CUDA and ROCm GPU Ionic Models: Raw Performance and Energy Performance

Arun Thangamani, Tiago Trevisan Jost, Raphaël Colin,
Vincent Loechner, Stéphane Genaud, and Bérenger Bramas

thangamani@unistra.fr
University of Strasbourg and Inria, France

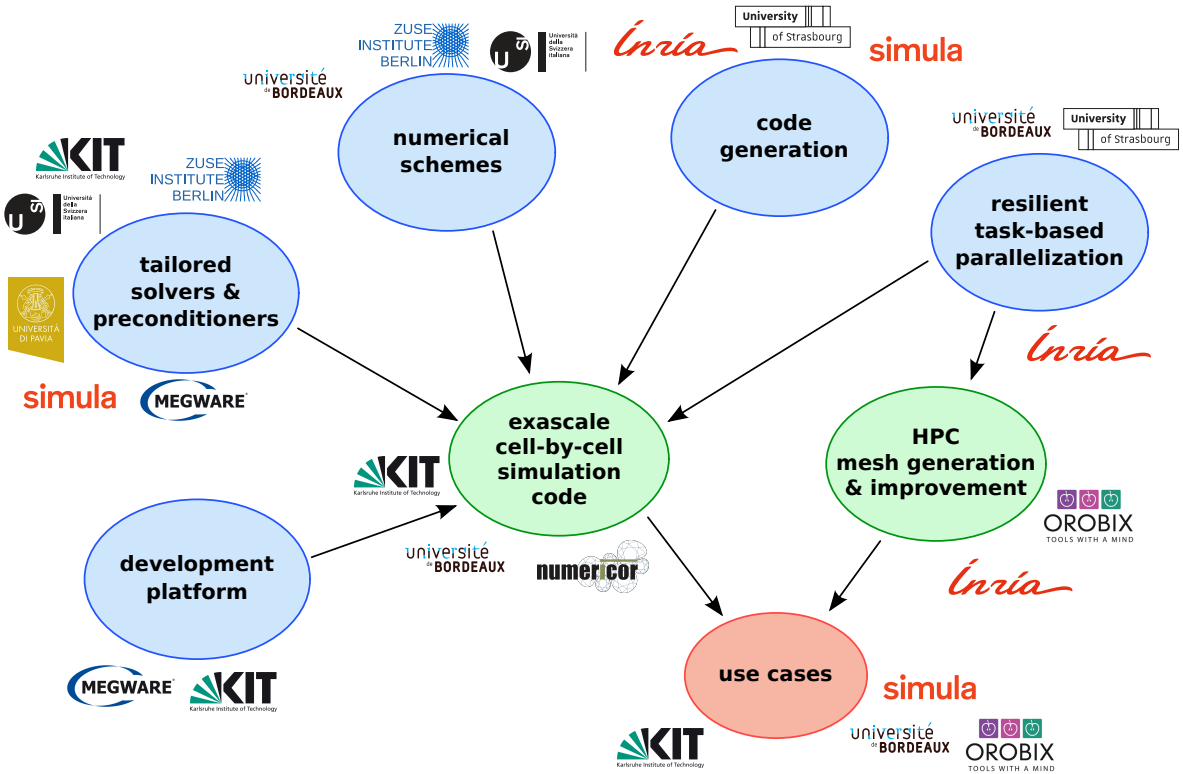
MICROCARD European Project, Work Package 6

3rd MICROCARD Workshop, July 2023

EuroHPC Joint Undertaking

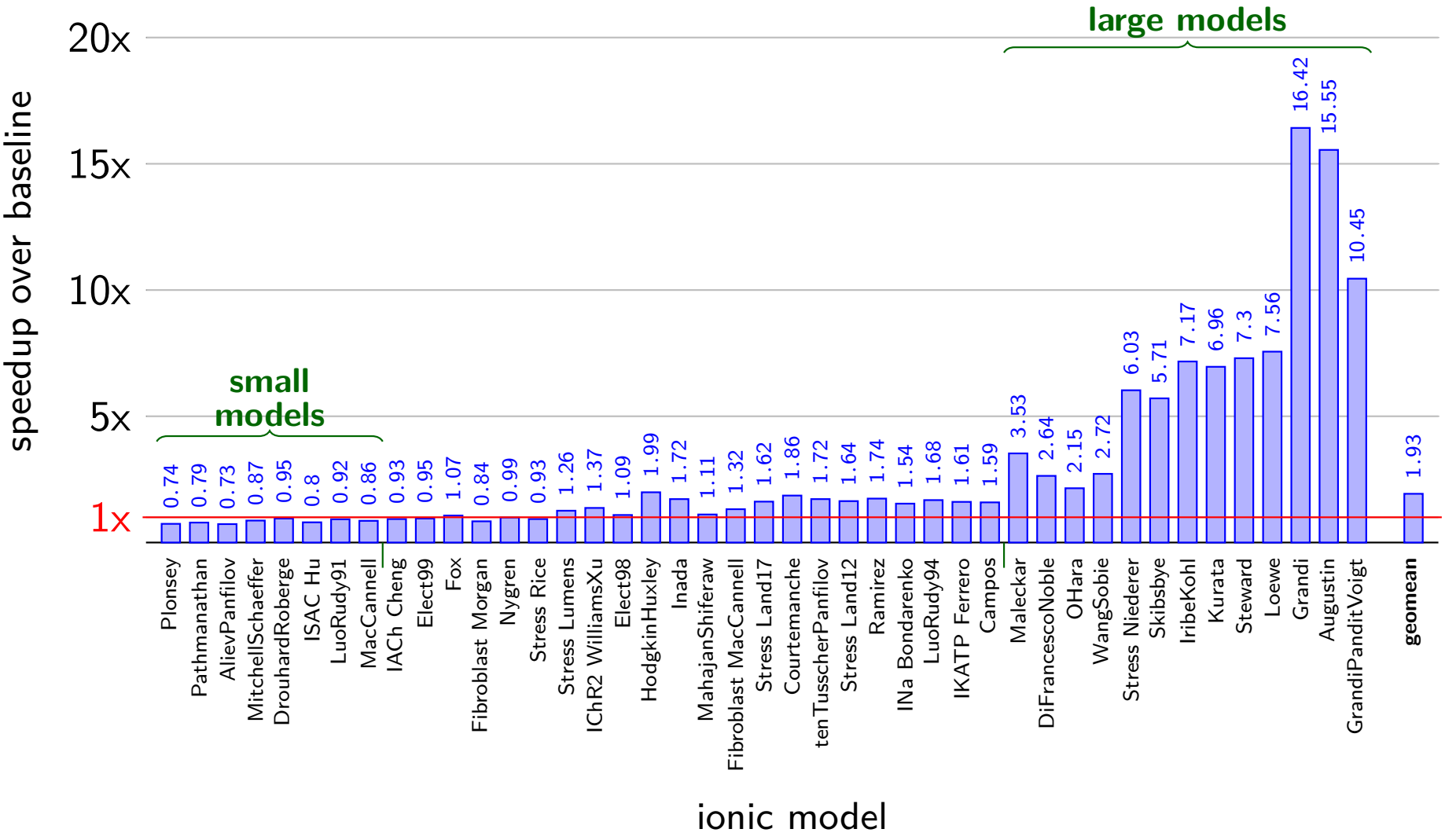
The project leading to this application has received funding from the European High-Performance Computing Joint Undertaking EuroHPC (JU) under grant agreement No 955495. The JU receives support from the European Union's Horizon 2020 research and innovation programme and from France, Italy, Germany, Austria, Norway, Switzerland.

