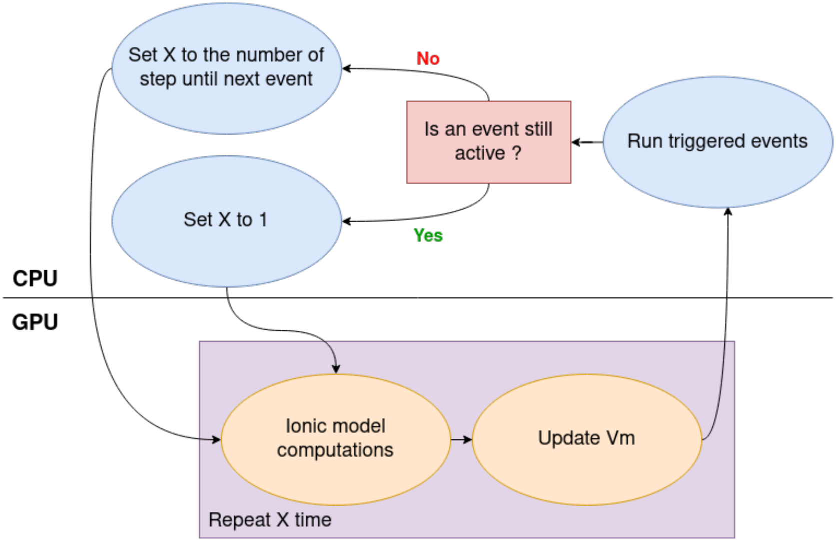
Creating large enough tasks is difficult

- Large amount of short timesteps
- One task per timestep $\rightarrow$ Granularity is too fine

Workflow of a time step during ionic model simulation



Workflow of a time step during ionic model simulation



About data movements

Data movements in the GPU version of bench

The generated code uses unified memory

- Causes page faults when data is unavailable
- Produce migration overhead with each page faults

Replacing unified memory with StarPU fixes those problems

```
==52977== Unified Memory profiling result:
Device "Tesla V100-PCIE-16GB (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
   31567  41.359KB  4.0000KB  2.0000MB  1.245117GB  178.6918ms  Host To Device
    8442  152.38KB  4.0000KB  0.9961MB  1.226807GB  111.2176ms  Device To Host
    2452         -         -         -           -   1.153306s  Gpu page fault groups
Total CPU Page faults: 4306
```

Figure: Unified memory profiling on AlievPanfilov (./bin/bench -I AlievPanfilov -a 100 -n 819200 –numstim=0) showing **1.15s** of page fault handling on a **6.22s** execution

About benchmarks

Architecture

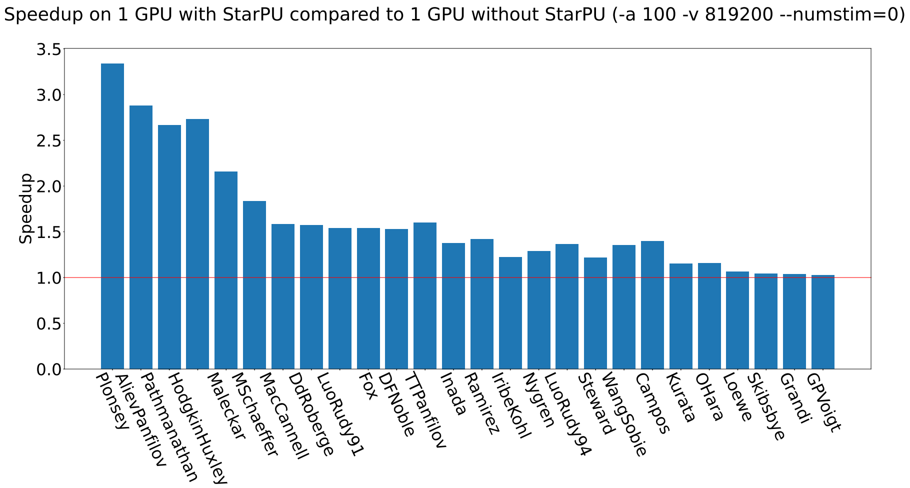
All experiments are run on plafrim's sirocco 14-16

- 2x 16-core Intel Skylake
- 2 NVIDIA V100
- 384 GB Memory

Experiments

- All experiments are run on bench
- All experiments are run with **no stimuli**
  - This allows us to measure just the ionic model
  - We still keep the data output on the terminal every 100 timesteps
- Models are sorted by makespan on the default setting
- Plugins models are excluded

Results (1 GPU)



