Geoscientific Model Development Discussions

Geosci. Model Dev. Discuss., referee comment RC2
https://doi.org/10.5194/gmd-2022-141-RC2, 2022

Comment on gmd-2022-141

Anonymous Referee #2

Referee comment on "Porting the WAVEWATCH III (v6.07) wave action source terms to GPU" by Olawale James Ikuyajolu et al., Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2022-141-RC2, 2022

General comments

- The paper is well formulated in that the problem of interest is well described, the approach to GPU porting is well described, and some key aspects of performance are explained well in detail. I am requesting major revisions to the manuscript due to the importance of using appropriate baselines for GPU performance reporting. If the GPU performance is much slower than most CPUs, as seems to be the case, it is a very important data point for the reader to understand clearly.
- Comparing GPU runtime against a single CPU core is inappropriate because in production simulations, the entire CPU would have been used. I request that the authors change this comparison to use all available CPU cores in the CPU baselines when reporting speed-up numbers.
- On the same note, I request that the authors include a "roofline" plot of their ported kernel. This can be obtained directly using Nvidia's ncu-ui tool (or several other tools if desired). Absent this, then the authors need to at least provide the floating point operations per second (flop/s) achieved by the kernel as well as the maximum expected flop/s for the observed DRAM-oriented arithmetic intensity in their kernel. This is a more objective performance metric because the baseline is fixed for a given kernel and GPU hardware choice. The documents below should help in doing this:

https://www.nersc.gov/assets/Uploads/Talk-NERSC-NVIDIA-FaceToFace-2019.pdf

https://arxiv.org/pdf/2009.02449.pdf

Specific comments

- Line 52: The phrase "totally different" is not an accurate statement. The "breadth-first" SIMT dispatch of code over threads on GPUs is different than the more "depth-first" execution of code of CPUs. GPUs need a larger degree of parallelism exposed at one time than CPUs. Beyond this, much of the programming and optimization approach does remain the same. The order of execution as dispatched on the hardware should still match the order of memory accesses in arrays. Floating point and integer divisions and floating

point transcendental operators should be minimized. Data movement to and from DRAM should be minimized.

- Line 127: It might be more accurate to say that OpenACC contains the most mature implementation using the Nvidia compiler suite on Nvidia GPUs.
- Line 129: These lists do not correspond to one another in a respective sense; and I believe, as written, this will lead to confusion for the reader. An unspoken yet commonly used mapping from OpenACC levels of parallelism to CUDA levels of parallelism is as follows: gang == blockIdx.x (i.e., "grid"-level parallelism); worker == threadIdx.y (i.e., "block" level parallelism); and vector == threadId.x (i.e., finer "block"-level parallelism). Perhaps a better more general statement is something similar to "Gang, worker, and vector parallelism expose increasingly fine granularities of parallelism to distribute work over grid, block, warp, and thread-level parallelism on Nvidia GPUs." That should be true in all cases for the OpenACC spec itself and Nvidia hardware.
- Line 130: Gangs must operate independently without synchronization.
- Line 132: SIMD is not an accurate description of the parallel dispatch strategy on Nvidia GPUs. Please describe it as "SIMT" (single instruction, multiple thread).
- Line 135: Please specify that declare create is needed specifically due to using module-level variables directly in device code instead of passing them by parameter.
- Line 138: Please add that the expectation is that "W3SRCEMD" will then further dispatch parallel threads in the worker and / or vector levels.
- Line 167: Can you give the reasoning for reducing the optimization on Summit? If bugs were encountered, this can be useful information for the reader to understand that these issues can crop up sometimes.
- Line 207: Occupancy is largely affected by register and shared memory usage. Using large local, thread-private arrays (i.e., on the stack) can affect register usage, but I believe it is incorrect to say that occupancy is affected by the size of the module-level arrays created outside the kernel. Can the authors explain in more detail what is meant here?
- Line 222: I may be misunderstanding this, but it's not clear why the impact of data transfers could not be assessed. Nvidia's nvvp and nsight tools should be able to show the cost of all transfers.
- Line 242: Using only 32 threads per gang seems like it would lead to problems hiding memory fetching latency from DRAM via thread switching within an SM. Have you tried increasing this to 64 or 128, or is 32 something required by the algorithm itself?
- Line 252: I think it's important to note at this point that there is another option that has, so far, not been discussed. The NSEAL loop could be pushed down the callstack into the innermost routines. While this would be a significant refactoring effort, it would allow developers to fission the one large kernel into multiple smaller kernels. This will increase the number of kernel launches (potentially increasing an important kernel launch latency cost), but each individual kernel would no longer suffer register spillage, which is likely the number one performance problem in the approach used in this paper. I believe this approach should at least be mentioned in the manuscript so that the reader understands there are multiple potential approaches.
- Line 257: There is not only a maximum number of registers per thread, but I believe there is a minimum number supported by the hardware as well, which may explain this behavior.
- Figure 8: The plot seems confusing because a blue line is represented for "time" in the legend, which make it seem like the red and green portions are potentially no longer representing "time". Perhaps add two entries to the legend so that the labels are "Occupancy", "Time (latency regime)", "Time (occupancy regime)", and "Time (neutral regime)".
- Line 274: I do not consider this to be an appropriate performance comparison. A GPU should not be compared to a single CPU core. It should be compared to an entire CPU with reasonable optimization efforts performed on both the CPU and GPU. For instance, if the GPU code is 6-7x faster than a single P9 core, then when using the P9 as it would typically be used (21-42 cores), the V100 performance is actually 3-6x slower than a single P9.

- Line 300: I greatly appreciate considerations of correctness in this paper. This is often overlooked in GPU refactoring manuscripts.
- Line 307: “exponentially” is likely an inaccurate term. Perhaps use “significantly” or a similar word instead.
- Line 318: Please mention the baseline here. Line 320 seems to indicate that the comparison is against an entire CPU whereas the manuscript seems to indicate that it is against only one core of the CPU.
- Line 323: The authors should mention briefly here the other potential approach described in the comment for line 252 above.
- Line 324: Are the authors certain the register usage is due largely to constants? What is the evidence for this claim?