LIGHTWEIGHT EMBEDDED DSLS FOR GEOSCIENTIFIC MODELS.

May 25, 2022 | Zbigniew Piotrowski | Forschungszentrum Jülich (currently ECMWF)

Computational challenge for legacy codes

Established legacy codes, often employing coding and parallelism paradigms from the previous decade, **are usually not able to properly exploit efficient hardware chips of modern supercomputers**. Available FLOPs are used only in several percent, and peak memory bandwidth is not always saturated. Common reasons for this include:

- inability to execute on GPUs,
- lack of or simplistic intranode (e.g. shared) memory parallelization,
- too frequent MPI communication,
- lack of overlapping MPI communication with computation,
- poor performance of implementation of modern/complex Fortran and MPI constructs.

There is not a single easy solution (otherwise it would already be done), and competing programming concepts are continuously developed to support the effort.

Mitigation strategies and their adverse consequences

1. Use auto-parallelization features of modern compilers.
   - Does not seem to work for non-trivial codes.
2. Complete code rewrite using performance portability frameworks, e.g. Kokkos, Raja, OpenCL, perhaps Julia.
   - Time and resource consuming, may impair domain scientist productivity, suboptimal performance.
3. Introduction of OpenACC and OpenMP directives.
   - Obfuscated code, directives support (and bugs) vary between compilers, some specialized code still needed, suboptimal performance.
4. Use libraries optimized for a particular architecture.
   - Typically addresses only a fraction of numerical formulation.
5. Mutiple code ports targeting different architectures, e.g. using CUDA or HIP next to the legacy CPU code,
   - Lots of code duplication, difficult to maintain.
6. Use full Domain Specific Language, e.g. GridTools.
   - Requires language rewrite, may suffer from long compilation times due to lots of work delegated to the (e.g. C++) compiler, readability is questionable, strong dependence on the external project.

A couple of remarks on the ESM model development

Regardless of the strategy for performance portability, legacy geophysical codes require substantial developments already at the Fortran/C level.

- **Code modularization** is preferred to a monolithic construction, as it facilitates porting to new supercomputers, debugging and community development and teaching.
- Code **analysis** towards increasing memory locality, avoiding intermediate memory stores and computational intensity needs to be performed **by a human** rather than (or at least: before) delegated to the external software solutions.
- **Computations in halo** to replace the MPI communication are often not easy to implement, as the **special stencils at the boundaries** (especially for regional models) come between the major computations.
- Prioritization of the computations at the **MPI subdomain boundaries** enables overlapping the computations and communication, but its direct encoding **obfuscates the code**.

Most often, iteration over the domain points is just the technical implementation of the abstract ”computational grid” concept. Therefore it seems justified to **replace the loop sets with** abstract names representing **topological characteristics** of the computation.

What if neither the large software engineering teams are available, nor the domain scientists are ready for the paradigm shift ?

- The overarching goal here is to develop a minimal, lightweight strategy to extend code and performance portability of legacy Fortran/C codes across modern HPC architectures.
- This aims at the optimal productivity of the domain scientists, while offering reasonable (but suboptimal) computational performance on GPUs, and potential lightweight strategy to port to different emerging architectures.
- The coding would benefit from the legacy code development ways on the CPU machines, including full debug capabilities, while allowing for gradual GPU implementation.

Proposed solution: **intervene at the level of loops containing heavy stencil computations and abstract them, instead of populating the code with several sets of compiler directives.**

Tenets of lightweight eDSL

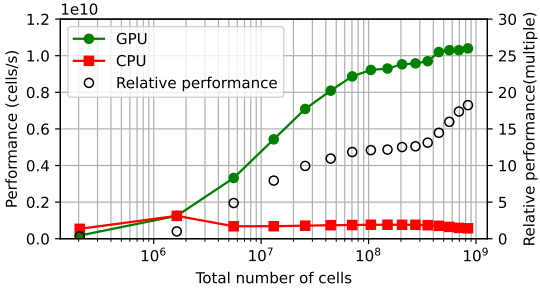
Main design concepts:

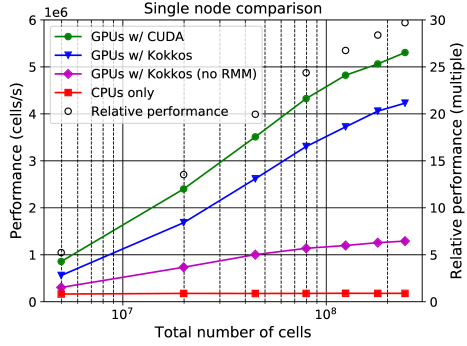
- Set of Fortran loops around kernels are replaced by preprocessor macros named after the scope and grid of the operation (e.g. AgridXYZFullDomain). In turn, C implementation benefits from the lambda function syntax to pass on the stencil computation to the kernel.
- Definition of memory allocation is abstracted to accommodate accelerator-specific attributes, e.g. MANAGED or DEVICE (Fortran case) or special memory pool allocator is implemented (C case).
- Due to Fortran semantic constraints, a couple of search/replace operations, e.g. using sed, are also needed.
- Preprocessing and build processed is controlled externally, e.g. via CMake. Current backend uses pure MPI CPU or MPI + CUDA Fortran directives. For C-based codes, Kokkos backend (and thus extended performance portability) is easily achievable thanks to lambda function concept.

Testing framework

- Standard experiment reproducing 3D passive tracer (sphere) rotation in box.
- The advection is performed using a fully three-dimensional, nonlinear iterated upwind scheme called MPDATA, used operationally in the latest dynamical core of COSMO NWP framework. MPDATA, in a form of a dwarf formed originally for the ESCAPE project is now cast in the eDSL form and executed on CPU+MPI and GPU+MPI scenarios.
- Reference results with integration accuracy (error norm after a full signal revolution in the domain) are known, so consistency between CPU and GPU implementation can be easily assessed.
- This test typically employs the same number of gridpoints in each three dimensions, standard for atmospheric DNS studies, but different from mesoscale applications.
- Reference result exists for the computational grid size of 59^3. For strong scalability, we test 7-,14- and 21-fold grid refinement, i.e. 413^3, 826^3 and 1239^3 grids, respectively.

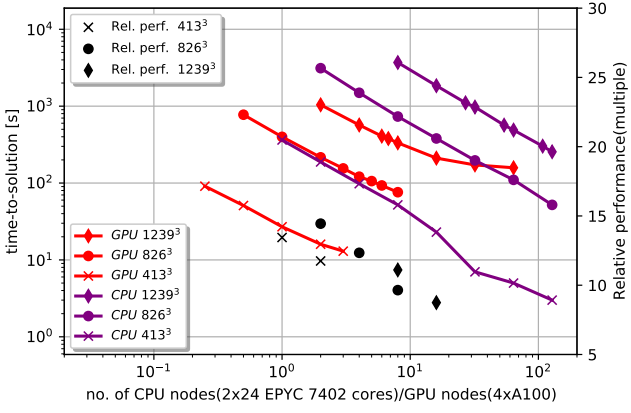
Single node eDSL performance - Fortran and C codes





- Both Fortran and C codes achieve major, 20-30 fold speedup. However, typical speedup when using GPU over CPU in practical stencil applications is expected to be closer to the memory bandwidth ratio, e.g. 3-4 times.

Strong scalability - Fortran passive tracer advection



- In a strong scaling scenario (getting the answer as quickly as possible by using the largest feasible number of computing resources), GPUs clearly outperform CPUs again, but it is difficult to get good performance at the higher GPU counts, probably because to little work to saturate each GPU.

Take home message

- The minimally invasive way to achieve performance portability may be: to abstract contents of computationally-heavy loops, to automatically generate GPU kernels, e.g. using preprocessor macros.