WP6 - Code Generation for Ionic Models



Outline

Performance: AVX-512; **32 Cores**; Cells - 8192



Why MLIR?

- parallelization: OpenMP is easy (and done) the loop is parallel
- loop optimization: standard techniques, polyhedral and many others
- vectorization:
  - $\rightarrow$ need to do it manually or at the compiler level

Why MLIR?

- parallelization: OpenMP is easy (and done) the loop is parallel
- loop optimization: standard techniques, polyhedral and many others
- vectorization:
  - → need to do it manually or at the compiler level
- GPUization:
  - can manually write CUDA, ROCm and *etc...*
  - → or can do it at the compiler level

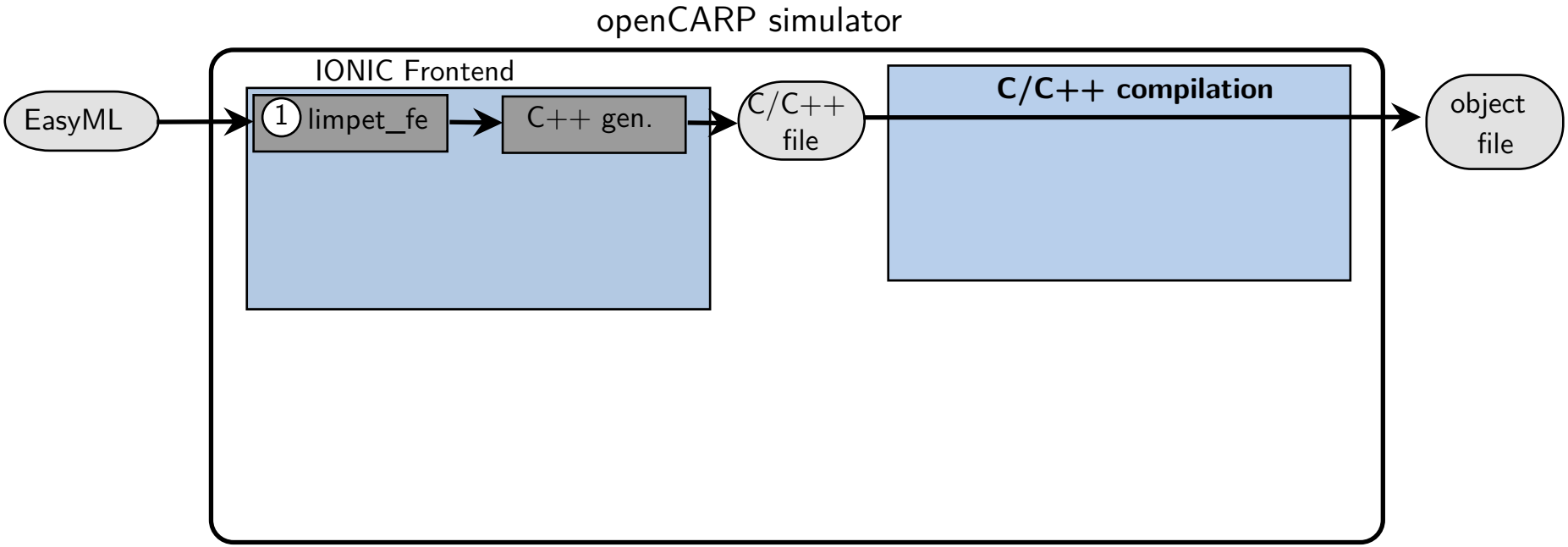
Why MLIR?

- parallelization: OpenMP is easy (and done) the loop is parallel
- loop optimization: standard techniques, polyhedral and many others
- vectorization:
  → need to do it manually or at the compiler level
- GPUization:
  - can manually write CUDA, ROCm and *etc...*
  → or can do it at the compiler level

