Review of "Comparing the Performance of Julia on CPUs versus GPUs and Julia-MPI versus Fortran-MPI: a case study with MPAS-Ocean (Version 7.1)" by Robert R. Strauss, Siddhartha Bishnu, and Mark R. Petersen

General comments

The manuscript presents a shallow water model, written in the Julia programming language for CPUs and GPUs, and compares its performance to an object-oriented Python code and an established Fortran code. Since Julia is now emerging in the field of geophysical model development, and its application to unstructured-mesh PDE solvers is novel, this article is suitable for GMD. The manuscript is well written, and the model validation part is done very well. Unfortunately, the performance comparison study, which is one of the main parts of the manuscript, has major flaws. I can only recommend publication after substantial revisions are made to this part of the manuscript. Moreover, while I applaud the authors for providing code that is supposed to reproduce their results, I encountered several issues when trying to run it.

This reviewer is familiar with Julia, but many potential readers won't be. The Julia compilation model, which is the key to its performance, is first briefly described in the results section and further discussed in the optimization tips section. These descriptions are not fully correct (see the specific comments below), and could be made more clear. I think it would be a good idea to centralize this material and provide a short introduction to Julia in the methods section, emphasizing how its compilation model enables high-performance computing and how static code generation facilitates GPU computations.

The authors obtained very impressive speed-ups from using GPUs - up to 100,000x for the computation time. Their result that the GPU computation times for their model do not depend on problem size is very surprising and, frankly, suspicious. Looking at the provided code, the authors seem to profile their model using the Julia macros `@elapsed` and `@benchmark`. However, based on the CUDA.jl profiling guide[1], this is not the right way to profile GPU code, which launches kernels asynchronously. The correct way is to use `CUDA.@elapsed` and add synchronization to functions profiled by `@benchmark`. Otherwise, only the cost of launching kernels will be timed and not the actual cost of computations. If this is how the GPU profiling was done, then the GPU benchmarks must be repeated and the manuscript revised based on the new results.

In general, while many speed-up numbers are provided in the paper, there is no rigorous discussion of the obtained values. These values should be put in the context of known hardware capabilities. MPAS-Ocean is a well-established code, and its performance bottlenecks are surely known. Since it is a low-order finite-volume code, most likely the main performance limiter is memory bandwidth. The authors provided many details about the hardware they used, such as peak flops, the number of CUDA cores, and cache sizes. However, the memory bandwidth value is only provided for the RTX 8000 GPU, but not the CPU. It is also unclear whether it was measured empirically, or taken from the vendor specification. Please focus on the hardware characteristics that are relevant to the model computational performance and frame the discussion of speed-ups around these characteristics.

While there are many ways to compare performance of heterogeneous systems, presenting comparisons of a single CPU core to a single GPU is not a valid practice. As the aim of the paper is to demonstrate the suitability of Julia for HPC, comparisons between a full CPU node and a GPU should be done, while perhaps noting their TDP values. As an MPI version of the code was created, this should be straightforward.

There is a number of possibilities to extend the performance analysis part of the paper, which, while not absolutely necessary, would significantly strengthen it. Since the number of kernels executed by each model time step is small, this present an opportunity to use profilers to do performance analysis at the level of individual kernels. Roofline plots could be created. The scaling study could also be extended to cover weak scaling.

[1] https://cuda.juliagpu.org/stable/development/profiling/

Specific comments

- Lines 38-40: "In recent years, shallow water solvers such as Oceananigans.jl (Ramadhan et al., 2020) and ShallowWaters.jl (Klöwer et al., 2022) have been developed in Julia. These codes (…) are equipped with capabilities for running on GPUs to achieve high performance." According to its documentation `Oceananigans.jl` is not just a shallow water solver, but can solve nonhydrostatic and hydrostatic Boussinesq equations. There is no mention that `ShallowWaters.jl` can run on GPUs in its documentation, and its `Project.toml` doesn't include any GPU packages.
- Line 77, equation (2a): shouldn't the kinetic energy term be under the gradient operator ?
- Line 82: (2b) is referred to as the discrete momentum equation. Shouldn't this be (2a) ?
- Lines 278-279: "making all types and subtypes concrete rather than abstract, to minimize on-the-fly compilation" How does making the types concrete minimize compilation ? Julia performs just-in-time compilation regardless of whether the supplied arguments are concrete or abstract. The cost of abstract types is the cost of missed optimizations and additional runtime dispatch.
- Lines 288-289: "In addition, single-precision floating point numbers (CUDA Float32 data type) calculations may execute significantly faster than Float64 (Julia Development Team, a)." It is true that most customer-grade GPUs have limited double-precision capabilities. This is usually not the case for GPUs targeting the HPC market. However, if the presented code is bandwidth limited, wouldn't the maximum possible speed-up from switching to Float32 be 2x ?
- Figure 3.: The title mentions occupancy, but only execution times are shown. Which kernel was profiled ? Are the results the same for other kernels ? It would be better if the x axis ticks showed some typical block size values, such as 32, 64, 128, 256, 512, and 1024.
- Lines 330-333 and Figure 4.: Any idea why Fortran-MPI computations scale worse than Julia ?
- Lines 336-347: Again, I found this discussion of the Julia compilation model not fully correct. Julia does not require variables with full type declaration to achieve fast code, since it performs aggressive type inference. The authors discuss an issue that only concerns type declarations of struct members. Moreover, the authors again seem to suggest that just-in-time compilation occurs only for `Any` or abstract types, which is not correct. Please revise this paragraph. As mentioned in the general remarks, some of the material regarding the compilation model should probably be presented earlier in the text.

Technical comments

- Some words are not capitalized consistently and correctly (ssh and SSH, python and Python, Numpy and NumPy).
- Line 9 in the abstract: "The GPU-accelerated Julia code is attained a speed-up" – spurious "is".
- Line 153: `pressueGradient` should be `sshGradient`.
- I was not able to run the provided code as is. I only tried to run the optimized version. The optimized code contains typos and references to undefined variables. Some examples of the problems I encountered:
  - in `GPU_CPU_performance_comparison_meshes.ipynb` the file `cuda_time_steppers.jl` is not included, which makes CUDA tendency functions undefined.
  - In the same file `calculate_ssh_tendency_cuda!` is used, but it is not defined in `cuda_tendencies.jl`, or anywhere else.
  - In `calculate_normal_velocity_tendency_cuda!` there is a typo in `mpasOean.maxLevelEdgeTop`. Moreover, `maxLevelEdgeTop` is not a member of the `MPAS_Ocean_CUDA` struct. There are more issues along the same lines. Please provide a version of the optimized code that can be run.
- I had some issues with obtaining the mesh files used in this study. The readme file points to a Zenodo archive[2]. However, the article uses meshes of size 128x128, 256x256, and 512x512. The Zenodo archive contains meshes of size 64x64, 96x96, 144x144, 216x216, and 324x324. Where can I find the mesh files used in this study ?

[2] https://zenodo.org/record/7419817#.Y63p4C-B1pQ