Reply to Reviewer 2

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.

We thank the reviewer for pointing out the need for a better baseline comparison to understand GPU performance. We have completed a detailed node-level performance comparison of GPU and CPU resources on the Summit supercomputer and included in Section 3.3.1. Using all 42 CPU cores and 6 GPUs on a summit node (with 7 MPI ranks per GPU), a speedup of 1.4x (1.36x) is achieved for a mesh size of 228K (59K). We also ran 4, 3 and 2 MPI ranks per GPU configurations. The speedups of all the GPU configurations are compared with the full CPU 42 cores run (with 7 MPI ranks per GPU). The results (Table 3 in revised manuscript) of the different multi-process configurations are included in the revised manuscript. The revised manuscript also includes an updated Figure 6.

	CPU	**GPU**			
GRID	**42 ranks**	**42 ranks (7/GPU)**	**24 ranks (4/GPU)**	**18 ranks (3/GPU)**	**12 ranks (2/GPU)**
59K	180.72	133.18 **[1.36x]**	124.28 **[1.45x]**	127.17 **[1.42x]**	146.34 **[1.23x]**
228K	683.66	483.22 **[1.41x]**	523.91 **[1.30x]**	533.08 **[1.24x]**	612.60 **[1.12x]**
228K (W3SRCEMD)	589.56	389.12 **[1.52x]**	407.54 **[1.45x]**	366.00 **[1.61x]**	395.31 **[1.49x]**

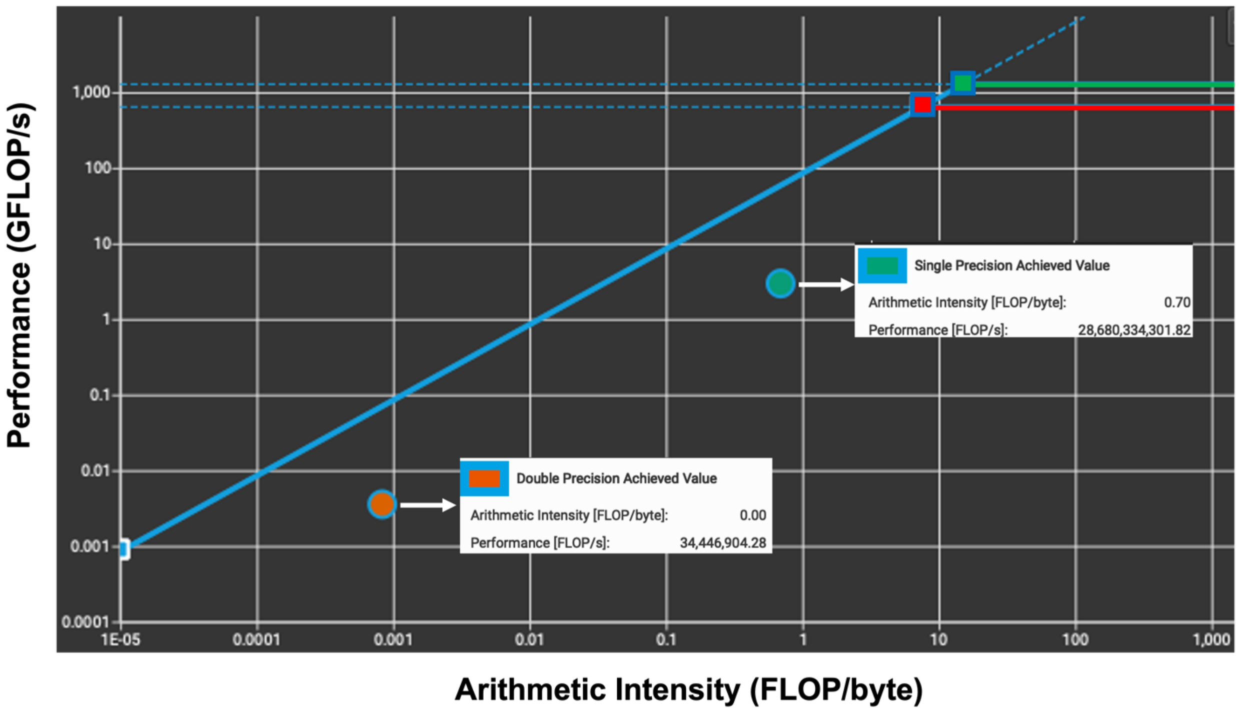
Table 3. Node baseline comparison runtime (seconds) with different MPI ranks per GPU configuration on a Summit node. The bold numbers in [ ] are the speedup relative to the whole CPU on a summit node (42 MPI ranks). In the last row, we present the speedups and simulation times when comparing only the GPU-accelerated subroutine, W3SRCEMD.