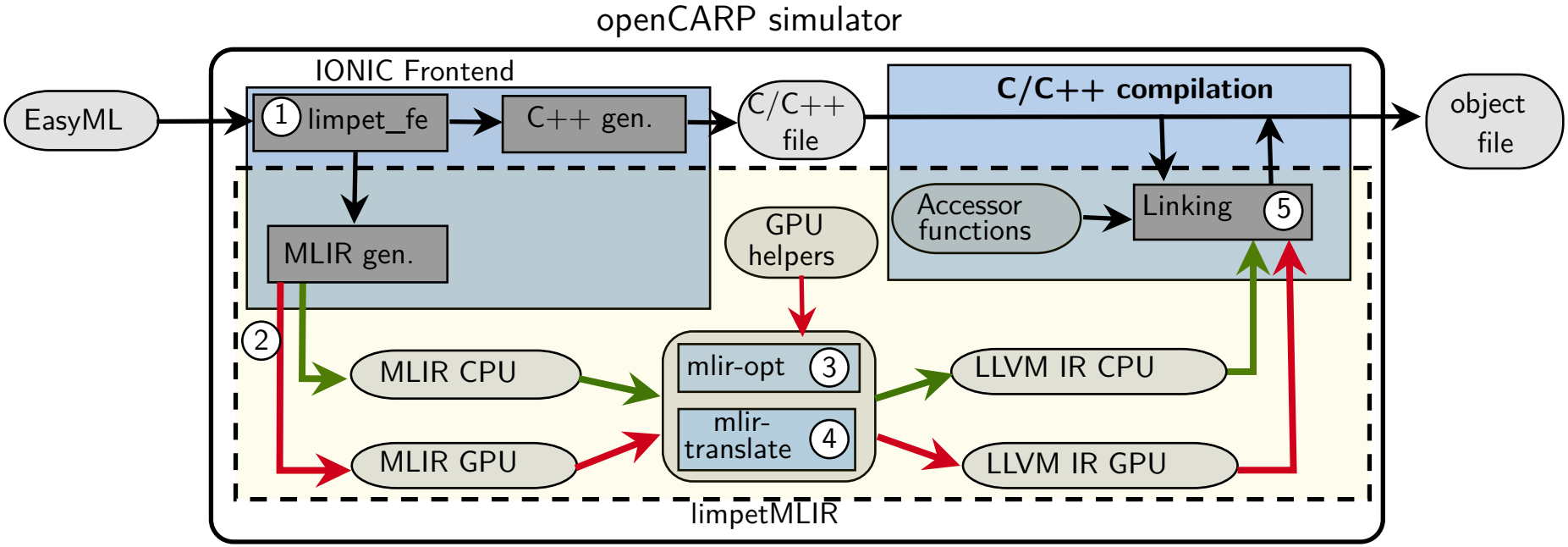
→ using an existing compiler infrastructure - **MLIR**
to benefit from generality, other optimizations
and automatic code generation passes

