Accelerating an overhead-sensitive atmospheric model on GPUs using asynchronous execution and kernel fusion

Kazuya Yamazaki
Information Technology Center
The University of Tokyo
Kashiwa-shi, Chiba, Japan
ORCID 0000-0002-9215-7093

***Abstract*—Methods to mitigate the kernel launch overhead, one of drawbacks of GPUs, were implemented to an overhead-sensitive atmospheric model using OpenACC and CUDA and were evaluated. OpenACC enables kernels to run asynchronously in either one or multiple GPU queues. Moreover, CUDA allows different loops to be collocated in one kernel by branching operations based on block indices. While the default synchronous execution on A100 GPU lagged behind the A64FX CPU in strong scaling, the single-queue asynchronous execution reduced the total model runtime by 37%, and the kernel fusion of the core application component further accelerated the entire model by approximately 10%. In overhead-sensitive applications, the single-queue asynchronous execution is recommended because it can be easily implemented and maintained. If a small number of kernels are executed particularly frequently, it would be worth the effort to eliminate synchronizations and introduce CUDA Graphs, or bundle kernels using CUDA.**

I. Introduction

GPUs and similar accelerators are becoming dominant in supercomputers due to their high performances per cost or power. A wide range of applications, from traditional ones to the recently surging field of machine learning, are well-suited to the highly parallel nature of these accelerators.

However, GPUs do exhibit some weaknesses. For instance, GPUs typically execute applications as separate kernels invoked by host processors. Because it takes some time to launch kernels [1], applications with a particularly large number of iterations or timesteps may experience slowdowns due to kernel launch overheads. Another limitation of GPUs arises from their architecture, which is optimized for parallel processing. When an application includes tasks that cannot be massively parallelized, GPUs often underperform. Recently, products featuring tight integration between host CPUs and accelerators have emerged. In such products, it may be feasible to retain some subroutines on CPUs without incurring excessive wait times for host-device data transfers. However, even in such tightly coupled processors, kernel launch overheads can still slow down applications with a vast number of kernel launches.

This work is funded by JSPS KAKENHI (Grant Numbers JP20H00580, JP20K14517, JP20K21787, JP21H03447, JP22H00507, and JP23K11123) and JST CREST (JPMJCR21M2).

It is crucial to overcome such drawbacks of GPUs. By adapting as many HPC applications as possible to GPUs, resources can be more focused on GPU-equipped supercomputers, which benefits GPU-optimal applications as well. In pursuit of this goal, an overhead-sensitive atmospheric model was ported to GPUs and methods to mitigate overheads were evaluated in this study.

Multiple atmospheric models have been ported to GPUs. Some studies confined their focus to resource-intensive subroutines [2]–[7] due to the extensive codebase of an entire atmospheric model, while others managed to port the entire model [8]–[12].

Previous studies have addressed the overheads of GPU kernel launches and synchronizations. For iterative solvers, kernels can be radically asynchronized, which compromises the accuracy of individual iterations [13]. Even if more iterations are required for convergence, the speedup by asynchronous execution may outweigh the increase in iterations, significantly reducing the total time for convergence. Conversely, most operations in atmospheric codes need to be executed accurately to adhere to conservation laws. Another strategy called persistent kernel [14] involves running the main loop inside a huge CUDA [15] kernel, drastically reducing the number of kernel launches. In this approach, all subroutines inside the main loop must be ported to CUDA. However, if the application supports CPU machines as well, such a CUDA-based approach forces developers to maintain multiple sets of codes for CPUs and GPUs. Hence, a single set of directive-based codes is preferable to CUDA-based codes in large-scale applications supporting a wide range of processors. NVIDIA developed another framework called CUDA Graphs [16], with which a set of kernels can be bundled as a “graph” and reissued quickly. Kernel launches in existing codes can be captured and transformed to graphs objects, which have less overheads than normal kernels. Both OpenACC and CUDA kernels are supported by CUDA Graphs. However, the target code may need revisions because all operations in the captured code must be asynchronous according to the CUDA Graphs specification. Hence, it takes some effort to deploy CUDA Graphs to large-scale applications, necessitating quantitative