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

We wish to thank the reviewer for their positive comments and suggestions raised regarding the baseline comparison.

From the roofline model (Fig. 8 in the revised manuscript), the kernel is limited by the data transfer bandwidth between the CPU and the GPU. The kernel has a very low arithmetic intensity for both simple and double-precision floating-point (Fig. 8) computations, performing only very few flops for every double and integer loaded from DRAM. Most of the kernel's time is spent in executing memory (load/store) instructions. This is due to too many local scalars and arrays in the W3SRCEMD subroutine and the huge WW3 memory requirement. For example, VA, the spectra storage array, is approximately 5Gb (20Gb) for a spatial grid size of 59,000 (228,000) and spectral resolution of 50x36. VA is one of the large arrays used in W3SRCEMD. W3SRCEMD is simply too big and complicated to be ported using OpenACC routine directives and therefore requires significant code refactoring. We have modified the manuscript based on the node comparison and roofline analysis. In sections 3.3.1 and 3.4 of the revised manuscript, we provide a detailed explanation of the baseline comparison and roofline analysis, respectively.



🟩 Single Precision Achieved Value	
Arithmetic Intensity [FLOP/byte]:	0.70
Performance [FLOP/s]:	28,680,334,301.82

🟧 Double Precision Achieved Value	
Arithmetic Intensity [FLOP/byte]:	0.00
Performance [FLOP/s]:	34,446,904.28

Figure 8. Roofline model for W3SRCEMD kernel from Nvidia Nsight Compute. The red and green dots are the double and single precision values respectively.

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.

Here, we are referring to just the structure of the code. CPU codes need to be modified by adding directives and in some cases require little or more refactoring. We have removed the 'totally' to avoid confusion.

Line 127: It might be more accurate to say that OpenACC contains the most mature implementation using the Nvidia compiler suite on Nvidia GPUs.

We have modified the sentence according to the reviewer's suggestion.

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.

We have modified the sentence according to the reviewer's suggestion.

Line 130: Gangs must operate independently without synchronization.

We have removed the 'can' in the sentence to emphasize this.

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).

We have modified the word according to the reviewer's suggestion.

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.

We already stated this in section 3.2.1.

Line 138: Please add that the expectation is that "W3SRCEMD" will then further dispatch parallel threads in the worker and/or vector levels.

We have modified the sentence.

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.

Yes, the -O3 optimization flags do not run for summit. Worked with the OLCF support but couldn't figure why. It does not affect the conclusion. We have added this as a footnote to the revised manuscript.

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?

The sentence should be 'Many local arrays, sometimes of spectral length, and scalars used within W3SCRE and its embedded subroutines'. We have modified this in the revised manuscript.

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.

The sentence has been removed since we have determined that CPU-GPU data transfers dominate the program.

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?

Yes, 32 is something required by the algorithm probably due to register or shared memory usage.