Outline

openCARP - Compilation Flow

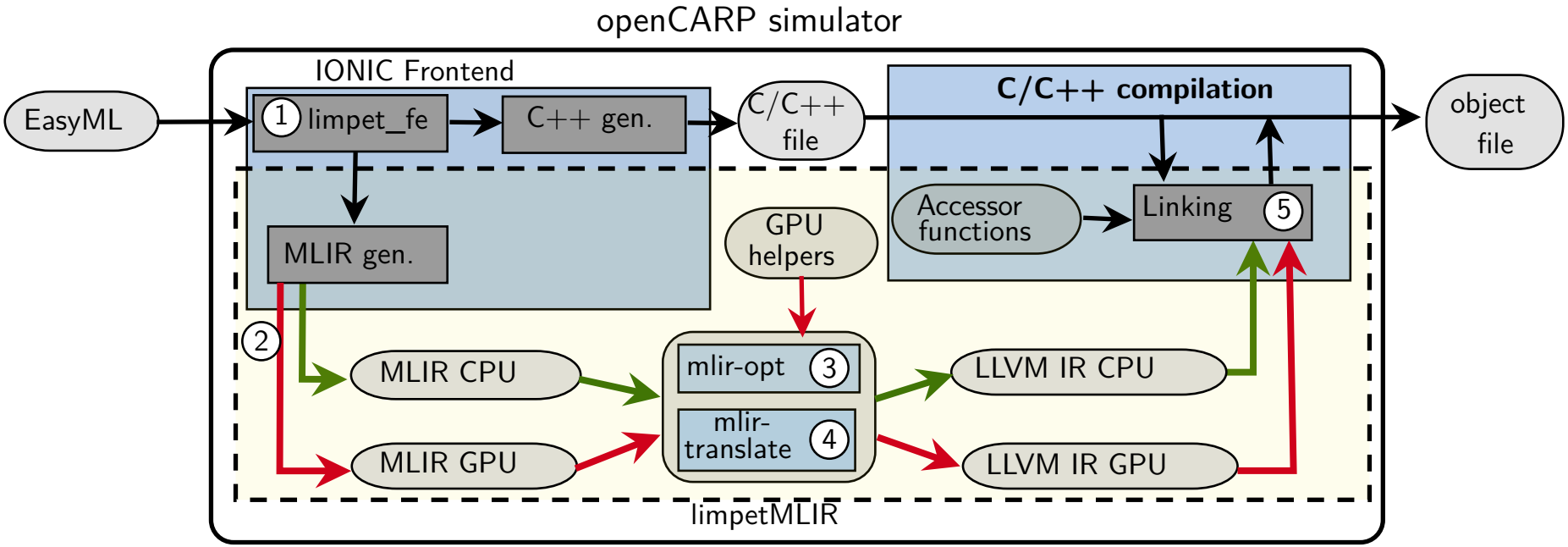


limpetMLIR: Our Proposed Compilation Flow



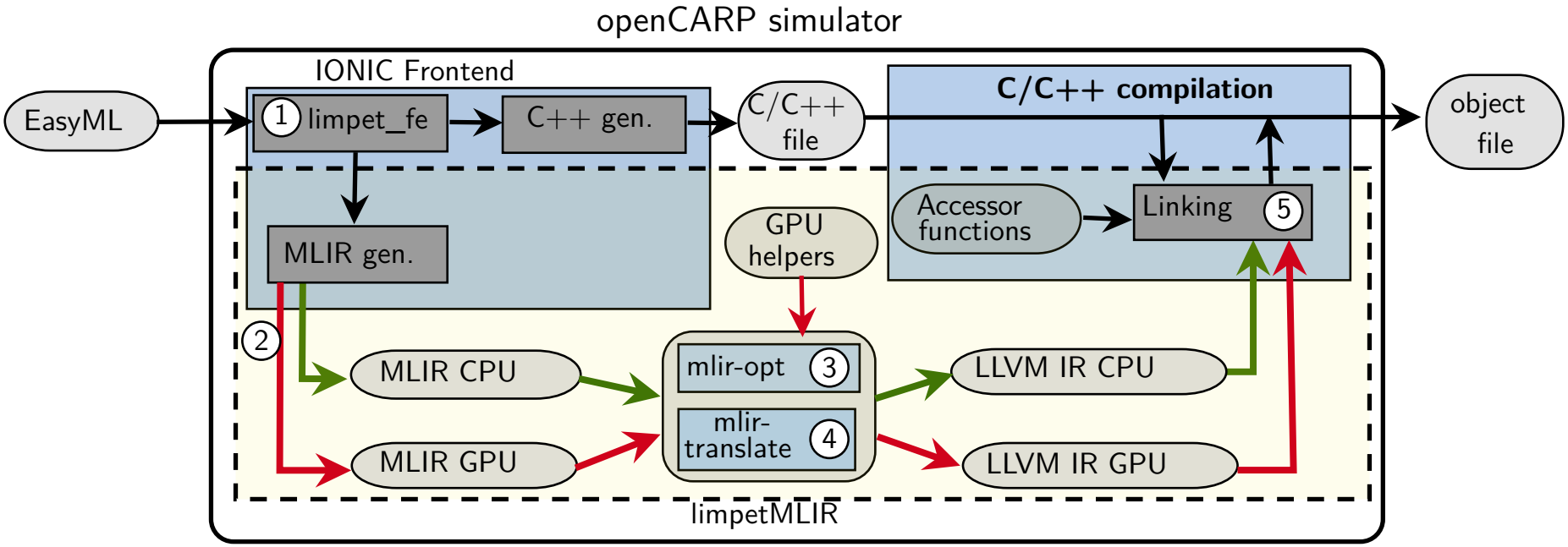
1 - From EasyML DSL the `limpet_fe` python script generates an AST, which serves as a common entry point for baseline openCARP and limpetMLIR.

limpetMLIR: Our Proposed Compilation Flow



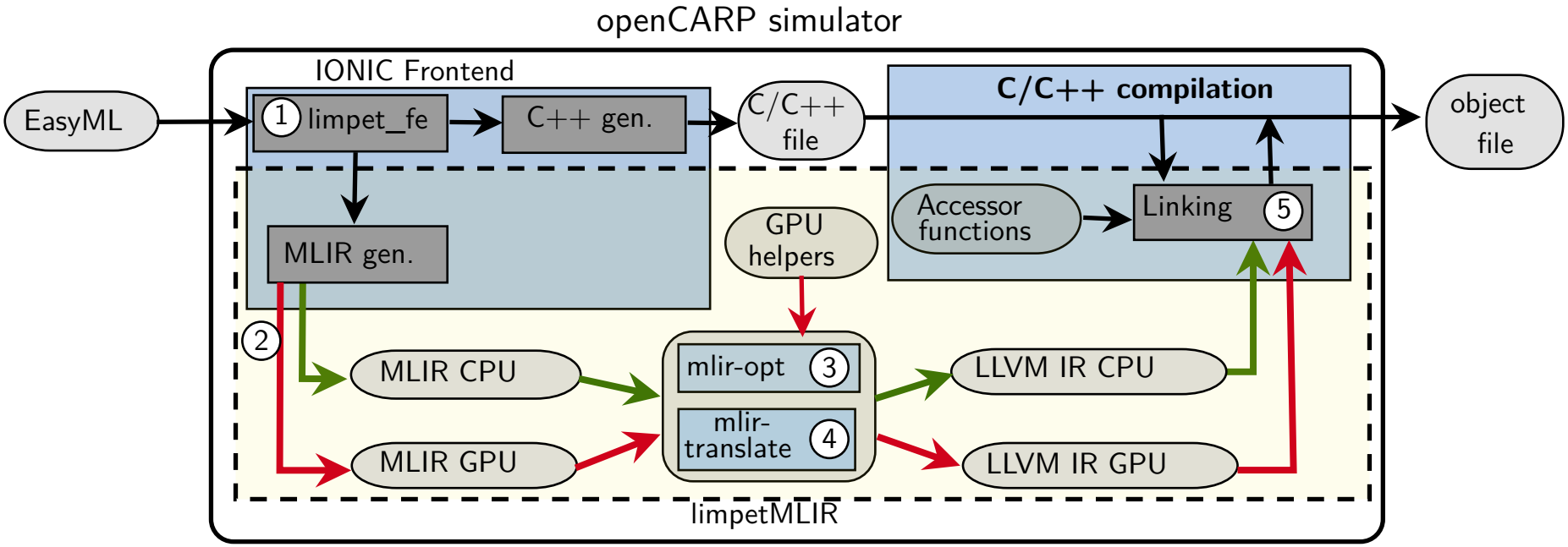
2 - Using python bindings, our limpetMLIR emits MLIR code for compute intensive kernel using `scf`, `arith`, `math` and `memref` dialects.
- Choose type `vector<?xf64>` for vectorized CPU; and `f64` for GPU.

