LIGHTWEIGHT FORTRAN EDSL

As applied to 3D MPDATA nonlinear Eulerian advection scheme

March 16, 2022 | Zbigniew Piotrowski | JSC

Computational challenge for legacy codes

Established legacy codes, often employing coding and parallelism paradigms from the previous decade, **are usually not able to 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 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.

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 C++ rewrite, may suffer from long compilation times due to lots of work delegated to the 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 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 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 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.

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)
- Definition of memory allocation is abstracted to accommodate accelerator-specific attributes, e.g. MANAGED or DEVICE.
- In practice, the latter requires reasonable separation of the code modules as the variable types must match throughout the code at compile time (so not everything needs to be ported at once).
- 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 is easily achievable thanks to lambda function concept.

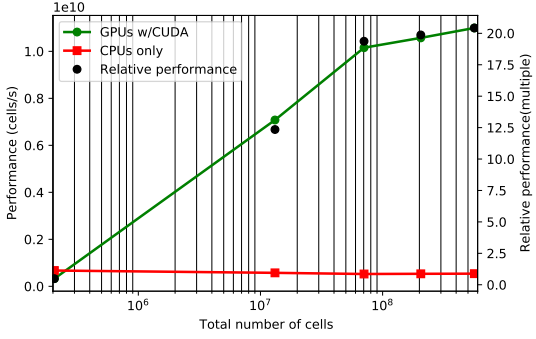
Challenges and lessons learned

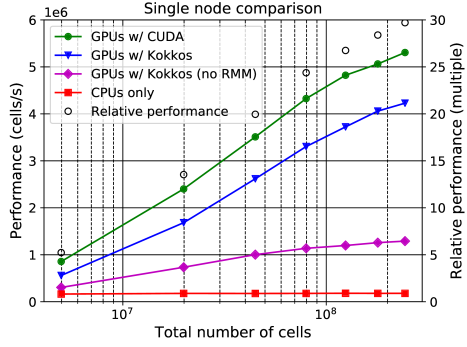
- The first attempt relied on the fully automatic GPU memory management via 'MANAGED' attribute.
- However, this resulted in unexpected Host - Device memory transfers that were difficult to control.
- A simple solution was to force the auxiliary variables to stay at the device using DEVICE attribute.
- A second non-trivial task was to adapt MPI layer to use the device. So far, only 1D MPI decomposition in the slowest-varying dimension is coded efficiently, so the continuous GPU memory buffer can be passed directly to the MPI call. 3D MPI decomposition requires additional tuning effort.
- CPU version can still use the 3D MPI memory decomposition.
- Most of the preparation, like increasing computing intensity or thoughtful implementation of boundary conditions is still on the pure Fortran size, as opposed to the full-blown DSL attempts like GridTools, where we hope that some work will be done by the library.

Testing framework

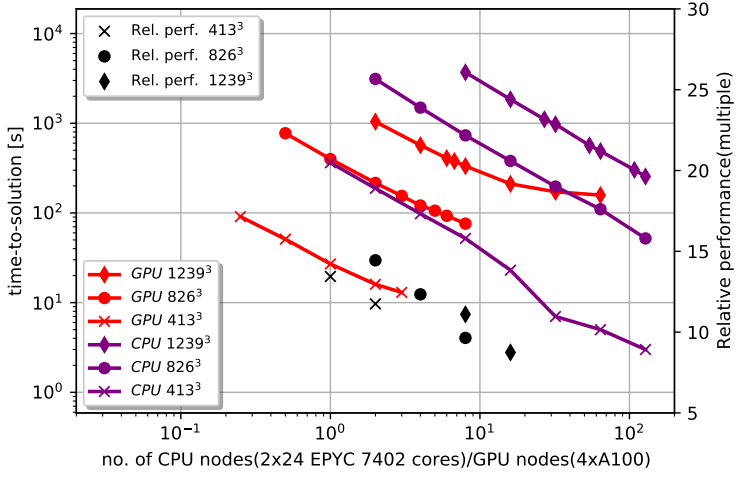
- Standard experiment reproducing 3D passive tracer (sphere) rotation in box.
- The advection is performed using a fully three-dimensional, nonlinear iterated upwind scheme called MPDATA, used operationally in the latest dynamical core of COSMO NWP framework. MPDATA, in a form of a dwarf formed originally 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





Strong scalability



JÜLICH Forschungszentrum

Conclusions

- Very reasonable speedup is achieved, similar to the original effort in JSC on the Parflow GPU implementation (a hydrological code written in C).
- Scalability on the very large number of GPU nodes is not great, work on optimizing MPI communication is needed to improve it.
- Code remains fully debuggable, overhead from the perspective of domain scientist is minimal.
- Future work should extend the eDSL implementation to full fluid solver. In this exercise, an iterative GCR Krylov solver would be ported.
- Likely, a memory pool would have to be implemented in Fortran to mitigate limited amount of the GPU memory.
- A couple of other backends seem straightforward, e.g. OpenMP, OpenACC.
- Loop abstraction provides clean possibility to computing border regions needed for halo on priority GPU stream.
- The whole effort on eDSL implementation with C and Fortran-based is now summarized in the draft of the eDSL publication, close to submission.