Dear reviewer,

Thanks a million for your precious time and comments. We will reply the reviewer's comment point-to-point in the following part.

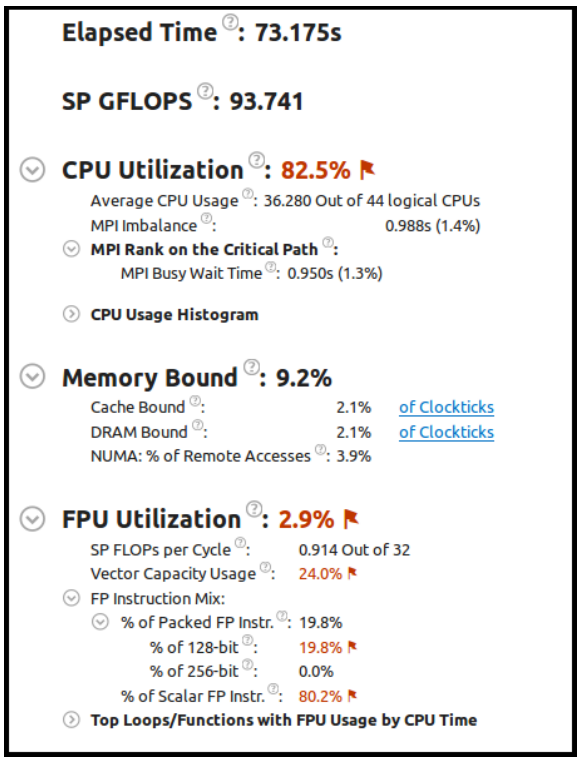
"The paper describes the work done by the authors on optimizing the GNAQPMS model for Intel Knights Landing. The paper is well organized. I agree with the authors that there are still options to further optimize the code but the work performed so far isa good first step and worth publishing. There are a few changes that need to be addressed. Also the language could use some general improvement. I will first give some specific comments that I believe need to be addressed before publication. After that I list a few (optional) suggestions for future optimizations."

Reply: We really appreciate for your praise. We believe this work could be a good example for the model developer who wants to transplant their model to KNL platform. Thanks for your comments. As for the language and grammatical issues, we will invite the native English speaker or some relative company to do the copy editing for English.

According to your specific comments, the following modifications have been done.

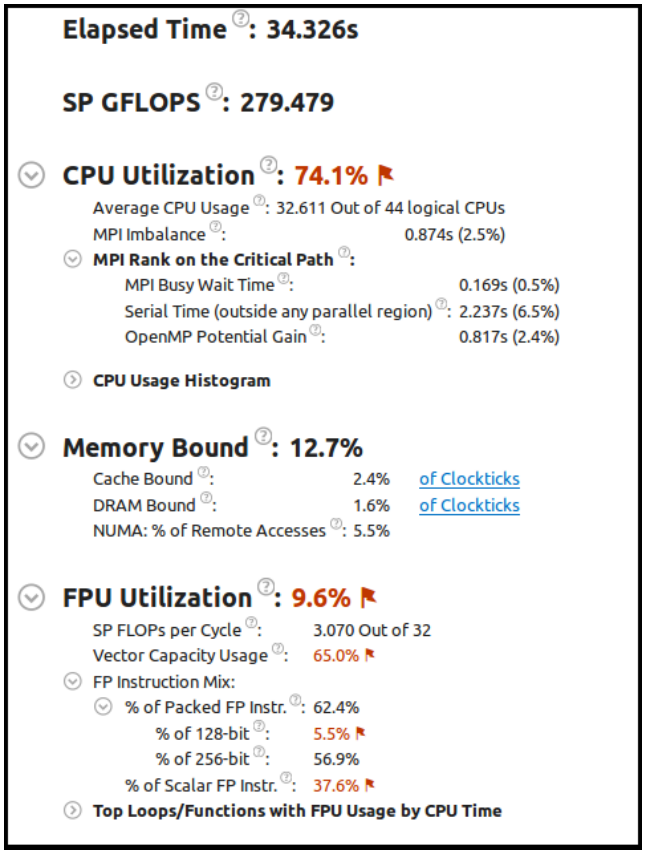
(a) section 1, page 3: according to the introduction Section 3.1 is supposed to discuss where the bottlenecks of the code are. I assume that this refers to the runtime measurements shown in Figure 2? Under the term "bottleneck" I would have expected a discussion whether the code is bound by memory bandwidth by showing measurements of memory bandwidth and flops/s and comparing them with the peak performance obtained for the STREAM and LINPACK benchmarks. I understand that hardware counters are not very accurate on Intel architectures but you could still count them with an emulator or by hand. The paper does not show in its current state what the bottlenecks of the different parts of the code are.

Reply: Thanks for your kind comments. We agreed with the reviewer. We have tried to measure the memory bandwidth and flops/s of the Base-V GANQPMS by the Vtune tools (R1).