limpetMLIR: Our Proposed Compilation Flow



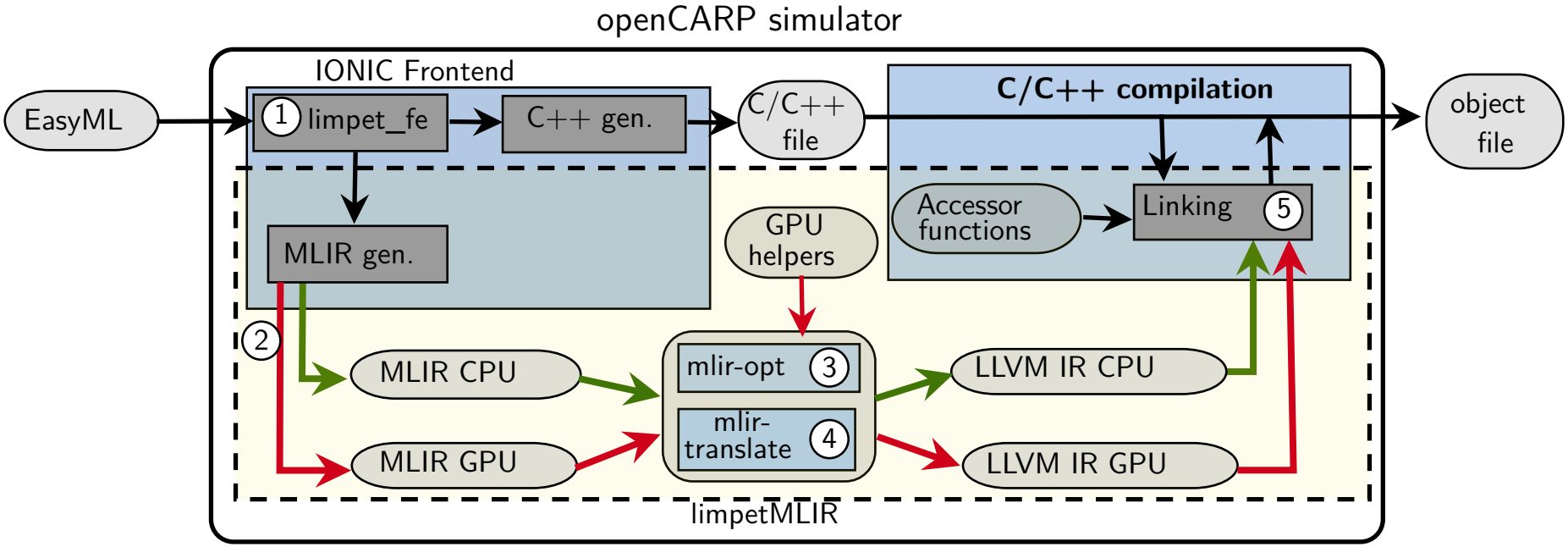
3 - MLIR lowering pass. For GPU the MLIR code can be lowered to either `nvvm` (the Nvidia CUDA IR) or `rocdl` (the AMD ROCm IR).

limpetMLIR: Our Proposed Compilation Flow



4 - MLIR translator pass converts the final MLIR code to LLVM IR (either CPU or GPU).

limpetMLIR: Our Proposed Compilation Flow



5 - Last is the linking phase, where c/c++ and LLVM IR (either CPU or GPU) are linked together into an object file.

Code generation

- Methods for temporal discretization - Euler, Runge-kutta 2 and 4, Rush-Iarsen, Sundnes, Markov, and Rosenbrock
- *GPU helper* functions $\Rightarrow$ for LUT and Rosenbrock external function calls.
- Memory management for GPUs
  - Unified memory model for CUDA (Nvidia),
  - Manual memory allocations for ROCm (AMD),
  - `/bin/bench` executable.