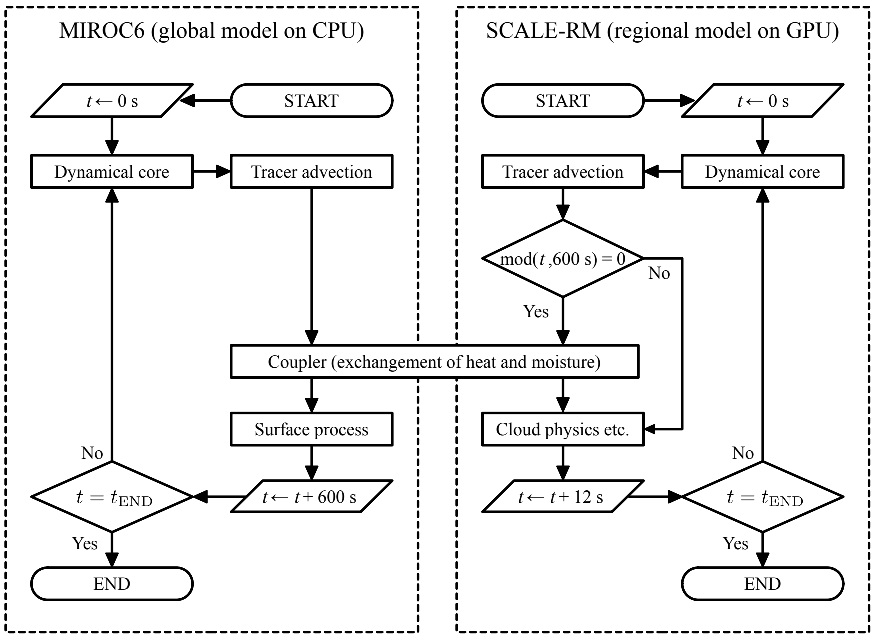

Fig. 1. Overview of the SP-MIROC model. t_{END} is the simulation period, which is 10 days in the whole-model evaluation in this study.

evaluation of performance benefits before adoption. Finally, asynchronous kernel execution supported by OpenACC [17] has been employed by atmospheric models [10], [12]. While this approach is expected to help hide launch overheads, its quantitative contribution to performance has been scarcely documented.

The purpose of this study is to provide developers of large applications with insights to mitigate kernel launch overheads. An atmospheric model, sensitive to kernel launch overheads, was ported to GPUs, with or without methods to mitigate kernel overheads. The performance benefits for both the entire application and a core component were quantitatively evaluated. Finally, overhead mitigation methods suitable for existing large-scale applications were discussed.

II. Background

The target application in this study is SP-MIROC [18], an atmospheric model for decadal global simulations. This model employs a strategy known as super-parameterization or multi-scale modeling framework [19]–[21]. In a super-parameterized model, two models with distinct horizontal resolutions are integrated to conduct a multi-scale simulation (Fig. 1). A low-resolution model encompasses the entire simulation domain, typically global, and simulates large-scale circulations. The other component, a high-resolution regional model, simulates small-scale activities such as convective clouds and turbulence, and conveys their large-scale impacts to the low-resolution component. The high-resolution component simulates a multitude of small two-dimensional domains, each coupled with a unique location of the low-resolution component. By simulating the regional domains in two dimensions rather than three, the total computational cost is significantly reduced compared to a monolithic high-resolution model covering the entire globe. Regional domains do not interact with each other and communicate with the large-scale component infrequently to exchange heat and moisture. This allows super-parameterized models to execute decadal global simulations within reasonable resource constraints. Furthermore, explicit simulation of cloud activities, performed by the high-resolution component, enhances the representation of impactful meteorological phenomena compared to traditional low-resolution climate models [12], [18], [22]. In summary, super-parameterized models are both lightweight enough for long-term simulations and capable of representing convection-sensitive phenomena more effectively than traditional low-resolution climate models.

SP-MIROC is composed of two primary components written in Fortran. The first is a global model called MIROC6 [23] used for long-term climate simulations [24]. The second is a regional model called SCALE-RM v5.3.6 [25]–[27] modified to perform two-dimensional simulations instead of three. In this study, SP-MIROC operates exclusively in a specific configuration employed by [18], despite its ability to run in various horizontal resolutions. The global component comprises 256×128 horizontal grids and 40 vertical layers. There are 2048 regional domains, each with 32 horizontal grids spaced every 2 km and 39 vertical layers. Each regional domain is coupled with a set of 4×4 columns in the global component, exchanging heat and moisture. Although coupling a regional domain to a single column of the global model is more straightforward, this “blockwise coupling” helps reduce the resource usage drastically with only minor impacts on scientific performances. The time step of the regional component is 12 s and the global-regional coupling is performed every 600 s. In an 16-GPU run, which is considered as the optimal configuration for SP-MIROC on A100 GPUs (see Section V-B), the coupling process accounted for 2.2% of the total simulation time.

Both SP-MIROC components possess a dynamical core that solves the primary equations of fluid dynamics. They also include subroutines for tracer advection and various subgrid parameterizations (Fig. 1). The dynamical core of SCALE-RM is executed most frequently per timestep because it is divided to three substeps, each of which features a 4th-order 4-step Runge-Kutta method invoking the actual dynamical core four times. It accounts for approximately 40% of the total runtime, while the remaining 60% is shared by many other kernels. When ported to GPUs, it contains 32 kernels and can be divided into 9 groups of data-independent kernels.

The hybrid nature of super-parameterization allows for long-term high-resolution simulations but also makes the application sensitive to overheads. The timestep of the high-resolution component must be small to satisfy the the Courant-Friedrichs-Lewy stability condition. In this work, the timestep of SCALE-RM is configured as 12 s. To simulate years of atmospheric flows per day, which is necessary for climate studies, each timestep computation must be as quick as 10 ms. Because more than 500 kernels are present in a timestep, kernels run only for an average of 20 µs, which is short enough to expose launch overheads.

