Geosci. Model Dev. Discuss.,
https://doi.org/10.5194/gmd-2020-323-AC1, 2021

Geoscientific Model Development Discussions

Interactive comment on "The GPU version of LICOM3 under HIP framework and its large-scale application" *by* Pengfei Wang et al.

Pengfei Wang et al.

lhl@lasg.iap.ac.cn

Received and published: 16 February 2021

GMD-2020-323 by Wang et al. Responses to Reviewer(s)

Feb. 10, 2021

I. Response to Reviewer #1

Interactive comment on "The GPU version of LICOM3 under HIP framework and its large-scale application" by Pengfei Wang et al.

Anonymous Referee #1 Received and published: Jan. 6, 2021

The paper presents implementation of LICOM3 using HIP framework targeting an HPC cluster with AMD GPUs. Atmospheric modeling in not my area of expertise, there-

Printer-friendly version

Discussion paper

fore I will not comment on this aspect of the paper; I will only comment on the model implementation and parallelization efforts. Here are some specific comments about individual sections.

Response: Thanks for your insightful comments. We were planning to document the final version and test at first. Therefore, we only present the final version and the results from a large-scale test in the manuscript. Another paper shows details of the model porting and results of the small-scale test are preparing. Based on your suggestions, we have done some additional tests, added much more information of the system's software and hardware, and further polished this manuscript's English. Also, we have modified the format of the references. The point-to-point responses are listed below.

Section 2.3: "The nodes of both partitions are interconnected through the high performance InfiniBand (IB) networks." Which partitions? Prior system description does not mention any partitions, instead it states that the system consists of 7000 nodes with 4 GPUs per node.

Response: Thanks for your questions. In fact, the implementation of the network is a 3-level fat-tree architecture using Mellanox 200Gb/s HDR InfiniBand, whose measured point-to-point communication performance is about 23GB/s. The supercomputer consists of 6 partitions (or rings, each has 1200 nodes, and the job system schedules about 7200 nodes). We have modified the descriptions in the revised manuscript in Section 2.3, Lines 132-136.

Section 3.1: There is some repetition here, e.g., the authors state several times that HIP supports both AMD and NVIDIA GPUs.

Response: Thanks. We have revised the repetition in Section 3.1.

Section 3.2: Good work on converting the original Fortran code to C and verifying the results to be identical between the two implementations. However, the authors do not say much about this ported code in terms of the optimizations applied to it, if the code

is purely executed as a single thread on each node, if any 3rd party libraries have been used, etc. Also, how the execution time of this newly ported C version of the code compares to the execution time of the original Fortran code? It is important to have a well-performing CPU code as a starting point.

Response: Thanks for your comments. First, we would like to correct the statement of this porting. Actually, we have only ported the main computation part of LICOM3 from Fortran code to C, which includes the seven kernels listed in Figure 2 (readyt, readyc, barotr, bclinc, tracer, icesnow, and convadj). All seven kernels account for about 25% of the code and about 99% of the computation and communication time (Table R1). Other parts, mainly the initialization, the time integration control, and the IO parts, still use Fortran code. Therefore, the so-called C version of LICOM3 is a Fortran and C mixed code, not a pure C code. In this way, we can replace F90 subroutines one by one and easily check the correctness of ported C codes. The C code is purely executed as a single thread on each node.

The Fortran and C mixed code is the starting point of following CUDA-C or HIP programming. To guarantee the correctness of the porting, we did NOT optimize the model in this step, and there were not any additional 3rd party libraries used. We simply translated the Fortran code to C code sentence by sentence. The translations include converting array index from Fortran (start from 1) to C (start from 0), changing the array subscript sequence, etc. Since the code is the same for 100km and 5km, we only test the Fortran and C mixed version for lower resolution cases.

We have tested the byte-to-byte correctness of the Fortran and C mixed code and tested the execution time. The Fortran and C hybrid version's speed is slightly faster than the original Fortran code, less than 10%. Figure R1 shows a speed benchmark by the LICOM3 for 100km and 10km running on an Intel platform. The results are the time running one model month for a low-resolution test and one model day for a high-resolution test. The details of the platform are in the caption of Figure R1. This indicates that we have successfully ported these kernels from Fortran to C.