Outline

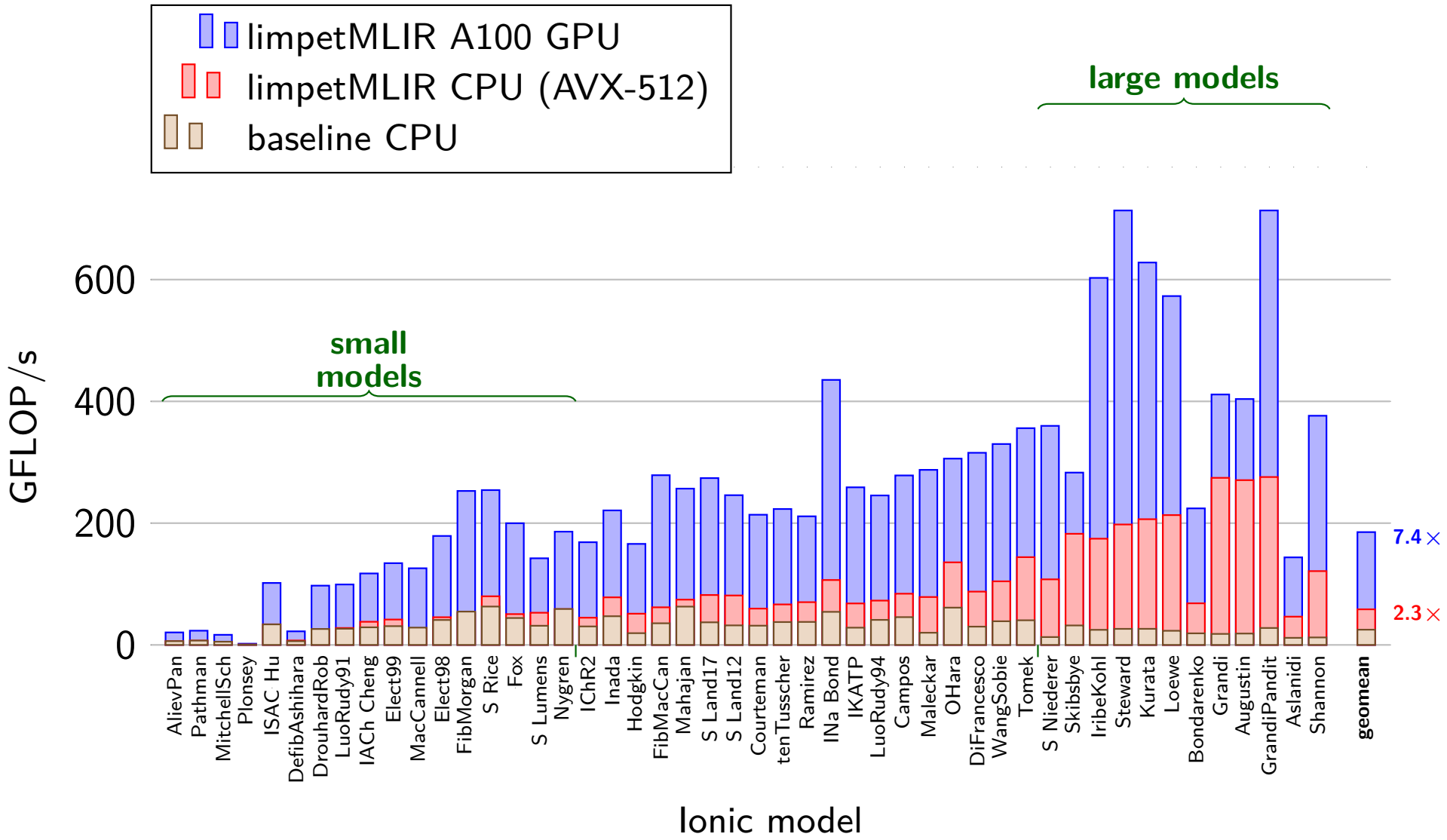
Results

- limpetMLIR implemented on top of openCARP (git master branch) using the LLVM infrastructure from trunk (version 15.0.x)
- try it with the following cmake options available at: `git.opencarp.org/openCARP/openCARP/-/blob/master/docs/BUILD_WITH_MLIR.md`
- or with the following docker file + image available at: `https://seafile.unistra.fr/f/608712e4bee94ea2a709/?dl=1`

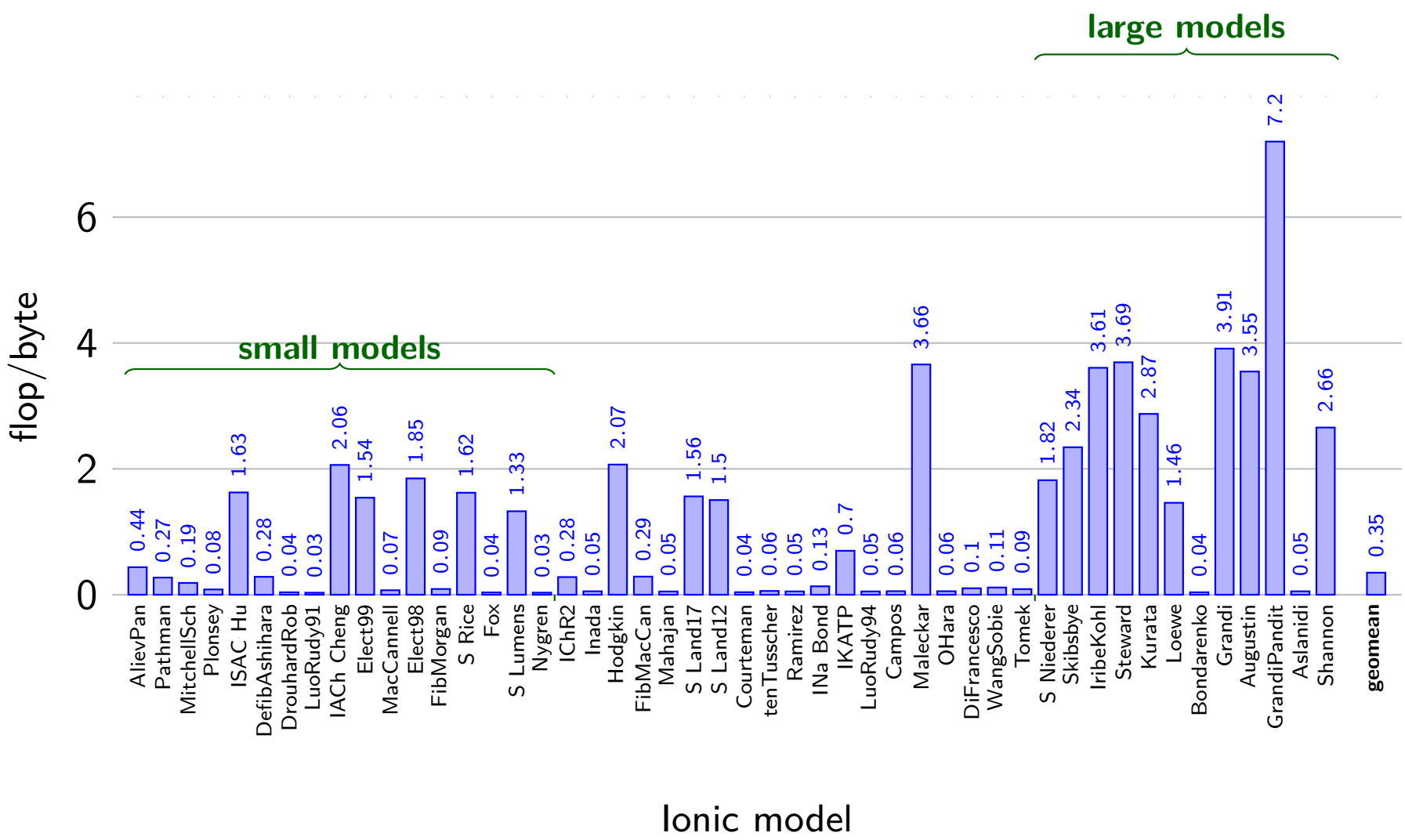
Performance evaluation

- Evaluated with 48 ionic models available in openCARP
- benchmark running the ionic model alone (`bin/bench`);
  - cells = 819,200
  - timestamp = 100
  - step = 0.01 (10,000 computation steps)
- CPU: a 2x 18-core Cascade Lake Intel Xeon Gold 6240.
- GPU:
  - an A100 Nvidia GPU (9,700 GFLOP/s peak perf. on `double`)
  - an MI50 AMD GPU (6,600 GFLOP/s peak perf. on `double`)