III. GPU PORTING AND OPTIMIZATION

The SCALE-RM codes were ported to GPUs primarily using OpenACC directives, while the MIROC6 codes were left on CPUs due to the lightweight nature of the global component. Even though the mainstream SCALE-RM codes are being ported to GPU as well [7], the SP-MIROC version was ported independently of the main branch for further optimization. All array operations were ported to GPUs, eliminating host-device array transfer in the main loop. The SCALE-RM codes contain complex loops with long bodies, which could not be adequately parallelized due to a shortage of registers. To ensure sufficient parallelization, such loops were divided into simpler and more tightly nested ones. Although the SCALE-RM codes were modified to simulate two-dimensional fluid dynamics, kernels were implemented as collapsible three-dimensional loops handling all regional subdomains assigned to a GPU in parallel. In a 16-GPU configuration, each kernel handles 128 regional subdomains, and the typical CUDA grid size was 1000.

As will be demonstrated, the synchronous execution of GPU kernels, referred to as the “Sync” implementation in this study, results in a significant portion of the runtime consumed by kernel overheads. Therefore, the Sync approach, which is the default behavior in OpenACC and OpenMP target [28], is not suitable for SP-MIROC.

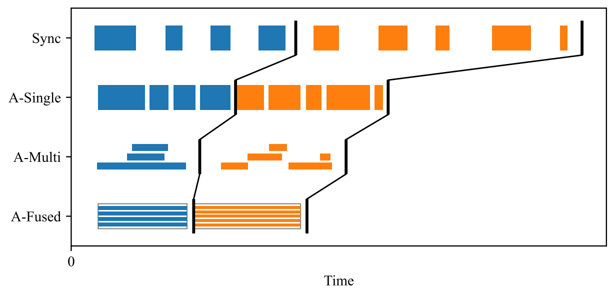

Fig. 2. Hiding kernel launch overheads using asynchronous execution. Rectangles represent individual kernel executions. Kernels are grouped by colors and black lines. Kernels in a group are data-independent, while kernels in different groups have data dependency. Kernels merged together in the A-Fused configuration are grouped by gray broken rectangles.
In the Sync configuration, large gaps are present between executions due to kernel overheads. Such gaps will be partially hidden in the A-Single approach. Multiple kernels are executed concurrently in A-Multi to hide overheads, while kernels are merged to fewer ones in the A-Fused implementation.

This study compares the performance of three methods designed to hide and/or reduce kernel launch overheads. The first method, A-Single, hides launch overheads by launching GPU kernels asynchronously in one GPU queue (Fig. 2) using the OpenACC’s `async(0)` clause. The second method, A-Multi, launches kernels asynchronously in multiple queues (Fig. 2) using the OpenACC `async` clause without an argument. The third method, A-Fused, merges data-independent OpenACC kernels to fewer CUDA kernels (Fig. 2).

In the A-Single approach, issues of data dependency do not arise as kernels are executed sequentially. Therefore, synchronization is only required before MPI communications and interactions with CPU codes. This makes it easy to implement and maintain A-Single, even for a large codeset. This approach necessitates language support for explicit queue assignment or an option to run kernels asynchronously with respect to CPU, but only in one GPU queue. While OpenACC’s `async(0)` clause realizes this, the `nowait` clause in OpenMP 5.2 [28] does not support such queue management. Hence, the A-Single approach is not suitable for existing versions of OpenMP target.

The A-Multi approach involves concurrent execution of multiple kernels. This method is more portable because it can be implemented in both OpenMP and OpenACC. Furthermore, the computation may be accelerated beyond that in A-Single because GPU resources unused by one kernel can be utilized by another. However, the concurrent kernel execution introduces issues of data dependency. Application developers must ensure that kernels are data independent or a synchronization directive is inserted to resolve the dependency. This forces developers to inspect all kernels carefully, which can be quite painful for a large application. Furthermore, any modification of the source code after the initial GPU porting prompts a reevaluation of the data dependency. Hence, the

A-Multi approach considerably reduces the maintainability of the application. Moreover, synchronizations required to resolve data dependencies have some overheads, which offset the speedup. Nevertheless, an A-Multi version of the SCALE-RM dynamical core was implemented in this study to quantitatively evaluate the performance benefit.

The A-Fused approach, while being the most costly in terms of implementation, is expected to provide the greatest performance benefit. CUDA kernels are parallelized at two levels: blocks and threads. While divergent branching between threads in a block results in a performance penalty, branching between blocks incurs less cost. Therefore, operations can be switched with respect to CUDA block numbers to run different tasks in one kernel. In this way, data-independent OpenACC kernels can be bundled to fewer CUDA kernels. In this work, no additional CUDA-specific optimization, such as the use of shared memory, was implemented to focus on the overheads. Single-queue asynchronous execution is suitable for this approach because most of the merged kernels are data dependent to each other. In this study, an A-Fused version of the SCALE-RM dynamical core was implemented using the single-queue approach. In general, the directive-based approach is preferred in atmospheric models for maintainability. However, using CUDA for the dynamical core may be justified if it constitutes only a small portion of the entire source code and a reasonable speedup is achieved.