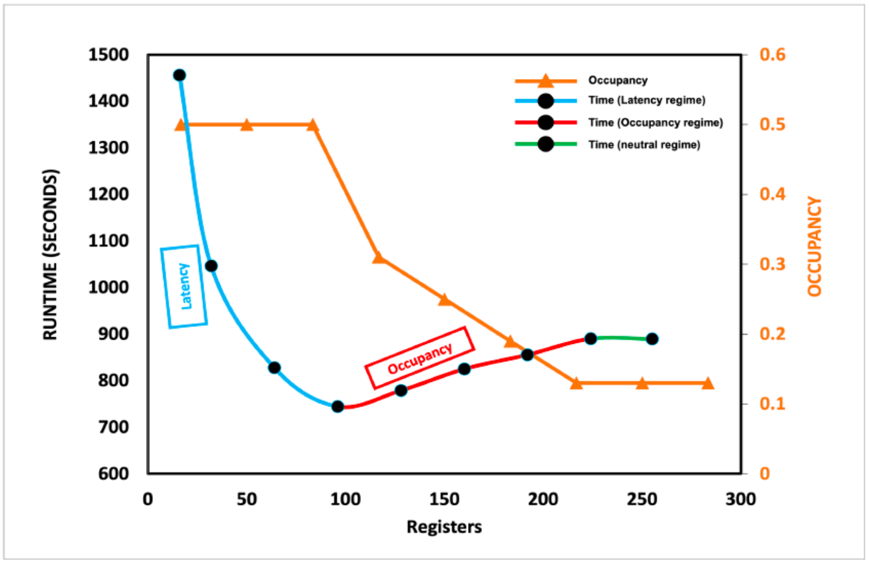
Line 252: I think it's important to note at this point that there is another option that has, so far, not been discussed. The NSEAL loop could be pushed down the callstack into the innermost routines. While this would be a significant refactoring effort, it would allow developers to fission the one large 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.

Yes, there is room for further GPU optimization by pushing down the grid loop counter into the W3SRCEMD which would require major code refactoring. Moreover, this is the only way to reduce register usage, increase GPU occupancy and avoid launching a single large W3SRCEMD kernel. We have included this potential in the conclusion section of the manuscript.

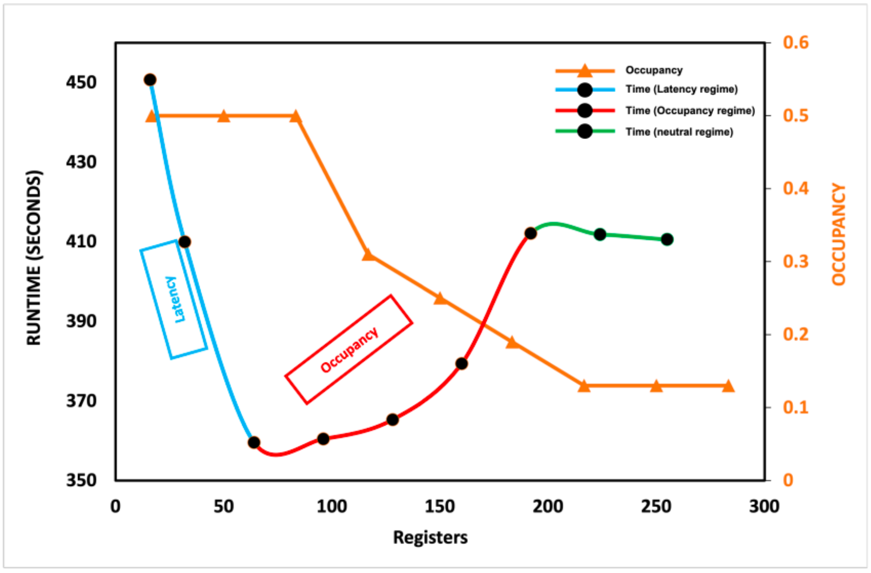
Line 257: There is not only a maximum number of registers per thread, but I believe there is a minimum number supported by the hardware as well, which may explain this behavior.

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)".

The figure (Fig. 5) has been modified based on the reviewer comments.



(a)



(b)

Figure 5: WW3-W3SRCEMD.gpu runtime (primary vertical axis) and occupancy (secondary vertical axis in orange) based on register counts on Kodiak (a) and Summit (b). For the runtime, the light blue part of the line represents the latency-dominant region, the red part represents the occupancydominant region, and the green part represents the neutral region. We analyzed the result of running 228K mesh size

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.

Once again, we would like to thank the reviewer for bringing this up. We have added a node level baseline/comparison section to the manuscript. The baseline comparison was done only on the summit machine because KODIAK LANL has been decommissioned. Section 3.3.1

Line 300: I greatly appreciate considerations of correctness in this paper. This is often overlooked in GPU refactoring manuscripts.

We would like to thank the reviewer for the kind words

Line 307: "exponentially" is likely an inaccurate term. Perhaps use "significantly" or a similar word instead

We have modified the sentence.

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.

We have modified the conclusion and manuscript to focus on the baseline comparison.

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?

W3SRCEMD subroutine has several large subroutines (based on the type of source term switches selected) and they make use of many scalars and constants.