We have added more descriptions of the porting and added a figure (Figure R1) in the revised manuscript in Section 3.2, Lines 187-195.

Section 3.3: From section 3.2, I understood that the entire code was ported to C. However, section 3.3 states that a mixed Fortran/C code was used here. Please clarify to resolve this inconsistency. Also, I do not quite follow the discussion in lines 222-227 about array size. I understand that the original Fortran code uses static arrays and these arrays need to be changed to be dynamically allocated in order to move them to the GPU global memory. Is this what you are implying here? I also have a hard time following the discussion about halo data packing and Fig. 4.

Response: Thanks for your questions. Sorry for the confusing statements. As our response to the previous section, the code of LICOM3 is not a pure C code but a Fortran and C mixed one. We have modified the vague statement in the manuscript to avoid this inconsistency in Lines 187-189.

The "lines 222-227" problem is not really what you understand. It occurs for the 5km resolution version when the static data array beyond 2GB. We met this problem in both the HIP hipcc and CUDA nvcc compiler. It is perhaps due to the compiler limitation of the GPU compilation tool. We have revised this part to state the problem clearly, in Lines 228-230.

To optimize the heavy halo in the kernel "barotr", we have tried to reduce the amount of data using the halo data packing method. Only the necessary data are transferred. The test indicates that the method can decrease by about 30% of the total wall clock time of "barotr" when 384 GPUs used. But we have not optimized other kernels so far because its performance not as good as 384GPUs' when GPUs scale beyond 10000. This optimization is not turn on in the following experiments, and we just keep it here as an option to improve the performance of 'barotr'. Other methods, such as NCCL (NVIDIA Collective Communication Library), are also implemented and tested, but not presented here. We have revised this part to state the problem clearly, in Lines 252-

255.

Section 3.4: What storage and file system do you use? These I/O performance numbers are not very meaningful without specifying the underlying storage architecture. It is possible that the low original performance of the I/O operations was solely due to a low-performing storage system and would not be an issue on a higher-performing storage. Also, how exactly is data loaded/stored to disk? The size of data transfers would play a huge role on the overall performance.

Response: Thanks. The storage file system of the supercomputer is ParaStor300S with a 'parastor' file system, whose measured write and read performance is about 520GB/s and 540 GB/s. We have added this information in the revised manuscript, Lines 138-139.

We agree with you that the I/O time depends on system I/O performance. When the large scale LICOM3 tests are performing, the system is forced to be one user mode by the administrator. It means no other users' job and heavy I/O tasks are running at the same time. Hence, the I/O of storage is enough to hold all LICOM3's I/O tasks. The improvement of I/O performance has been depicted in Figure 5 of the manuscript. The ratio of I/O in the total calculation time is less than 50% for 9216 cards, which is almost 90% before the I/O processes have been optimized.

After optimization, the forcing data, about 3GB total, are read from disk every model year, while the daily mean predicted variables, about 60GB total, are stored to disk every model day. Therefore, the performance of output dominated the I/O performance. We have added the above statements in the revised manuscript, Lines 256-257.

Section 4.1: I think the correct way to quantify the speedup is to use all CPU cores vs. the GPUs, not just a single CPU core. Also, it is not clear from the text how the GPUs are managed. For example, is each GPU managed by a single CPU thread, or is the same thread is used to manage all 4 GPUs?

Response: Thanks. We agree with you and have already computed the speedup using the one CPU, but not shown in the manuscript. Since each node has 32 CPU cores and 4 GPUs, each GPU is managed by one CPU thread in the present cases. We can also quantify GPUs' speedup vs. all CPU cores' on the same number of nodes. For example, the 384 (768) GPUs correspond to 96 (192) nodes, which have 3072 (6144) CPU cores. Therefore, the overall speedup is about 6.375 (0.51/0.08) for 384 GPUs and 4.15 (0.83/0.2) for 768 GPUs (Figure 6). The speedups are comparable with our previous work porting LICOM2 to GPU using OpenACC (Jiang et al., 2019), which is about 1.8-4.6 times speedup using one GPU card vs. two 8-core Intel GPU in small-scale experiments for specific kernels. Our results are also slightly better than Xu et al. (2015), which has ported another ocean model to GPU using Cuda C. But due to the limitation of the number of intel CPUs (maximal 9216 cores), we didn't obtain the overall speedup for 1536 and more GPUs. We have added discussions about this issue in the revised manuscript, Lines 300-307.