Variants of A-Single and A-Fused with CUDA Graphs were also implemented only for the SCALE-RM dynamical core. Codes of the dynamical core were revised to comply with the constraints by CUDA Graphs; synchronizations were eliminated and `async` clauses were applied to all operations including `!$acc data` directives. Versions using CUDA Graphs will be referred to as A-Single-G and A-Fused-G in this paper.

IV. Procedures for the Performance Evaluation

The performance of the GPU-ported SP-MIROC was evaluated in two aspects: the dynamical core and the whole model. The dynamical core experiment focused on the performance on GPUs in various kernel launch configurations and problem sizes. In contrast, the strong scaling of time and energy to solution was compared in the whole-model experiment.

Four types of computers, equipped with either Fujitsu A64FX CPUs, NVIDIA A100 GPUs, NVIDIA H100 GPUs, or an NVIDIA GH200 processor, were used for the performance evaluation. One is the Odyssey subsystem of the Wisteria/BDEC-01 supercomputer equipped with Fugaku-compatible A64FX CPUs. Odyssey consists of FX1000 nodes, each of which contains one A64FX CPU. The A100 GPUs are from the Aquarius subsystem of Wisteria/BDEC-01. One GX2570 M6 node in the Aquarius cluster contains two sockets of Xeon 8360Y CPU and eight water-cooled A100 40GB GPUs. The H100 GPU and the GH200 superchip are contained in a DELL PowerEdgeXE8640 server and an ARS-111GL-NHR server, respectively, and are air-cooled. Codes for A64FX were compiled using Fujitsu Development Studio, while codes for NVIDIA GPUs were compiled using NVIDIA HPC SDK.

As stated earlier, the dynamical core experiment targets behaviors in various kernel launch configurations. Therefore, performances were evaluated only on GPUs (i.e. A100, H100, and GH200). To focus on the kernel launch overheads, subroutine calls were manually inlined in the dynamical core experiment. The timelines of kernel executions were retrieved using the Nsight Systems profiler. Preliminary measurements indicated that collection of kernel information using Nsight Systems results in a small overhead of a few microseconds per kernel, varying among launch strategies. In addition, the total elapsed time of the dynamical core shown by Nsight Systems was approximately 10% less than actually measured for unknown reasons. This makes it impossible to directly measure exact kernel overheads using the profiler. Hence, the runtimes of the entire dynamical core were measured without profiling kernels as well, and the timelines retrieved by Nsight Systems were rescaled to the runtimes without the profiler. The number of two-dimensional domains to simulate per GPU ranged from 1 to 1024. The dynamical core was run 101 times for each domain number and the timelines of the last 100 runs were averaged.

For comparison, an OpenMP-offloaded version of the dynamical core was tested on AMD MI210 and Intel Data Center GPU Max 1100. AMD ROCm version 6.0.2 and Intel oneAPI version 2024.2.0 were used to compile the codes. However, it failed to perform reasonably on these devices. Asynchronous offloading resulted in a segmentation fault on the Data Center GPU Max 1100. Even the smallest kernels took around 300 µs on MI210, which implies approximately 30 times the overhead compared to HIP codes on MI210 or OpenACC codes on NVIDIA GPUs. In contrast, the OpenMP version on A100 compiled with NVIDIA HPC SDK did not encounter such fatal problems, albeit twice to three times slower than the OpenACC version. Therefore, the OpenMP implementations by AMD and Intel used in this study were assumed to be premature for overhead-sensitive applications, and detailed evaluation on these accelerators was not conducted in this work. Performance improvements are expected as the implementations of OpenMP target by these vendors mature in the future.

The whole-model strong scaling experiment targets the full combination of the MIROC6 and SCALE-RM components in the standard SP-MIROC resolution. See [18] for meteorological details (e.g. parameterization schemes and boundary conditions). Simulations were performed on A64FX and A100 in Wisteria/BDEC-01, which supports the acquisition of power usages of the whole node. Ten-day simulations were performed and the time to solution was compared. The total power consumption of the nodes used was multiplied by the runtime to yield the energy to solution. Power consumed by the host modules (e.g. CPUs and main RAMs) as well as that of GPUs was included in the A100 runs. The program codes for the whole-model experiment is meant for long-term development and maintenance. Hence, the subroutine inlining employed in the dynamical core experiment was not performed in the

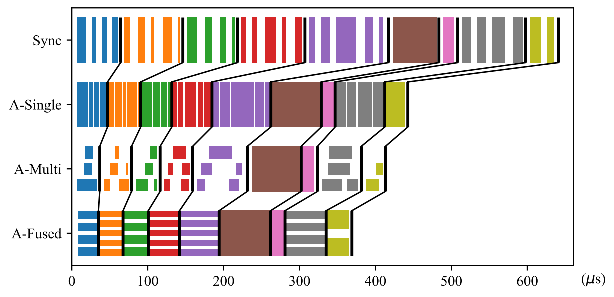

Fig. 3. Timelines of kernel executions in the 512-domain dynamical core on GH200. Rectangles represent individual kernel executions. Kernels are grouped by colors and black lines. Kernels in a group are data-independent, while kernels in different groups have data dependency. As stated in Section II, there are 32 kernels divided to 9 groups.

whole-model experiment. As a result, many kernels in the dynamical core were contained in separate subroutines.

