Reply on RC1
Marco A. Giorgetta et al.

Author comment on "The ICON-A model for direct QBO simulations on GPUs (version icon-cscs:baf28a514)" by Marco A. Giorgetta et al., EGUsphere, https://doi.org/10.5194/egusphere-2022-152-AC1, 2022

Reply to the reviewer comment 1 (RC1) from 2022-05-27

Dear reviewer

Thank you very much for your review comment on our article. We are happy about your positive judgment of the effective 6x CPU-to-GPU speed-up in the node-to-node comparison.

We have improved the manuscript based on your comments, especially related to your comments about the cluttered figures. These figure have been redrawn. Figure 5 now shows the time-to-solution and the cumulative strong scaling of the integration timer for the R2B7 benchmarks alone, combining the results from the three machines in a single panel to facilitate the comparison between the machines, while Figure 6 now shows time-to-solution and cumulative strong scaling for the components, with separate panels for each machine similar to the old Figure 5. The text of Section "6 Benchmarking results" has been changed accordingly, discussing now first the overall results from Figure 5 before continuing with the results for the components based on Figure 6. The discussion of the strong scaling is followed by a short section on the weak scaling, based on the tabulated results alone. We hope that these changes have removed the cluttering. The stronger separation of the discussion of the integration as a whole from the components is certainly beneficial. Thanks for the recommendation!

A pdf file (generated by latexdiff) of the differences between the old and new manuscript, which includes changes related to your comments, is attached.

For all other comments please find our point-to-point replies below.

Best regards,
Marco Giorgetta

Abstract

RC1: Line 15: Why is the strong scaling poor?
Probably you refer to the statement on line 17: "While the application shows good weak scaling making also higher resolved global simulations possible, the strong scaling on GPUs is relatively weak, ...". This judgment is based on findings presented in section 6. We found that the turn-over (or time compression) achieved with the base setups cannot be increased very much in an efficient way by increasing the number of compute nodes, i.e. by exploiting strong scaling, for example towards the goal of 1 simulated year per day. In practice the strong scaling on the GPU machines allowed to double once or twice the number of nodes of the base setups with still acceptable efficiency. In contrast, on the CPU machine even a 6 times doubling was still efficient, but here we start from a considerably lower turn-over rate so that no high turn-over can be achieved. It is this once or twice doubling on GPUs compared to 6 times doubling on CPU that lead to the statement "... the strong scaling on GPUs is relatively weak ...". The data for the strong scaling for each doubling step, $S_s$, are given in Table 3. And the cumulative strong scaling, $S_{s,cum}$, over multiple doubling steps is shown in Figure 5 for the whole model ("integrate") as well as for the model components.

1 Introduction

RC1: Line 55: The literature review on GPU ported weather models is not exhaustive. WRF entire dycore and select physics packages have been ported to the GPU gaining speedup of 5x to 7x, NEPTUNE's dycore (NUMA) has been ported to the GPU gaining speedup of up to 10x. I am sure MPAS has had OpenACC work too.

Indeed, the literature review is not exhaustive. Since we have implemented an end-to-end GPU implementation for real scientific runs, we are most interested in other complete implementations. However, we do respect the hard work which has gone into WRF and MPAS, and will add references to Huang, et al. (2015) for WRF, and Kim, et al. (2021) for MPAS (which incidentally reports a speedup of 2.38x).

2 Model configuration for QUBICC experiments

RC1: Line 70: Please provide details (names) of the "Sapphire" physics packages ported to the GPU. Was there a study to choose the physics packages needed for high-resolution grid simulations?

More details for the physics parameterizations comprising the ICON "Sapphire" physics are given further down in section 2.5 and its subsections. The text now refers to these subsections. We did not make a separate study to chose this set of parameterizations for our high resolution experiments. Rather we proceeded in 2 steps. First we excluded parameterizations which are not suitable at high resolution, because they were originally developed for significantly larger cell sizes, and secondly we use parameterizations which qualitatively correspond to the parameterizations used in the operational setups of the ICON model used by the German Weather Service (DWD) for global and regional weather predictions at 13 and 2 km resolution, respectively. The RTE+RRTMGP radiation scheme is a more modern version of the RRTMG code used for many years by DWD (and just recently replaced by the ECrad model). The cloud microphysics code is the same as in operational use. The vertical diffusion scheme is different but relies on similar theory and discretization. Similar arguments can be made for the land scheme.

RC1: Line 75: What is the time integration method used in ICON-A, I assume it is horizontally-explicit vertically-implicit (HEVI) scheme.

Yes, it is horizontally explicit and vertically implicit (HEVI). Details are described in the documentation of the dynamical core by Zaengl et al. (2015).

4 Porting ICON to GPUs
This is a very good decision that is often missed. If one decides to run for example a particular physics scheme on the CPU, then all the speedup will go down the drain.