Section 5.1: Interesting discussion about failure rates. However, I wonder how realistic your assumptions are with regards to existing studies such as https://www.christianengelmann.info/publications/gupta17failures.pdf.

Response: Thanks for your suggestion and the interesting paper. We have read this manuscript and have cited it in the revised version. Unlike Gupta's study (23 types of failures), only three types of failures we have mostly met are discussed in Section 5.1. We only do simple analysis by predefined hypothesis (failure occur rateãĂŰ"10" ãĂŮˆ"-5" , which means 1 failure in 100000 hours.) to illustrate the difficulty of submitting jobs beyond 10000 GPUs. The realistic failures are essential, but not the critical problem of this study, so we did not discuss it more. Lines 360-364.

Overall, the presented implementation is rather straightforward and not well-thought. The authors ported to GPU several time-consuming kernels within the existing HPC application. Such approach is typically not very productive as it limits the design choices and does not give enough flexibility to optimize the overall application. Before proceed-

ing with such an effort, the authors should have analyzed the amount of time spent in each of the subroutines on the CPU with regards to all other aspects of the code, such as I/O and network communication. Figure 8 shows such kernel time distribution after porting, but I could not find much about the amount of time spent on cross-node communication. It is important to understand the potential benefits of one or another approach before starting the actual implementation effort.

Response: Thanks for your comments and suggestions. Before the implementation, we actually have tested the performances of all seven Fortran subroutines on a supercomputer system using Intel CPUs. Figure R2 shows our test results, including the seven subroutines' time cost percentage for 384 and 9216 CPU cores and the subroutines' time cost at six scales. Here, we used the 1/20° resolution LICOM3. Although the two experiments have been conducted on different platforms, the two model versions' configurations are identical, particularly using the same time steps, 1s for the barotropic part, the 60s for both baroclinic and tracer parts. That will grantee the same numbers of halo for two versions. Because the code of I/O in Fortran and the Fortran-C mixed version is the same, we will not discuss more I/O for the original Fortran code.

We found that the "tracer" is the most time-consuming subroutine for the CPU version. With the increase of CPU cores from 384 to 9216, the ratio of cost time for "tracer" is also increasing from 38% to 49%. "readyt" and "readyc" are computing intensive subroutines. "tracer" is both computing intensive and communication intensive subroutine. "barotr" is communication intensive subroutine and the communication of "barotr" is 45 times than that of "tracer". The computing intensive subroutines can achieve good speedup by GPU, but the communication intensive subroutine will achieve poor performance.

We have done a set of experiments to measure the time cost of both halo update and memory copy in the HIP version. These two processes in the time integration are conducted in three subroutines: "barotr", "bclinc," and "tracer". The figure shows that "barotr" is the most time-consuming subroutine, and the memory copy dominates,

which takes about 40% of the total time cost. These results can also be treated as a reference for the CPU experiments because the Halo update codes in the HIP version are the same as the CPU version.

We have added discussions about the above issues in the revised manuscript, Lines 240-254.

Next, there is no discussion about dominant computations in each of the kernels and how to best implement them on the GPU. There is no information in the paper that would indicate anything about the quality of implementation of these GPU kernels, e.g., how well they use GPU resources. I would like to see achieved FLOPS and memory bandwidth of these kernels with respect to the roofline model for this particular GPU to be convinced that the authors did a good job porting these kernels. The discussion about performance improvements is very convoluted by the fact that the authors start comparing performance at some rather large scale of 384 GPUs. What about performance of a single GPU on a much smaller model vs. performance of a single CPU socket, or a 4-GPU node vs. all CPU cores in that node? There is no discussion about what happens inside a single node, for example, how well are all 4 GPUs are utilized and if there is anything that one could benefit from the fact that these 4 GPUs have access to the same host memory.

Response: Thank you very much for your suggestion. This paper's critical point is developing a high-resolution heterogeneous version of the ocean circulation model, so we just give out some brief introduction of GPU implementation and optimization methods in section 3. Follow the Reviewer's suggestions; we append more discussions about floating-point operations performance of the subroutines in section 4.1. The addition roofline test is archived by a 100km version.