Because there are 2048 regional domains in the standard SP-MIROC, the number of processes assigned to the SCALE-RM regional component was selected from divisors of 2048. In the A64FX CPU run, 512, 1024, or 2048 processes were assigned to SCALE-RM. Each process occupied one A64FX core. Sixteen processes were assigned to the global component when the regional component has 512 or 1024 processes. In contrast, 32 processes were allocated for the global component coupled to 2048 regional processes to catch up with the fast regional simulation. Hence, the total numbers of SP-MIROC processes were 528, 1040, or 2080, and were accommodated in 11, 22, or 44 A64FX CPUs. As for the A100 experiment, 4, 8, 16, or 32 GPUs were committed for the regional component. Each GPU was driven by one host process of the regional component. The number of processes in the global component was 16 for the 4-GPU or 8-GPU runs, and 32 for the 16-GPU and 32-GPU runs. All global processes ran on the host Xeon CPU along with regional host processes. As introduced earlier, one Aquarius GX2570 M6 node has 8 GPUs, which cannot be fully occupied by the 4-GPU run. Therefore, the energy usage of the 4-GPU run was derived as the total node energy subtracted by the idle energy drain of the unused 4 GPUs.

V. Experimental Results

A. Performance of the dynamical core

Fig. 3 presents timelines of kernels in the dynamical core simulating 512 domains on GH200. It is evident that the default synchronous execution (Sync) results in substantial gaps between kernel executions. The gaps shrink considerably in the A-Single configuration. The A-Multi and A-Fused configurations further accelerate the dynamical core by concurrently running multiple kernels.

On NVIDIA GPUs, asynchronous execution consistently reduces overheads in a wide range of problem sizes and GPU types (Fig. 4). Even when the problem size approached zero, the Sync version took 380–460 μs. As there are 32 kernels in the dynamical core, the kernel overhead is approximately

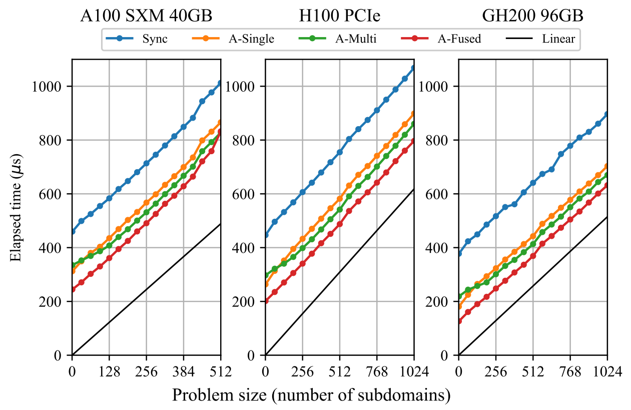

Fig. 4. Dependence of dynamical core runtimes on the number of domains to simulate. Note that the scale of the horizontal axis is different between A100 and the others to better illustrate overheads in near-zero problem sizes.

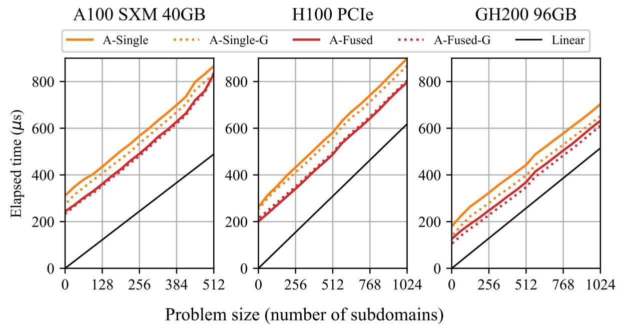

Fig. 5. Same as Fig. 4, but for comparison against implementations with CUDA Graphs. Dotted lines represent implementations with CUDA Graphs, while solid lines represent ones without CUDA Graphs.

12 μs. While the overhead was shorter in newer GPUs, the reduction was modest even in the tightly-coupled GH200 processor. In contrast, the A-Single and A-Multi runs finished in 180–330 μs when the problem size was close to zero. Hence, the overhead was reduced to approximately 8 μs per kernel. Although the A-Multi version outperformed A-Single for most of the problem sizes (Fig. 4), the difference was less than 1 μs per kernel. As the A-Multi approach considerably reduces the code maintainability, A-Single would be preferable to A-Multi for large-scale applications. The near-zero-size elapsed time was further reduced to 130–240 μs in the A-Fused run, despite the introduction of branching in the merged kernels. A-Fused consistently outperformed both A-Single and A-Multi. Hence, it may be worth the effort to implement A-Fused to a small amount of code invoking kernels particularly frequently.