R1 the HPC-performance report of the Base-V GNAQPMS

The FLOP/S of the Base-V GNAQPMS detected by Vtune showed in R1 is about 93.714 GFLOPS, and that of the Opt-V GNAQPMS is reaching 279.326 GFLOPS (R2). The Vtune Memory Bound measures the fraction of slots where pipeline could be stalled due to demand load or store instructions. And both of two reports indicate that the memory bound is not the dominant limitation of model performance.



R2 the HPC-performance report of Opt-V GNAQPMS

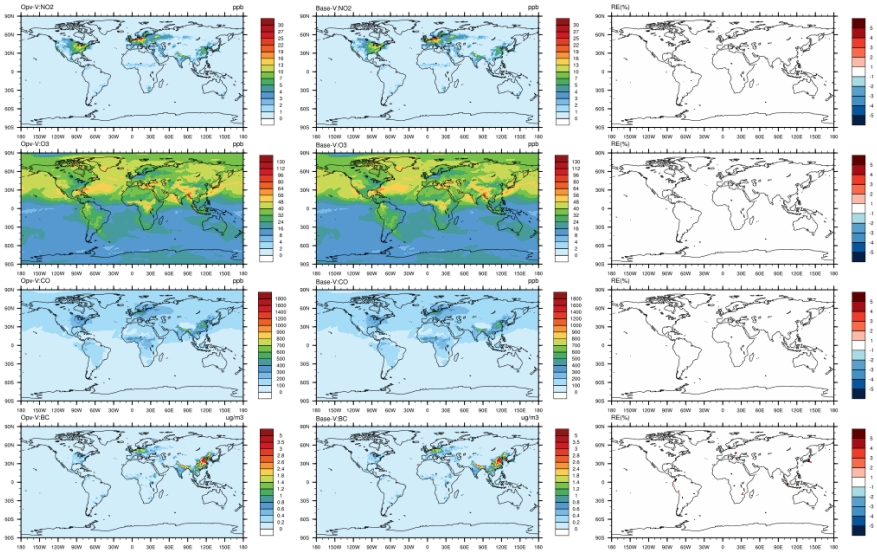
However, because of the bad modularity of GNAQPMS, we cannot get the testing data of GFLOPS and bandwidth of every section of the model, and data we got is based on the function- and loop-level. And obtaining the data of each section may lead to a large amount of work to pack the code, which could not be finished immediately. According to the current test results, we still can draw a preliminary conclusion that the insufficient use of vectorization is still a main bottleneck for our model but not the bandwidth. And we have done the further work to improve the vectorization and got a good speedup, which would be presented in the future paper.

(b) section 4.2, page 9: comparing the patterns and values without giving any specific relative difference between base and optimized version of the code is not enough to claim that the results are identical. The authors need to show some precise numbers like the total mass of the air and the different chemical species in both versions and the difference between optimized and base version.

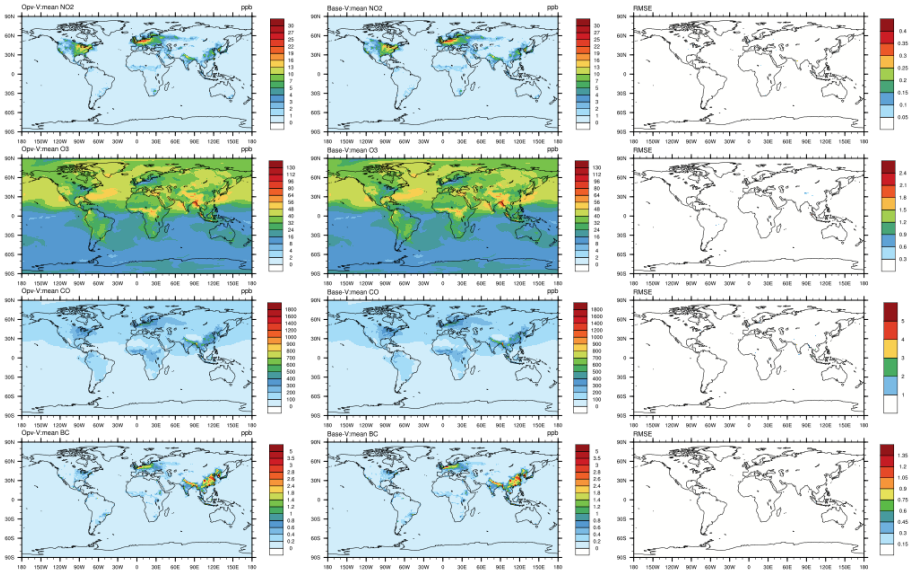
Reply: Thanks for your precious comments. Actually, we designed two mechanisms to verify our results from the two models (Opt-V, Base-V). At first, we added an extra module to test

our results. The function of this module is outputting the concentration of specific chemical species after each chemical or physical processes. The processes write its own data into its own files respectively at the same time, and each chemical and physical module would only run one time to insulate the effect of other modules. Then, an additional small program will read the files from the two version of GNAQPMS and calculate/report relative error and absolute error. This method is likes that sampling the results during the running period. By this way, we can find the primary error due to every section. On the other hand, we could plot the spatial distribution of two model results after a long time integration, as we did in the paper. Actually, we calculated the difference of two plots in Fig. 5(in paper), and considering the beauty, we only put the two spatial distribution plots. And according to the first step, after a serious of test and debugging, we found that the compile flag, such as -xCore-AVX2 and –xMIC-AVX512, would affect the results because of the sensitiveness of calculation accuracy. The same –fp-model flag could reduce the error raised by advection, but the error could not be completely diminished because of numerical sensitivity.