Figure R4 shows the roof-line model using the Stream-GPU and LICOM program's measured behavioral data on a single computation node bound to one GPU Card depicting the relationship between arithmetic intensity and performance floating points

operations.

The blue and gray oblique line is the fitting line related to the Stream-GPU program's behavioral data using 5.12e8 and 1e6 threads, respectively, both with blocksize of 256, which attain the best configuration. For details, the former is approximately the maximum threads number restricted by GPU card memory achieving the bandwidth limit to 696.52 GB/s. In comparison, the latter is close to the average number of threads in GPU parallel calculations used by LICOM, reaching the bandwidth of 344.87 GB/s on average. Here we use the oblique gray line as a benchmark to verify the rationality of LICOM's performance, accomplishing the bandwidth of 313.95 GB/s averagely.

Due to the large calculation scale of the whole LICOM program, the divided calculation grid bound to a single GPU card is limited by video memory; most kernel functions actually issue no more than 1.2e6 threads. As a result, the floating-point operations performance is a little far from the oblique roof-line shown in Figure R3. In particular, the subroutine bclinc apparently stray off the whole trend for the reason of including frequent 3d-array Halo MPI communications as well as a lot of data transmission between CPU and GPU.

There is no discussion about how the halo exchanges are implemented at the MPI level, how much overlapping is happening for computation, communication, and I/O.

Response: Thanks for your suggestion. The halo exchange in the MPI level is similar to POP have (see Jiang, et,al. 2019). We did not change these codes in the HIP version. We call an F90 subroutine to do halo from GPU space. The overlapping between communication data packing/unpacking and point-to-point communication was implemented. We did not apply the overlapping of computation, communications, and I/O in this version because the calculation time was much less than communication time and I/O time. There have no existing solutions to increase these halo performances; we hope it can be improved in the future LICOM3 version. This work's core contribution is to develop a 10000+ GPUs runnable ocean model, which still has sped up in

26200 scales. The seven core computation process put into GPUs space by the HIP framework is the critical solution. All other are not important issues because they cost 99%+ computation time in the model 'stepon' procedure. The brief hardware and software environments are presented in the manuscript. Since all supercomputer has their unique settings, this model may have different performance in other computers than we have.

The paper is short on many technical details, ranging from characteristics of the underlying hardware, to software/compiler environment, to implementation details, making it very difficult for others to put the results obtained by the authors compared to other work. Response: Thanks. We have revised our manuscript following your suggestions, and We have tried to provide much more information as we can.

Above all, the paper is hard to read due to a very poor use of language; almost every sentence needs corrections.

Response: Thanks for your suggestion. We have revised the manuscript as possible and will invite a 3rd part English editor to improve the revised manuscript.

If the authors want their paper to be printed with the existing implementation, they must provide a much better description and analysis of this implementation. Interactive comment on Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2020-323, 2020.

Response: Thanks. We have revised our manuscript following your suggestions, and We have tried to provide much more information as we can.

Reference

Jiang, J. R., P. F. Lin, J. Wang, H. L. Liu, X. B. Chi, H. Q. Hao, Y. Z. Wang, W. Wang, and L. H. Zhang, 2019: Porting LASG/IAP Climate system Ocean Model to GPUs using OpenACC. IEEE Access, 7(1), 154490-154501. doi:10.1109/ACCESS.2019.2932443

Xu, S., Huang, X., Oey, L. Y., Xu, F., Fu, H., Zhang, Y., and Yang, G. (2015). POM. gpu-v1. 0: a GPU-based Princeton Ocean Model. Geoscientific Model Development,

8(9), 2815-2827. https://doi.org/10.5194/gmd-8-2815-2015

Table R1 The seven major subroutines' time cost for one model day at different scales for LICOM3 (1/20°). The time cost of I/O and total time are also listed. These tests were conducted on a platform of Intel Xeon CPU (E5-2697A v4, 2.60GHz). Unit: second.