The use of CUDA Graphs further accelerated the dynamical core (Fig. 5). However, the contributions of CUDA Graphs to A-Single and A-Fused were modest. On GH200, where the acceleration was the largest, the overhead of A-Single per kernel was reduced from 5.6 μs to 4.4 μs. In contrast, on an H100 PCIe, A-Fused-G was slightly slower than A-Fused, indicating that the benefit of CUDA Graphs may vary depending on environment. Most of the launch overheads were probably hidden in A-Single and A-Fused, leaving relatively a small room for improvement by CUDA Graphs. Therefore, in large-scale applications where maintainability is prioritized, CUDA Graphs would be suitable only for sets of kernels that are invoked particularly frequently.

B. Performance of the entire SP-MIROC model

Fig. 6 illustrates the strong scaling of the entire SP-MIROC. While the CPU version and the asynchronous versions scale well up to 44 CPUs or 16 GPUs, the Sync implementation failed to scale adequately beyond 8 GPUs. In the 16-GPU cofiguration, the A-Single asynchronous execution accelerated the simulation by 37% compared to the Sync run. The performance benefit of A-Single was larger in this whole-model experiment than in the dynamical-core-only expeirment (Section V-A) because subroutine inlining was not performed in the whole-model experiment, exposing subroutine overheads as well as kernel latencies in the Sync run. The kernel merger using CUDA (A-Fused) provided additional speedup of 10% compared to the A-Single run. As the dynamical core constitutes only a small portion of the entire SP-MIROC code, it would be justified to compromise the single code principle and employ CUDA for the dynamical core.

Asynchronous simulations on GPUs consumed much less energy than on A64FX (Fig. 7). The A-Single and A-Fused runs consumed around 2.5 MJ, which is approximately 40% less than the A64FX run. In the Sync run, overheads resulted in an increase of the energy consumption especially when larger number of GPUs were committed.

Considering both the time to solution and energy to solution, the optimal number of A100 GPUs in the standard SP-MIROC configuration is 16, which is too small for large-scale supercomputers. However, the GPU-ported SP-MIROC is expected to scale well to larger machines in other configurations. As shown in Fig. 1, the GPU-ported SCALE-RM component scarcely interacts with other processes. There are no communications among GPUs and the global-regional coupling is infrequent (600 s) compared to the SCALE-RM timestep of 12 s. Hence, the weak scaling of SP-MIROC is expected to be almost perfect, although actual tests could not be performed due to the small scale of the Aquarius system. Assuming that enough resources are available, there are larger-scale SP-MIROC applications that are meteorologically important. For example, refining the horizontal resolution of the global component helps better reproduce activities of an impactful meteorological phenomenon called mesoscale convective system (MCS) [29]. If the nominal grid spacing of the SP-MIROC global component were refined from 156 km to 19.5 km, the number of regional domains would increase 64 times. Changing from 4×4 blockwise coupling to 2×2 would further multiply the number of regional domains by 4. In such a large-scale configuration, 4096 A100 GPUs would be utilized efficiently, simulating years of global MCS activities per day.

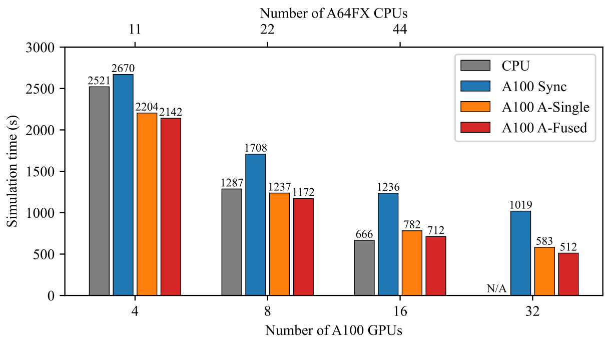

Fig. 6. Strong scaling of the full SP-MIROC model.

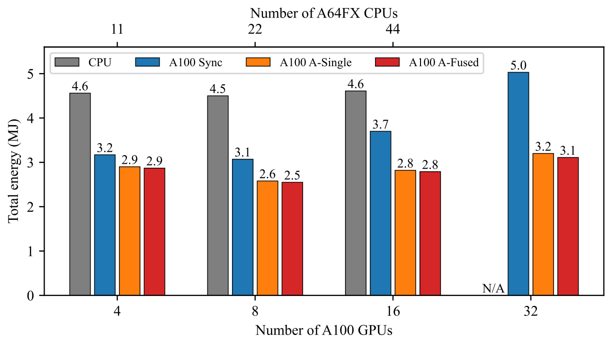

Fig. 7. Total energy usage of the full SP-MIROC model.

VI. Conclusion

As GPUs become dominant in top-tier supercomputers, it is increasingly important to adapt as many HPC applications as possible to GPUs. However, GPUs often exhibit significant overheads when launching kernels, which can slow down some HPC applications. This study focuses on porting and tuning an atmospheric simulation model, SP-MIROC, which is sensitive to kernel overheads, for GPU-equipped computers.