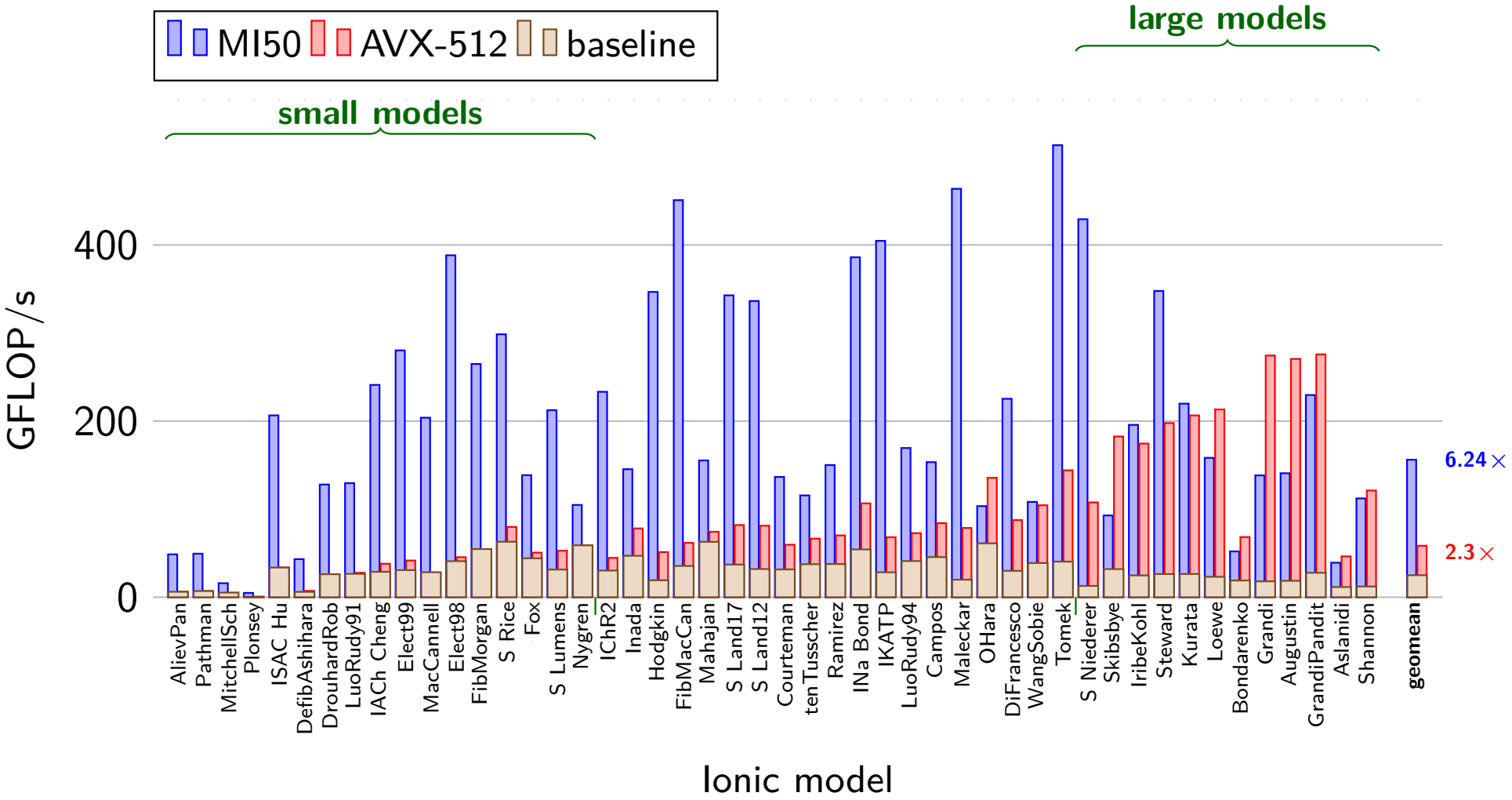
GFLOP/s performance - Nvidia A100



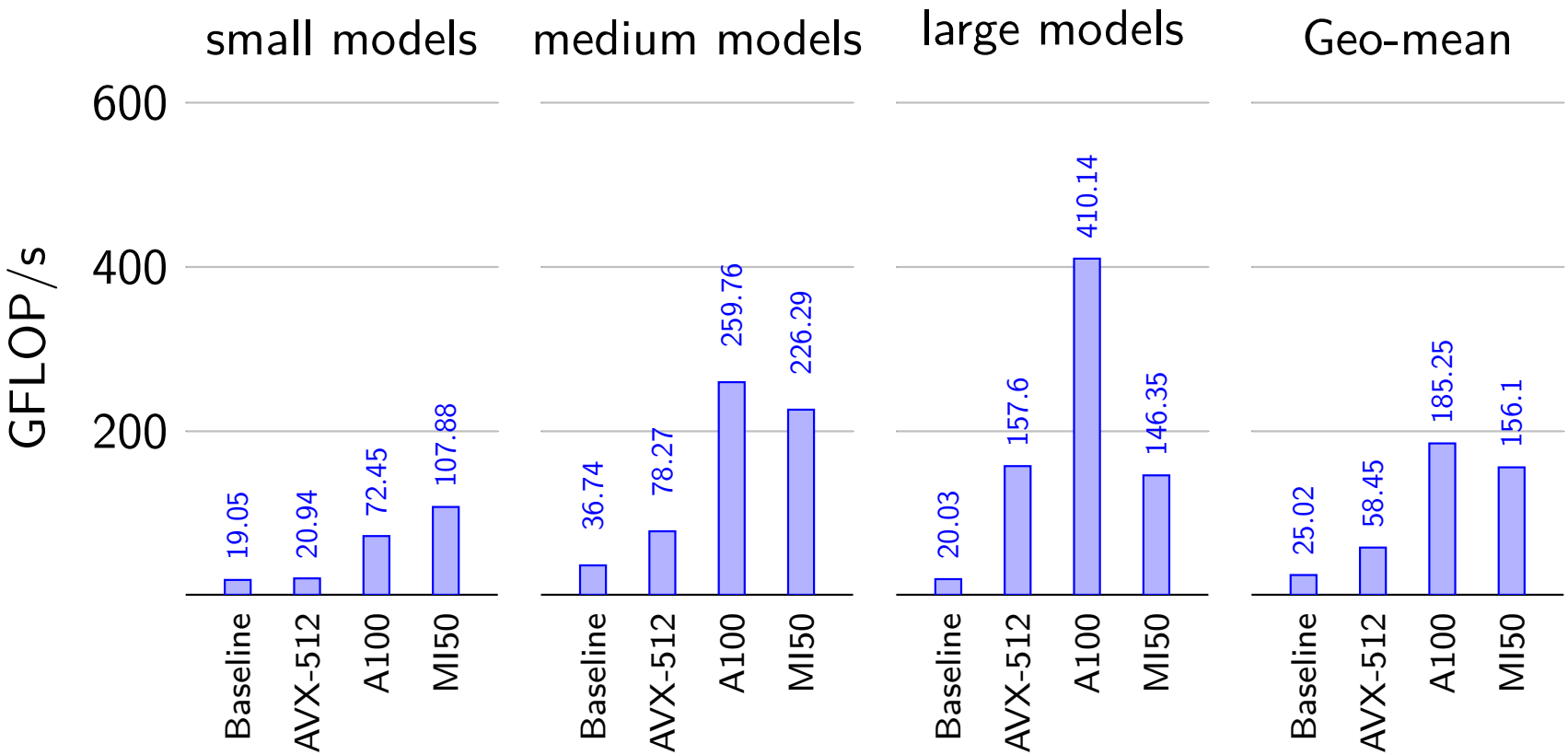
Compute-intensity of ionic models



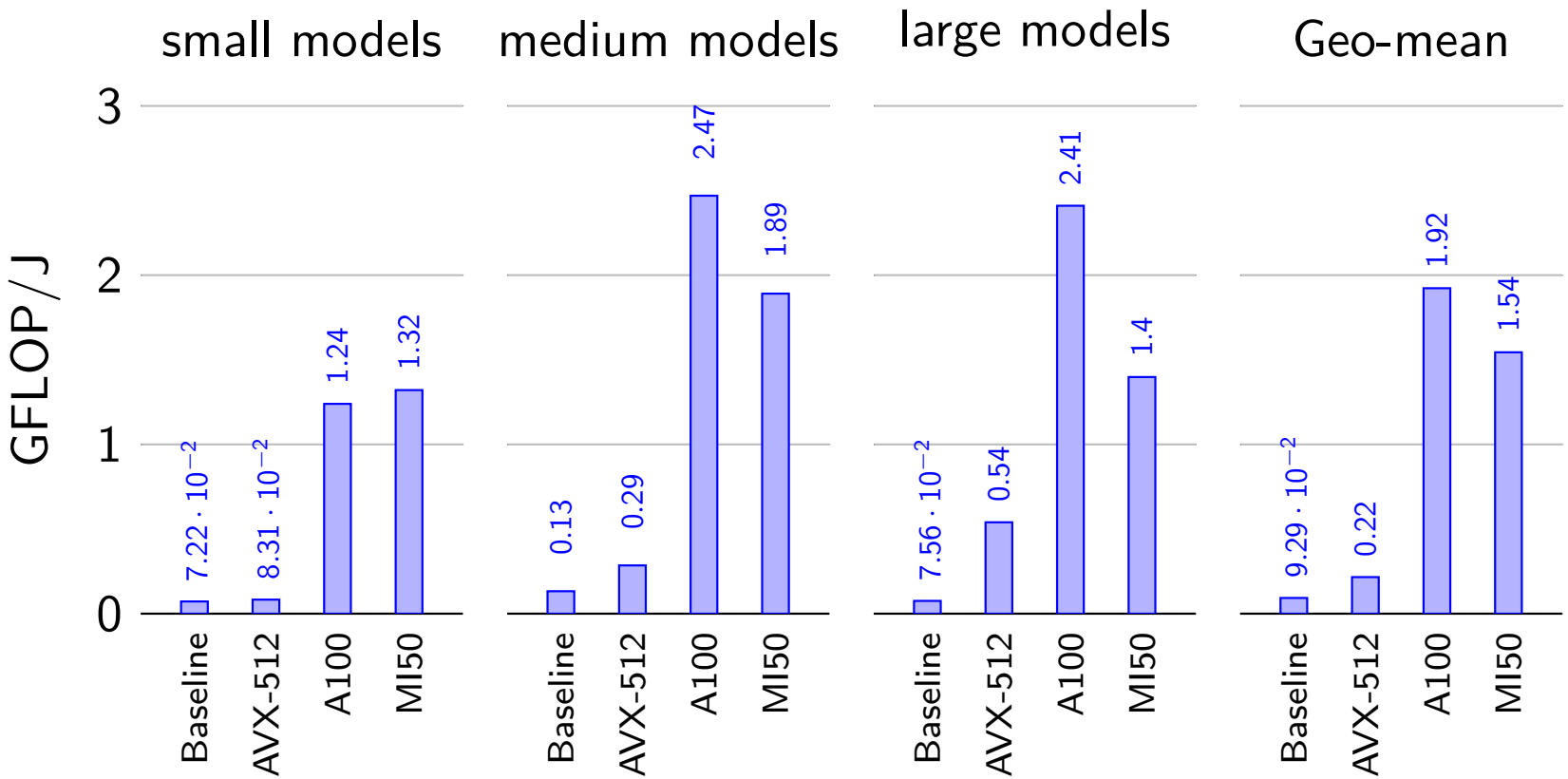
GFLOP/s performance - AMD MI50



GFLOPS/s comparision



GFLOP/J energy efficiency comparision



Outline

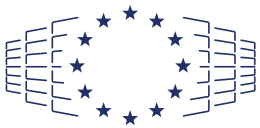
Conclusion

- ✓ GPU code generation (CUDA & ROCm)
- very good performance
  - A100: 7.40× vs baseline and 3.17× vs AVX-512
  - MI50: 6.24× vs baseline and 2.67× vs AVX-512
- very good energy efficiency
  - A100: 20.67× vs baseline and 8.73× vs AVX-512
  - MI50: 16.58× vs baseline and 7× vs AVX-512

Future interest of work:

- integrate with the solver to avoid memory transfers
- further optimization of the generated code
  (LUTs as splines, NN, Rosenbrock, FP approximations, ...)

Thank you!



EuroHPC
Joint Undertaking

University
of Strasbourg

Inria
INVENTEURS DU MONDE NUMÉRIQUE

The project leading to this application has received funding from the European High-Performance Computing Joint Undertaking EuroHPC (JU) under grant agreement No 955495. The JU receives support from the European Union's Horizon 2020 research and innovation programme and from France, Italy, Germany, Austria, Norway, Switzerland.