384 cores 768 cores 1536 cores 3072 cores 6144 cores 9216 cores readyt 1678.93 784.14 454.38 278.18 142.23 95.28 readyc 4751.21 2244.72 939.51 463.71 228.88 146.45 barotr 3449.31 1628.74 642.58 396.34 143.41 91.86 bclinc 1431.89 696.73 363.93 234.59 137.11 101.76 tracer 7439.98 3147.73 1822.47 1467.40 614.94 445.63 icesnow 31.54 17.17 8.61 4.80 2.85 2.37 convadj 620.61 370.03 193.95 90.87 42.50 29.56 I/O 150.52 128.62 134.42 132.80 152.85 157.15 Total 19553.98 9017.89 4559.86 3068.69 1464.76 1070.06

Figure Captions:

Figure R1. The wall clock time of a model day for the 10km version and a model month for the 100km version. The blue and orange bars are for the Fortran and the Fortran and C mixed version. These tests were conducted on a platform of Intel Xeon CPU (E5-2697A v4, 2.60GHz). We used 28 and 280 cores for the low and high resolution, respectively.

Figure R2 The seven core subroutines' time cost percentage for (a) 384 and (b) 9216 CPU cores. (c) the subroutines' time cost at different scales of LICOM3 (1/20°). These tests were conducted on a platform of Intel Xeon CPU (E5-2697A v4, 2.60GHz).

Figure R3 The ratio of the time cost of halo update and memory copy to the total time cost for three subroutines, "barotr" (green), "bclinc" (blue), and "tracer" (orange) in the HIP version LICOM for three scales (Unit: %). The numbers in the blankets are the time cost of the two processes (Unit: second).

Figure R4 Roofline model for AMD GPU and the performance of LICOM's main subroutines.

Interactive comment on Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2020-323, 2020.

Printer-friendly version

Discussion paper

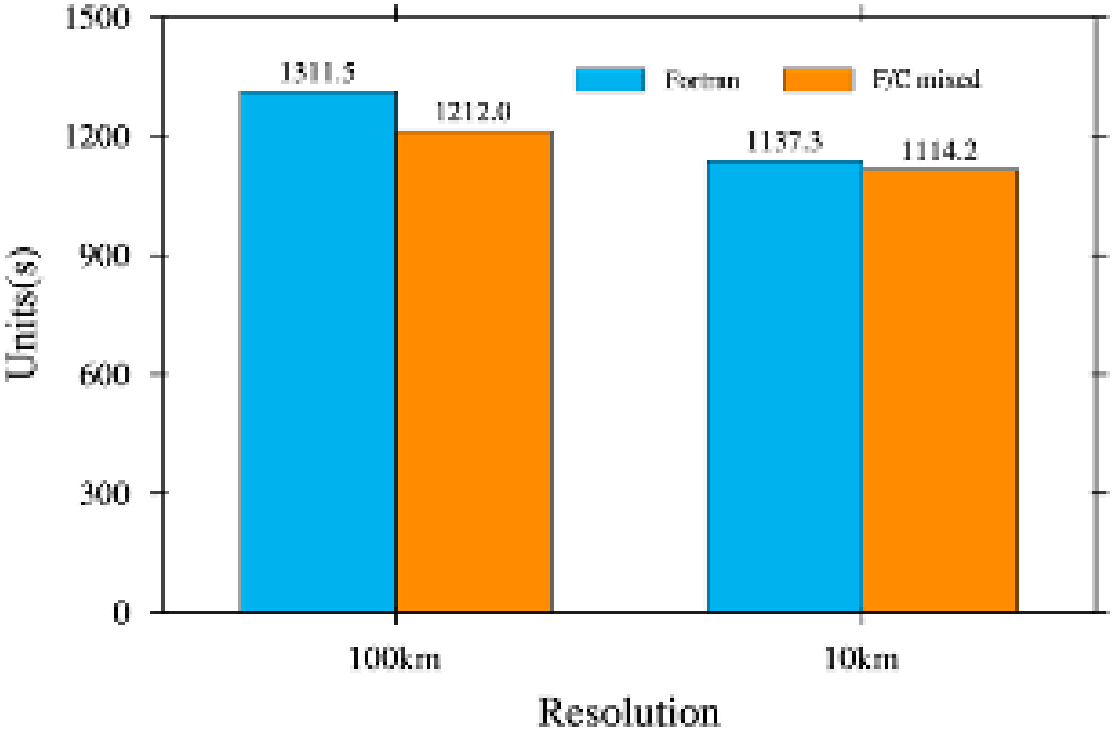

Fig. 1. The wall clock time of a model day for the 10km version and a model month for the 100km version. The blue and orange bars are for the Fortran and the Fortran and C mixed version.