SP-MIROC, written in Fortran, consists of a low-resolution model running on CPU cores and a high-resolution model ported to GPUs using OpenACC. Kernels in the high-resolution component were launched asynchronously with respect to host processes, but in single queue per GPU, using the OpenACC `async(0)` clause. For the dynamical core subroutine, which is invoked the most frequently in SP-MIROC, multiple approaches were implemented. These include the default synchronous execution (Sync), single-queue asynchronous execution (A-Single), multi-queue asynchronous execution (A-Multi), and fusion of OpenACC kernels into fewer CUDA kernels (A-Fused). In addition, variants of A-Single and A-Fused with CUDA Graphs were implemented.

As shown in Section V, the A-Fused version was the fastest, closely followed by the A-Multi and A-Single approaches. In an run using 16 A100 GPUs, A-Single reduced the total model runtime by 37%, and A-Fused further accelerated the entire model by approximately 10%. The asynchronous approaches hide overheads of subroutine execution as well as kernel launches. While the use of CUDA Graphs further accelerated A-Single and A-Fused, its performance benefit was only 1.2 µs per kernel or less, likely because most of the overheads were already hidden in the A-Single and A-Fused implementations without CUDA Graphs.

Considering the fact that the kernel fusion requires target loops to be rewritten to CUDA, rather than inserting directives, it would be worth the effort only if a small amount of code accounts for a large portion of the runtime. Similarly, CUDA Graphs would be beneficial for sets of kernels that are invoked frequently. However, not all subroutines in applications may be suitable for CUDA Graphs because no synchronization is allowed in a graph. In contrast, single-queue asynchronous execution requires minimal additional effort for an application ported to GPUs using OpenACC. As no kernels are executed concurrently, no issues arise from data dependencies between kernels. In addition, one can synchronize the kernels at any points desired. Therefore, for overhead-sensitive applications, launching kernels asynchronously in a single queue is recommended.

Acknowledgment

The author thanks Dr. Yohei Miki and Prof. Toshihiro Hanawa for supports and comments to this work. Four anonymous reviewers helped improve the clarity, completeness, and correctness of this article through constructive comments. The A64FX and A100 runs were performed on the Wisteria/BDEC01 supercomputer operated by the University of Tokyo. The H100 runs were performed on the Mercury computer operated by the University of Tokyo.

References

[1] L. Zhang, M. Wahib, and S. Matsuoka, “Understanding the overheads of launching CUDA kernels,” in Proceedings of the International Conference on Parallel Processing, Poster Session 17, August 2019.

[2] J. Michalakes and M. Vachharajani, ”GPU acceleration of numerical weather prediction,” 2008 IEEE International Symposium on Parallel and Distributed Processing, pp. 1-7, April 2008.

[3] J. C. Linford, J. Michalakes, M. Vachharajani, and A. Sandu, ”Multi-core acceleration of chemical kinetics for simulation and prediction,” in SC’09: Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal., pp. 1-11, November 2009.

[4] I. Demeshko, S. Matsuoka, N. Maruyama, and H. Tomita, “Ultra-high resolution atmospheric global circulation model NICAM on graphics processing unit,” in Proceedings of the 2012 International Conference on Parallel and Distributed Processing Techniques and Applications, July 2012.

[5] C. Yang, W. Xue, H. Fu, et al., “10M-core scalable fully-implicit solver for nonhydrostatic atmospheric dynamics,” in SC’16: Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal., pp. 57-68, November 2016.

[6] H. Fu, J. Liao, N. Ding, et al., “Redesigning CAM-SE for peta-scale climate modeling performance and ultra-high resolution on sunway taihulight,” In: SC ’17: Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal., pp. 1-12, November 2017.

[7] S. Asai, S. Nishizawa, K. Yamazaki, S. A. Adachi, T. Yamaura, Y. Kawai, and Y. Sato, “Performance evaluation of the GPU-enabled weather model SCALE,” in Japan Geoscience Union Meeting 2024, AAS02-P08, May 2024.

[8] T. Shimokawabe, T. Aoki, C. Muroi, J. Ishida, K. Kawano, T. Endo et al., “An 80-fold speedup, 15.0 TFlops full GPU acceleration of non-hydrostatic weather model ASUCA production code,” In SC’10: Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal., pp. 1-11, November 2010.

[9] I. Demeshko, N. Maruyama, H. Tomita, and S. Matsuoka, “Multi-GPU implementation of the NICAM atmospheric model,” in: Euro-Par 2012: Parallel Processing Workshops. Euro-Par 2012. Lecture Notes in Computer Science, vol 7640, August 2012.

[10] H. Yashiro, M. Terai, R. Yoshida, S. Iga, K. Minami, and H. Tomita, “Performance analysis and optimization of Nonhydrostatic ICosahedral Atmospheric Model (NICAM) on the K Computer and TSUBAME2.5,” in Proceedings of the Platform for Advanced Scientific Computing Conference (PASC ’16), Article 3, pp. 1-8, June 2016.