Multi-GPU

Multi-GPU version

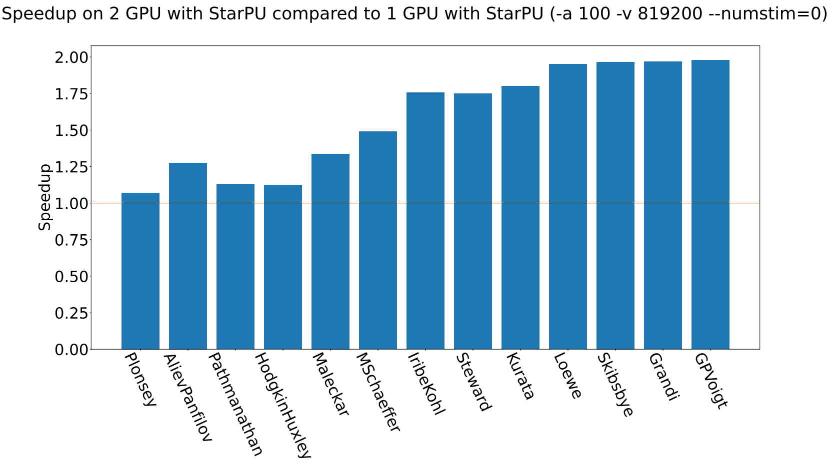
A multi-GPU version of bench is also available :

- Available on the starpu_multigpu branch
- Works with 13/48 of the models
  - Unfinished for models with LUT
  - Don't work on models using rosenbrock

About benchmarks

We use the task-based GPU version to compute speedup for the Multi-GPU version

Multi-GPU performances (strong scaling)



CPU + GPU

Hybrid version

- Old version available in branch starpu_task
- Performance issue
  - Still handle time steps the old ways
  - Even worse granularity issue
- Not clean or maintainable
  - Ionic models could not be compiled in both CPU and GPU at the same time
  - Current solution compiles the whole simulator two times

Section 2

LIMPET: Refactoring and multi-versions

About LIMPET

- Library that contains the ionic models
- Python scripts generate C/C++/MLIR from EasyML
- The model functions are then called by the simulator (mainly `compute`)

Before - LIMPET code structure

struct ion_if

- ionic model data
  - state variables
  - lookup tables
  - ...

struct ion_type

- basic info about the model *type*
  - name of the model
  - external data used or modified by the plugin
- the "interface" for ionic models (functions pointers)
  - `void (*compute)(int, int, ION_IF *, GlobalData_t **)`
  - `void (*initialize_sv)(ION_IF *, GlobalData_t **)`

Before - How does this work ?

→ a lot of generated code !

perl and python scripts generate code for allocating `ion_type` structures and assigning each function for each ionic model:

```
types[ii].trace = NULL;
types[ii].tune = tune_IMP_model;
types[ii].print_params = print_IMP_parameters_model;
types[ii].read_svs = read_IIF_svs_model;
types[ii].write_svs = write_IIF_svs_model;
types[ii].get_sv_offset = get_sv_offset_model;
types[ii].get_sv_list = get_sv_list_model;
types[ii].get_sv_type = get_sv_type_model;
```

Before - Model generation and memory allocations

- One `compute` function generated per model → the target (CPU / GPU) is chosen at compile-time
- The location of memory (GPU or main RAM) is also chosen at compile-time for the whole limpet library (every allocation goes through a macro)

Before - Issues

Code structure

- Out of place C code in the C++ codebase
- Lots of generated code
- `void` pointers everywhere and casts all over the place

Before - Issues

Code structure

- Out of place C code in the C++ codebase
- Lots of generated code
- `void` pointers everywhere and casts all over the place

Memory allocations

- All generated models will allocate memory on GPU if 1 GPU model is generated, even if they run on CPU!
- This is not flexible, especially in an heterogeneous environment

Before - Issues

Code structure

- Out of place C code in the C++ codebase
- Lots of generated code
- `void` pointers everywhere and casts all over the place

Memory allocations

- All generated models will allocate memory on GPU if 1 GPU model is generated, even if they run on CPU!
- This is not flexible, especially in an heterogeneous environment

Model generation

- Only one implementation per model: StarPU needs multiple implementations!

After - Rewriting

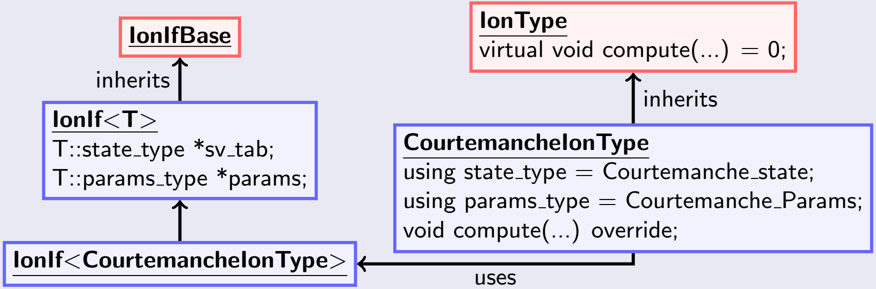
Code structure

- The old code is basically inheritance written in C
- We leverage C++ features to reduce the need for generated code
- By using C++ tools, more errors can be detected at compile-time