Printer-friendly version

Discussion paper

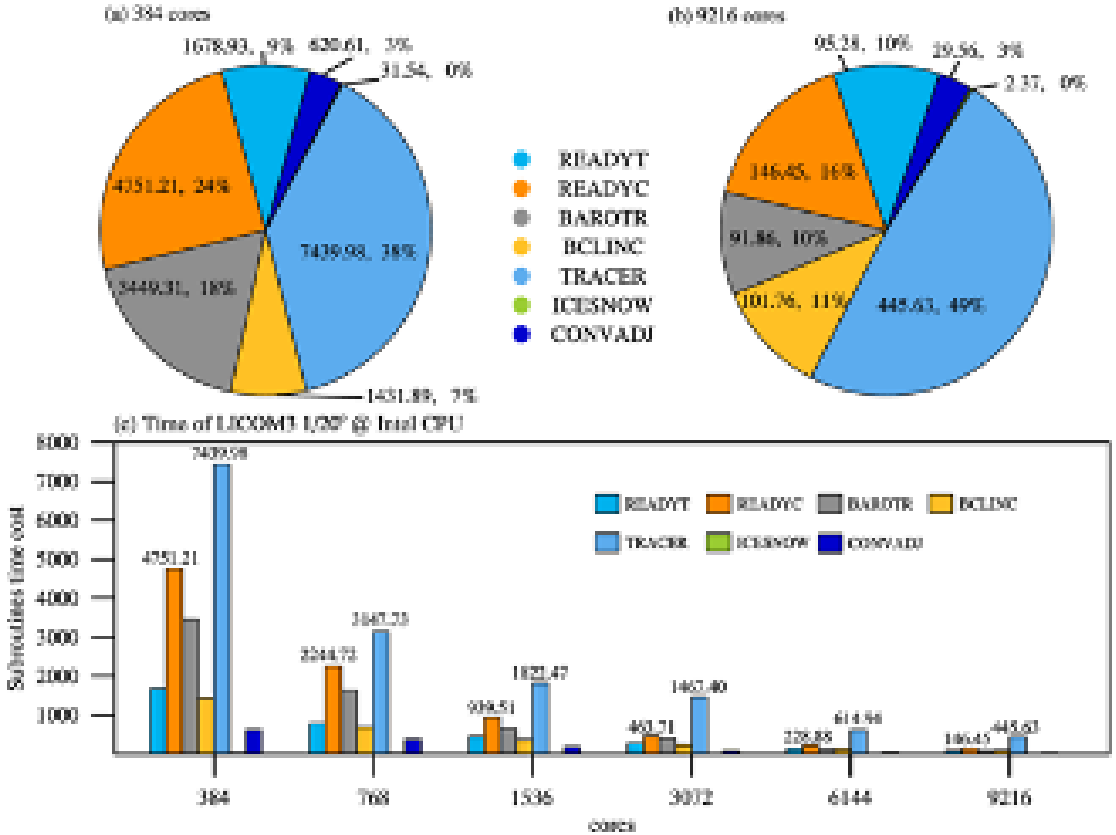

Fig. 2. The seven core subroutines' time cost percentage for (a) 384 and (b) 9216 CPU cores. (c) the subroutines' time cost at different scales of LICOM3 (1/20°).