Generally, there is no obvious difference between the spatial distribution by calculating the Relative Error (R3) and Relative Mean Square Difference (R4), and the REs of almost all grids are lower 1%, which, we think, is acceptable.



R3 he Relative Error between the results of Base-V and Opt-V GNAQPMS.



R4 The Relative Mean Square Difference between the results of Base-V and Opt-V GNAQPMS.

(c) section 4.3, page 9: how did you measure the power consumption? Do you trust VTune to give you these measurements or did you measure the power consumption yourself?

Reply: Thanks for your precious comments. Both Xeon and Xeon Phi platforms have the same built-in power and thermal sensors. We measured the power consumption with an IPMI-based script tool to query these sensors from a remote server at a constant interval, such as 0.02 seconds. IPMI, short for “Intelligent Platform Management Interface”, is a low-level interface specification that allows remote management at the hardware level without dependencies on the operating system. IPMI communicates with the server baseboard management controller (BMC), which is a reliable agent in the system for management and gathering system health, including power consumption data.

(d) How many OpenMP threads were used per MPI process? Did you try different configurations (like 2, 4, 8, 16, 32, 64 threads per MPI process)?

Reply: Thanks for your kind comments. At first, we unfortunately found that single node test environment (Cthor Lab.) had some adjustment of the machines, which lead to the difference of test results. Considering the time saving, our previous tests about the combination of

OpenMP threads and MPI processes are implemented by using the 1h-running test; we chose the best combination to do the 48h-test. Therefore, we retested all the results by testing the 48h workload and updated the data in the revised paper (Table 1). We tested the combination of different OpenMP threads and MPI processes as suggested by the reviewer. All of tests are fully using the hardware threads. And the results indicates that the best combination on CPU platform is 6 OpenMP with 12 MPI processors. For KNL, we use the command line "–env I_PIN_MPI_DOMAIN=N" to pin the OpenMP threads to the MPI processes. Moreover, the combination of 4 OpenMP and 34 MPI processes could get the optimal speedup (3.51, which is 3.34 in discussion paper).

Table 1 the speedup and walltime of different combination of OpenMP threads and MPI processes.

CPU（E5-2697 V4 with 36 physical cores and 2 hyperthreads）				
	OMP	MPI	WALLTIME	SPEEDUP
Baseline (No HyperThread)	0	36	4381.2	1
Opt-V	1	72	1769	2.477
	2	36	1625.72	2.695
	4	18	1614.9	2.713
	6	12	1580.1	2.773
	12	6	1612.3	2.717
	18	4	1790.2	2.447
	36	2	2243.4	1.952
KNL(KNL 7250 with 68 physical cores and 4 threads)				
Opt-V	2	136	1499.2	2.922
	4	68	1402.9	3.12
	2	68	1512.8	2.896
	4	34	1248.3	3.509
	8	34	1373.6	3.189
	16	17	1473.2	2.974

(e) The term "manual vectorization" is used many times throughout the paper. This is very misleading. Manual vectorization would be in my opinion if the code was rewritten with avx512 vector intrinsics in C! The vectorization used in this paper still relies on the compiler.

Reply: Thanks for the comment. We agreed with the reviewer. To avoid misleading the readers, we consider that “vectorization with compiler’s directives” may be a more accurate expression. And we will update this part in the revised paper.

Suggestions

(f) As mentioned before the paper does not investigate what the real bottleneck of the code is and how far the code is from optimal performance. I highly recommend to create a theoretical and measured roofline model. This would allow to answer how much potential for further optimization should still be possible.

Reply: Thanks for the precious comments. As we discussed in the Question 1, it is not easy to investigate the rootline of every section, and we can provide the general report in R1 and R2 now. We think it could need more time to test and establish a reliable roofline model. According to the test results, we consider that the model still has the optimization potential. And our goal is that our model can get the speed of 5-10 model/day.

(g) I would love to know how many of all floating point operations are vectorized. I understand that counting floating point operations on Intel architectures through hardware counters is not very accurate. Maybe you could give some estimates from what the vectorization report shows you?

Reply: Thanks for your kind suggestion. From R1 and R2, the FPU utilization is increased from 2.9% to 9.6% after optimization, and the vector capacity usage is increased from 24% to 65%.