Indeed, we knew from the beginning that the only possibility for speedup was to port all code within the time loop. Practically it meant that a considerable part of the model code had to be ported, including the more demanding dynamics and tracer transport codes.

I am curious if ICON-A has adaptive mesh refinement (AMR) capability? If so, is that handled directly on the GPU?

ICON-A has a static 'nesting' capability (telescopic grids of increasing resolutions over areas of interest), which is now being ported to GPUs for future applications. The key word here is 'static', which contrasts with AMR, which is implicitly dynamic. Therefore ICON-A does not have AMR capability.

Deep subroutine calls are indeed a problem but it is still possible to get an efficient kernel. A difficulty with that approach is if a lot of temporary arrays (e.g. 1D) are used in the inner subroutines, in which case, the compiler will do GPU memory allocations there. What I found very helpful, is to move those out of the inner loops to the topmost loop. Other than that, if the nested subroutines are simple, it should be equally efficient without the need for major rewrite.

We considered moving the innermost loops upwards in the call tree, but decided against it: this goes in the direction of a full code rewrite, and the spirit of OpenACC is exactly to avoid such extensive refactoring of the code. Interestingly the parallelization over the two innermost dimensions (nproma and nlev) is sufficient for good performance if there are only one, or very few, blocks of nproma. The more critical performance bottleneck is actually the function call overhead of GPU routines designated with ACC ROUTINE SEQ. This call overhead is considerable and is only avoided if the compiler is capable of inlining the code. This is not always the case, and proper inlining is the subject of ongoing optimizations.

Kernels vs parallel loop. Kernels is often the safest option when you start porting the code. You can add a `acc loop independent` to the loops to effectively get a `parallel loop`. There should not be any performance difference after that.

This depends entirely on the degree of support for KERNELS in the OpenACC compiler. For example, we understand GCC has very limited support for KERNELS. Since our work emphasizes the code portability aspect, we only used KERNELS for simple code cases, e.g., initialization of arrays using array syntax.

Use of atomics is good, but the second approach looks simpler. Have you also tried "coloring" schemes where each thread constructs an independent portion of the list.

Atomics performed poorly in the two OpenACC compilers we worked with: Nvidia and CCE. Anecdotally, the performance of atomics is now improving, however we had long since moved to the parallel scan method based on the CUB library. While this is specific to Nvidia GPUs, a library with similar interface is also available for AMD GPUs. The atomic functionality in OpenACC is actually quite simple to use, but we would only consider reactivating that code if there were guarantees of commensurate performance with CUB.

Coloring schemes only work well in a static context, e.g., when dependencies on Cartesian grids are consistently NSEW neighbors, thus cells can be divided into “red” and “black”, with each color being update in two separate steps. Coloring will also work for more
complex *static* dependencies. However, in this case the dependencies are determined dynamically through evaluating the fields for particular conditions, e.g., some velocity condition exceeded. Therefore, one would have to create the dependency tree, calculate the coloring, then perform the updates in multiple steps (there could be many dependencies for any given cell). Thus for ICON it is much more efficient to perform the parallel scan, as mentioned above and in the paper. This solution works sufficiently to avoid performance bottlenecks.

RC1: Line 415: Indirect addressing (lists) are not good for the GPU, and I wonder if the fact that it is used through the dycore suggests that it should be re-written for better performance on the GPU, although 6x speedup is already very good?

The performance loss due to indirect addressing is subject to discussion, with the NIM team (Govett et al., 2017) reporting limited overhead if the data are layed out properly. Since the icosahedral grid is quasi-regular, attempts were made by our team to avoid the indirect addressing. However, these ultimately would have lead to the development of a completely new model, and were thus discarded.

RC1: Line 460: GPU aware MPI should be the right choice here, otherwise the cost of data transfer between CPU/GPU will kill performance.

You are correct. While we saw only minor gains with GPU-aware MPI for Piz Daint, which has nodes with a single GPU and single CPU, there are *much* larger gains (a factor 2x) on architectures with multiple GPU nodes and a single CPU, e.g., Juwels Booster.

5 Validation

RC1: Line 695: I agree b4b test do not make sense for CPU/GPU validation. But a figure for both single and double precision calculations representing the deviation between CPU/GPU is a good target. Have you also tried PGI's auto-compare feature? It does redundant computations on both CPU and GPU, and notifies where they are diverging for a given level of tolerance.

We have tried Nvidia's PCAST functionality in certain debugging situations and have also implemented (see Section 5.1) a mode to compare the code running concurrently on different MPI processes, one running sequentially on CPU, with the others running on GPU, to a given relative and/or absolute error thresholds. However, due to the chaotic nature of underlying problem, this technique is only viable for debugging one or very few time steps. Ultimately the errors grow beyond any round-off bound, similar to what happens for a perturbed system, thus justifying the use of tolerance testing (Section 5.3).

