The paper discusses porting of the ICON-A model to the GPU. An effective speedup of 6.3x times is obtained in a node-to-node comparison, which is very good in my opinion.

Note: Some of the questions/comments may not be relevant as I am writing notes as I go through the paper and understand it better. I will probably send a second round of comments after I go through the paper once more.

Line 15: Why is the strong scaling poor?

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.

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?

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

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

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.

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.

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.

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

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

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.

Could this be a reason for the less than optimal strong scaling results?
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? 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.

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 indeed 6.5x, land-surface should not be taking a significant portion.

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

Could you also provide figures for speedup, weak and strong scaling for end-to-end run without the subcomponents? The figure is cluttered at the moment and I would like to see one figure highlighting the overall results.

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