Evaluation of Programming Models and Performance for Stencil Computation on GPGPUs

Baodi Shan*
TotalEnergies EP Research & Technology US, LLC
Houston, Texas, USA
baodi.shan@stonybrook.edu

Mauricio Araya-Polo
TotalEnergies EP Research & Technology US, LLC
Houston, Texas, USA

***Abstract*—GPGPUs are widely used in high-performance computing. Therefore, it is crucial to experiment and discover how to better utilize their latest generations of relevant applications. In this paper, we introduce highly tuned stencil-based kernels for NVIDIA A100 and H100 (of a GH200) GPGPUs. Performance results yield useful insights into the behavior of this type of computation for these new accelerators. This knowledge can be leveraged by many scientific applications which involves stencil computations. Further, evaluation of three different programming models: CUDA, OpenACC, and OpenMP target offloading is conducted on aforementioned accelerators. We extensively study the performance and portability of various kernels under each programming model and provide corresponding optimization recommendations. Furthermore, we compare the performance of different programming models on the mentioned architectures. Up to 58% performance improvement was achieved against the previous GPGPU generation for a highly optimized kernel of the same class, and up to 42% for all classes. In terms of programming models, and keeping portability in mind, optimized OpenACC implementation outperforms OpenMP implementation by 33%. If portability is not a factor, the best CUDA implementation outperforms the optimized OpenACC one by $2.1\times$.**

***Index Terms*—stencil computation, GPGPU, CUDA, OpenMP**

I. Introduction

HPC-based on GPGPUs had mostly replaced CPU-based computing, thus becoming the primary source of computational power for large computing systems. Therefore, it is relevant to update and optimize workloads that rely on this kind of accelerators as they evolve. In this work's case, the target application is subsurface characterization ([1]), through wave equation solving, where the most computational demanding part is the stencil calculations. Research on stencil computing continuously produces technical advances ([2]–[5]) given its importance for many scientific and industrial applications([6]), from weather prediction ([7]) to earthquake modeling ([8]). We explore the potential impact of the new features and implementations of programming models such as CUDA, OpenMP and OpenACC by evaluating our stencil-based application on current GPGPUs. Then, we proposed optimizations for the stencil computation code and provided optimization suggestions for subsequent developers. We also proposed new optimization strategies at both the runtime and compilation levels for directive-based programming models. Lastly, we evaluated and analyzed the performance and changes of the three models on different generations of NVIDIA GPGPUs.

*This author is a Ph.D. candidate at Stony Brook University. This work was performed while the author was at TotalEnergies EP Research & Technology US, LLC.

II. Stencil Computation and Related Work

Stencil computations are at the core of many scientific applications. The one under analysis in this work is used to compute the differential operators required by Finite Difference scheme to solve the wave equation, in this case an acoustic isotropic approximation of it. The implementation followed in this work was introduced in [9] and optimized versions tailored for GPGPUs in presented in [10].

In this case, the spatial part of the wave equation is discretized using a 25-point stencil in 3D (8^{th} order in space), with eight points in each direction plus the centre point, therefore the patter has a characteristic 3D cross-shaped form.

Multiple works introduce implementations and optimization strategies for stencil computations, relevant to our work references are described as follows. The time skewing approach, as discussed in [11], [12], enhances the performance of stencil computations by augmenting data reuse and cache locality. Tiling is a popular technique where exchanging redundant computation along the boundaries of overlapped tiles decreases the required memory bandwidth [13], [14]. An alternative strategy for accelerating computation is split tiling, as described in [15]. Instead of using overlapped tiles, which can induce significant amounts of redundant computation, split tiling computes points in two distinct phases. Nguyen et al. [16] propose a 3.5D blocking algorithm that blends 2.5D spatial blocking with 1D temporal blocking. 2.5D spatial blocking involves blocking in a 2D plane and streaming along a third dimension to increase data reuse, storing active 2D planes in GPGPU shared memory.

III. Evaluation Setup

The primary testing platform is the NVIDIA GH200 Grace Hopper Superchip, in particular the H100 within. In the comparative evaluations, we also utilized computing nodes equipped with A100 and T4. The NVIDIA Hopper Architecture introduces a new memory indexing layer, termed "thread block cluster". The motivation of this feature is to facilitates data locality control at a granular level exceeding a solitary thread block on an SM.

IV. Exploration and Optimization of CUDA Implementations

Our code base encompasses a diverse array of kernel implementations, each of which has been extensively described in Sai et al. [10]. We selected the best kernels in-class from it: *gmem*, *smem*, *st_reg_fixed*, and *st_semi*. In our various implementations, we primarily include the following strategies. The first is "3D Blocking Using Global Memory Only" which is denoted as *gmem*; the second strategy is "3D Blocking Using Shared Memory," which is denoted as *smem*; the third strategy is "2.5D Streaming Fixed Registers," which we denote as *st_reg_fixed*; the fourth strategy is " 2.5D Streaming Semi-stencil," which was initially introduced for CPUs, which purpose is to increase memory reuse, it is denoted as *st_semi*.

TABLE I: Execution time for selective CUDA kernels on A100, H100 without and with thread block cluster. The numbers in angle brackets indicate the dimension of the thread block cluster. Figures in bold font represents the best performance in each column. The number in parentheses represents the performance improvement ratio of the optimal kernel in the column compared to the optimal kernel of A100.