6 Benchmarking Results

RC1: Line 820: Could this be a reason for the less than optimal strong scaling results?

Strong scaling is not optimal because, with increasing number of GPUs, the local block size simply becomes too small for it to run efficiently. This is less of a problem on CPU. We have tried to emphasize this point in the paper, e.g., "strong scaling on GPUs depends sensitively on the ability to maintain sufficient work for each node as the node count is increased."

RC1: Line 840: I am confused because I thought the 6.3x speedup is over the end-to-end computation including all of physics and dynamics?

Yes, for the single node setup we measure a speed-up of 6.4x for the entire time loop. On line 840 we only point out the lower limit of such a speed-up that would make any sense.
If we think about comparing the performance on one node with 1 CPU + 1 GPU to one node with 2 CPUs, a GPU over CPU speed-up should be significantly higher than 2 to make the GPU port a reasonable investment.

But here you present speedups of individual kernels. I am afraid Amdahl's law plays a significant role here, as the least performing kernel often governs overall speedup.

Yes, we think it is important to also present speedups of individual components. Amdahl's Law does play a role in the overall scalability, but it should not be assumed that the components Figure 4 have similar overall execution times. In particular, the time spent in the "land physics" (3x speedup) is insignificant compared to the time spent in dynamics (6.3x) or radiation (7.4x). To make this clear we have redrawn Figure 4. Panel (a) of the new figure shows the relative costs of all model components as a percentage of the time-to-solution of the integration. Obviously radiation and dynamics dominate the costs and land, cloud microphysics and vertical diffusion have only small shares in the costs. Panel (b) shows the speed-up of the integration and the components. Note that the "atmospheric physics" used in the old version of Figure 4 is no longer shown. Instead we show the results for all individual components, as used already in Figure 5.

RC1: Line 845: Could you give details how much percentage of time is spent in the land surface model specifically, but for others as well? If the overall speedup is is indeed 6.5x, land-surface should not be taking a significant portion.

Absolutely correct (see above). The new Figure 4 shows now the relative costs of the components. The text in the manuscript is adjusted accordingly. Concerning the land scheme, which has the poorest scaling, here the numbers for the relative costs: land on GPU: 4.3%, land on CPU: 2.0%.

Figure 5: Radiation is often the most-time consuming and I am not surprised you get the most speedup from it.

Unfortunately this is not automatically given. For the GPU port of ICON-A a first attempt was made to port the PSrad scheme, then the default radiation scheme for ICON-A, with OpenACC directives in a similar way as it worked for other physics parameterizations. This however did not reach any acceptable speedup. This was attributed to the code and data structures, where the additional dimension for the spectral resolution plays an important role. This lead to the replacement of the PSrad scheme by the RTE+RRTMGP scheme, where the developers have paid attention to these challenges from the beginning of the code development. RTE+RRTMGP also includes a small number of separate codes for CPU and GPU to account for the respective needs. But you are certainly right that radiation can reach a high speedup, simply because it can offer a lot of work and parallelism for a GPU, if the code is structure such that this can be exploited.

RC1: Some models often do not compute radiation every time step for this reason. Would you say that the GPU ported code offers a chance to do it every time step for more accurate results?

The setup used here computes radiative fluxes only every 18th time step. But still radiation is the most costly components when measured on a single node. Changing this to a setup with radiation computed every time step would result in 8 times higher integration costs. For our science case - the simulation of the QBO - we do not expect any advantage from such an increased frequency of the computation of radiative fluxes, and we cannot afford an 8-fold increase in costs. But we agree that the computation of all physical processes with equal frequency would generally be more appropriate.
RC1: Could you also provide figures for speedup, weak and strong scaling for end-to-end run without the sub-components? The figure is cluttered at the moment and I would like to see one figure highlighting the overall results.

The figures have been redrawn and the text adjusted, see our main reply.

RC1: Line 900: Land-surface model showing very poor strong scaling could be the cause of the model's over poor strong scaling then?

This is true to some extent, but the main reason, as mentioned above, is the strong scaling is limited by the work available for the GPU as the local block size decreases. Also, if - on GPUs - the land scheme scaled "perfectly", the whole model would still have a poor strong scaling for more than quadrupling the number of nodes. This can be seen from Figure 5, which shows the scaling as well as the time to solution. The reason is that also the components dynamics, transport, cloud microphysics and vertical diffusion, which together dominate the time-to-solution compared to land, have a scaling that is insufficient to achieve a good strong scaling.

Please also note the supplement to this comment: [https://egusphere.copernicus.org/preprints/egusphere-2022-152/egusphere-2022-152-AC1-supplement.pdf](https://egusphere.copernicus.org/preprints/egusphere-2022-152/egusphere-2022-152-AC1-supplement.pdf)