Printer-friendly version

Discussion paper

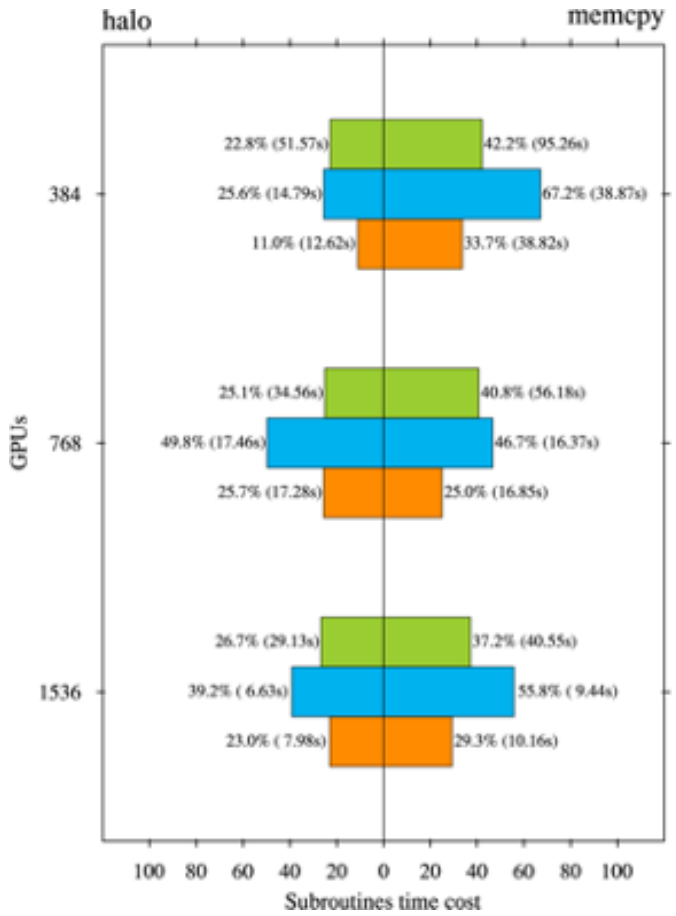

Fig. 3. The ratio of the time cost of halo update and memory copy to the total time cost for three subroutines, “barotr” (green), “bclinc” (blue), and “tracer” (orange) in the HIP version.

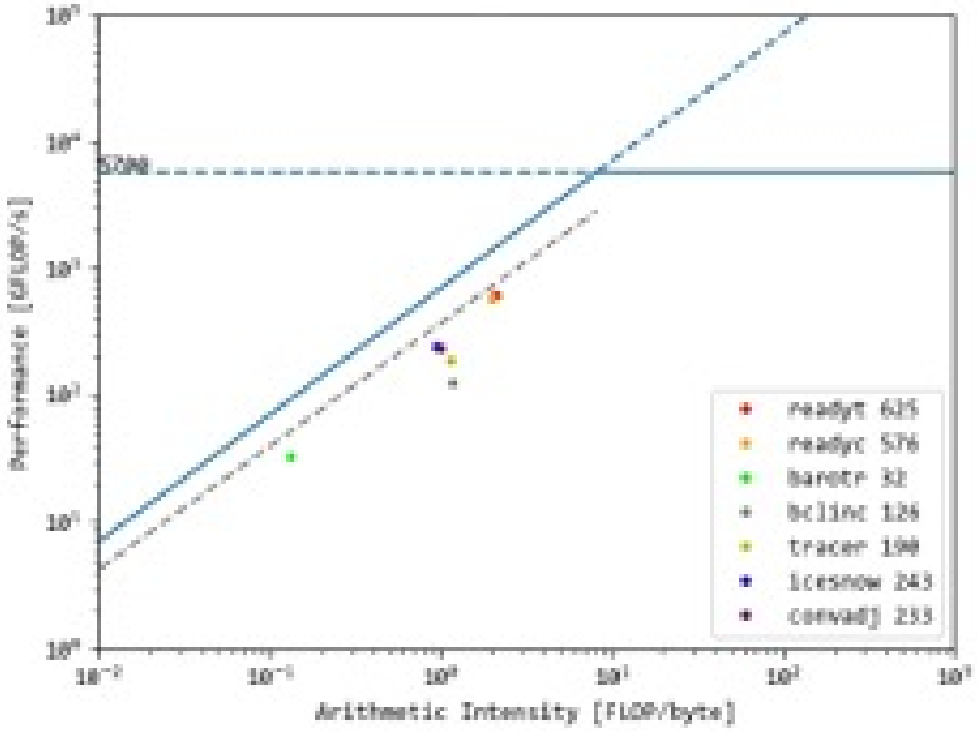

Fig. 4. Roofline model for AMD GPU and the performance of LICOM's main subroutines.