[11] O. Fuhrer, T. Chadha, T. Hoefler, et al., “Near-global climate simulation at 1 km resolution: establishing a performance baseline on 4888 GPUs with COSMO 5.0,” Geosci. Model Dev., vol. 11(4), pp. 1665-1681, May 2018.

[12] M. R. Norman, D. C. Bader, C. Eldred, et al., “Unprecedented cloud resolution in a GPU-enabled full-physics atmospheric climate simulation on OLCF’s summit supercomputer,” Int. J. High Perform. Comput. Appl., vol. 36(1), pp. 93-105, July 2021.

[13] S. Venkatasubramanian and R. W. Vuduc, “Tuned and wildly asynchronous stencil kernels for hybrid CPU/GPU systems,” in Proceedings of the 23rd international conference on Supercomputing (ICS ’09), pp. 244-255, June 2009.

[14] L. Zhang, M. Wahib, P. Chen, J. Meng, X. Wang, T. Endo, and S. Matsuoka, “PERKS: a locality-optimized execution model for iterative memory-bound GPU applications,” in Proceedings of the 37th ACM International Conference on Supercomputing, pp. 167–179, June 2023.

[15] NVIDIA, “CUDA Fortran programming guide” [Online], available at https://docs.nvidia.com/hpc-sdk/pdf/hpc245cudaforug.pdf, May 2024.

[16] NVIDIA, “Getting Started with CUDA Graphs” [Online], available at https://developer.nvidia.com/blog/cuda-graphs/, August 2024.

[17] OpenACC-Standard.org, “The OpenACC application programming interface version 2.7” [Online], available at https://www.openacc.org/sites/default/files/inline-files/OpenACC.2.7.pdf, November 2018.

[18] K. Yamazaki and H. Miura, “ Reproducibility of equatorial Kelvin waves in a superparameterized MIROC: 1. Implementation and verification of blockwise-coupled SP-MIROC,” J. Adv. Model. Earth Syst., vol. 16, e2023MS003836, May 2024.

[19] W. W. Grabowski and P. K. Smolarkiewicz, “CRCP: a cloud resolving convection parameterization for modeling the tropical convecting atmosphere,” Physica D: Nonlinear Phenomena, vol. 133(1), pp. 171-178, September 1999.

[20] W. W. Grabowski, “Coupling cloud processes with the large-scale dynamics using the cloud-resolving convection parameterization (CRCP),” J. Atmos. Sci., vol. 58(9), pp. 978-997, May 2001.

[21] D. Randall, M. Khairoutdinov, A. Arakawa, and W. Grabowski, “Breaking the cloud parameterization deadlock,” Bull. Amer. Meteor. Soc., vol. 84(11), pp. 1547-1564, November 2003.

[22] K. Yamazaki and H. Miura, “Reproducibility of equatorial Kelvin waves in a super-parameterized MIROC: 2. Linear stability analysis of in-model Kelvin waves,” J. Adv. Model. Earth Syst., vol. 16, e2023MS003837, May 2024.

[23] H. Tatebe, T. Ogura, T. Nitta, Y. Komuro, K. Ogochi, T. Takemura, T., et al., “Description and basic evaluation of simulated mean state, internal variability, and climate sensitivity in MIROC6,” Geosci. Model Dev., vol. 12(7), pp. 2727-2765, July 2019.

[24] T. Kataoka, H. Tatebe, H. Koyama, T. Mochizuki, K. Ogochi, H. Naoe, et al., “Seasonal to decadal predictions with MIROC6: description and basic evaluation,” J. Adv. Model. Earth Syst., vol. 12, e2019MS002035, December 2020.

[25] S. Nishizawa, H. Yashiro, Y. Sato, Y. Miyamoto, and H. Tomita, “Influence of grid aspect ratio on planetary boundary layer turbulence in large-eddy simulations,” Geosci. Model Dev., vol. 8(10), pp 3393-3419, August 2015.

[26] S. Nishizawa, H. Tomita, and Team SCALE, “Detailed formulation of SCALE-RM” [Online], available at https://scale.riken.jp/archives/scale_rm_description-5.3.6.pdf, April 2020.

[27] Y. Sato, S. Nishizawa, H. Yashiro, Y. Miyamoto, Y. Kajikawa, and H. Tomita, “Impacts of cloud microphysics on trade wind cumulus: which cloud microphysics processes contribute to the diversity in a large eddy simulation?”, Prog. Earth Planet. Sci., vol. 2(1), 23, August 2015.

[28] OpenMP Architecture Review Board, “OpenMP application programming interface version 5.2” [Online], available at https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5-2.pdf, November 2021.

[29] G. Lin, C. R. Jones, L. R. Leung, Z. Feng, and M. Ovchinnikov, “Mesoscale convective systems in a superparameterized E3SM simulation at high resolution,” J. Adv. Model. Earth Syst., vol. 14(1), e2021MS002660, December 2021.