After - Rewriting

Code structure

- The old code is basically inheritance written in C
- We leverage C++ features to reduce the need for generated code
- By using C++ tools, more errors can be detected at compile-time



After - Rewriting

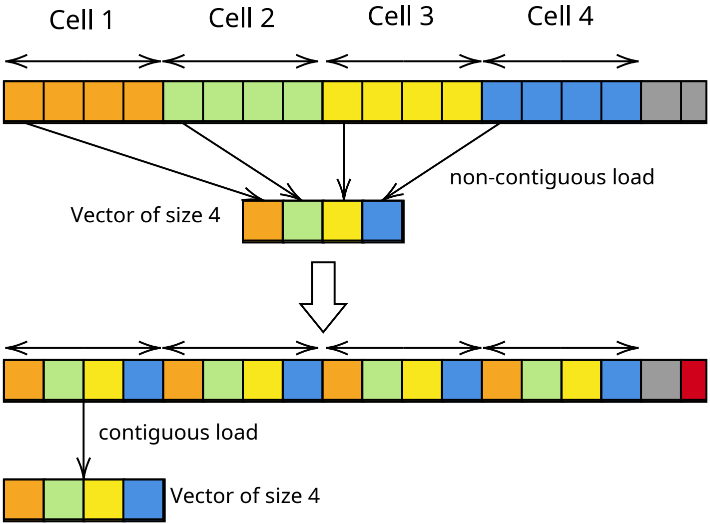
Code generation

- The compute function of each model is generated for all possible targets
- `IonIf` knows on which target it's running and calls the compute function accordingly
- No more memory allocations in generated code

Memory allocations

- Dynamic memory allocation functions depending on the target (no more macros)
- `IonIf` allocates and frees all memory in its constructor / destructor
- `IonIf` makes the correct memory allocations according to the target

Other useful things - Data Layout Optimization



- Can now be used on GPU
- Allows for easily switching during execution
- A single data structure for simpler code

Section 3

Conclusion

We presented:

- A Multi-GPU version of LIMPET
- Automatic data transfer handling with StarPU
- Improved timestep loop for granularity

- A refactoring of the LIMPET library
- Generation of multiple implementations
- Dynamic memory allocations for ionic models

Added bench option

New bench option → `--target`:

- Allows to chose the target for the execution of bench (`cpu`, `mlir-cpu`, `mlir-cuda`, `mlir-rocm`
- Can chose a target automatically with `auto` (default configuration)
  - `auto` chooses the most "advanced" target available for the given model ((`mlir-rocm` & `mlir-cuda`) > `mlir-cpu` > `cpu`)
- Already merged in the master branch

What's next?

- A hybrid version of LIMPET
- Heterogeneous versions of openCARP
- Task scheduling/Load Balancing for openCARP

- Integrate and test with the full openCARP simulator
- Memory management through StarPU handles

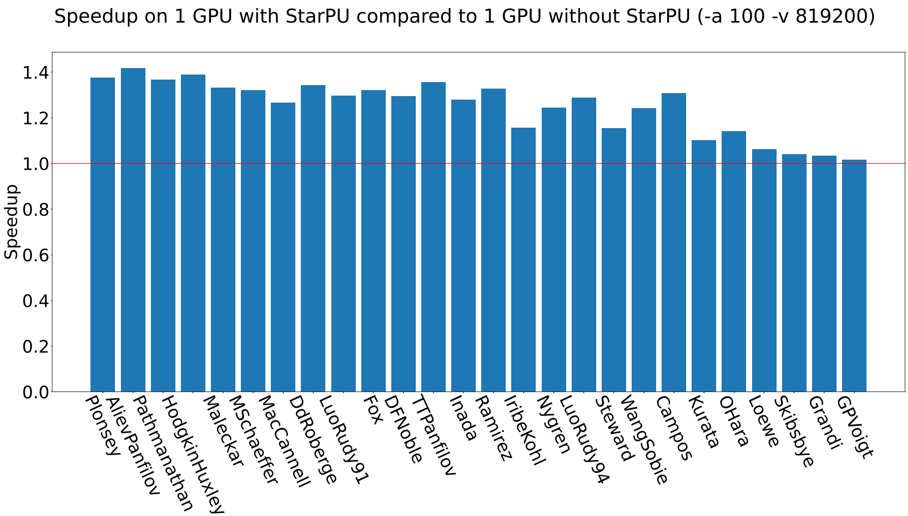
Thank you!

Thank you for your attention
Any questions?

Section 4

Annexe

Results (1 GPU) on default setting



Multi-GPU performances (strong scaling) on default setting