Kernel	**A100 [s]**	**H100,without thread block cluster [s]**	**H100, with thread block cluster [s]**
gmem	27.058	11.710	<1,3,1>12.041 <1,1,3>11.315
smem	23.740	11.635	<1,3,1>12.300 <1,1,3>11.751
st_reg_fixed	19.908	**9.445(41.5%)**	**<1,2>9.434(41.6%)** <1,4>9.733 <2,1>9.455 <4,1>9.776
st_semi	**16.154**	9.654	<1,2>10.435 <1,4>10.757 <2,1>10.799 <4,1>11.585

The grid size used for all the experiments is 1024^3 and the number of time iterations is 1000. Table I shows execution time for different CUDA kernels, due to the requirement that the size of thread block cluster dimension must be a multiple of the GPU block in the corresponding dimension, we have chosen different dimension values for different CUDA kernels. Since our CUDA code minimizes data exchange between the shared memory of different thread blocks, with almost every thread block performing computations solely from its corresponding shared memory, the benefits of thread block clusters are negligible.

According to Table I, on H100, the optimal implementation is the *st_reg_fixed*, while on A100, the optimal one is the *st_semi*. In *st_semi*, the strategy's advantage is in optimizing the load-store balance, with data mainly in shared memory, benefiting from new GPU generation improvements in shared memory. Conversely, *st_reg_fixed* involves exchanges between registers, shared memory, and global memory. Given the enhancements in the H100 over A100 [17], significant improvements in shared memory size, read/write speed, and register size are observed. As a result, *st_reg_fixed* benefits more than *st_semi*.

V. Exploration and Optimization on OpenACC and OpenMP Target Offloading

We utilized a data domain decomposition approach in OpenACC [18] and OpenMP [19], [20] analogous to that of the CUDA model, where parallelism is implemented at the 3D outermost loop, slow dimension. Nonetheless, these implementations encounter memory bandwidth bottlenecks and latency visibility issues, which asynchronous computation can mitigate. In the OpenACC case, we enhanced the concurrency by using `async`, and further improve the code performance by limiting the number of CUDA streams and the number of registers at compile time. In the OpenMP target offloading [21]–[23] case, due to the different mechanisms of CUDA kernels, the original size of the target task is kept unchanged, but the inner loop and the boundary task execute asynchronously by `nowait`. Since the boundary task is significantly smaller than the inner task, the performance improvement yield by this optimization method is relatively small.

TABLE II: Execution time of OpenACC and OpenMP versions on A100 and H100 (Grid Size: 1024^3, 1000 iterations)

Programming Model	**Device**	**Execution Time(s)**
OpenACC	A100 H100	53.188 23.196
OpenACC-async	A100 H100	44.222 19.229
OpenMP	A100 H100	58.568 29.527

Table II shows execution time for OpenACC, OpenACC-*async*, OpenMP on A100 and H100. The grid size of the simulation is 1024^3 and it takes about 22.1 GB GPGPU memory. The number of iterations conducted are 1000.

VI. Comparison of Different Programming Models

We compared the performance differences of various programming models on different GPGPUs and made some interesting findings. As the generations of GPGPUs continue to update, the gap between the three models is gradually narrowing. Compared to the T4 GPGPU, the performance of the OpenACC version on the H100 has increased by $13.0\times$, while the performance of the OpenMP version has even achieved an $14.4\times$ increase; the optimized CUDA version, however, only saw a $9.4\times$ improvement with respect T4 reference.

VII. Conclusion

We evaluated 3D stencil kernels on latest GPGPUs, showing that the H100 GPGPU (of a GH200) is up to 33% improvement over prior GPGPUs. We introduced CUDA-based asynchronous execution for OpenACC and OpenMP, boosting performance by 30%. Our comparison across GPGPU generations reveals narrowing performance gaps among OpenMP, OpenACC, and CUDA programming models, highlighting their growing viability for portability.

References

[1] T. Tylor-Jones and L. Azevedo, *A Practical Guide to Seismic Reservoir Characterization*. Springer Nature, 2023.

[2] B. Sun, M. Li, H. Yang, J. Xu, Z. Luan, and D. Qian, "Adapting combined tiling to stencil optimizations on sunway processor," *CCF Transactions on High Performance Computing*, pp. 1–12, 2023.

[3] A. Denzler, G. F. Oliveira, N. Hajinazar, R. Bera, G. Singh, J. Gómez-Luna, and O. Mutlu, "Casper: Accelerating stencil computations using near-cache processing," *IEEE Access*, vol. 11, pp. 22 136–22 154, 2023.

[4] Q. Sun, Y. Liu, H. Yang, Z. Jiang, Z. Luan, and D. Qian, "Stencilmart: Predicting optimization selection for stencil computations across gpus," in *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2022, pp. 875–885.

[5] M. Jacquelin, M. Araya-Polo, and J. Meng, "Scalable distributed high-order stencil computations," in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2022, pp. 1–13.

[6] A. Dubey, "Stencils in scientific computations," in *Proceedings of the Second Workshop on Optimizing Stencil Computations*, ser. WOSC '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 57. [Online]. Available: https://doi.org/10.1145/2686745.2686756

[7] O. Fuhrer, C. Osuna, X. Lapillonne, T. Gysi, B. Cumming, M. Bianco, A. Arteaga, and T. Schulthess, "Towards a performance portable, architecture agnostic implementation strategy for weather and climate models," *Supercomputing Frontiers and Innovations: an International Journal*, vol. 1, no. 1, pp. 45–62, Apr. 2014.

[8] P. Moczo, J. Kristek, and M. Gális, *The Finite-Difference Modelling of Earthquake Motions: Waves and Ruptures*. Cambridge University Press, 2014.

[9] J. Meng, A. Atle, H. Calandra, and M. Araya-Polo, "Minimod: A finite difference solver for seismic modeling," 2020.

[10] R. Sai, J. Mellor-Crummey, X. Meng, M. Araya-Polo, and J. Meng, "Accelerating high-order stencils on gpus," in *2020 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, 2020, pp. 86–108.

[11] D. Wonnacott, "Using time skewing to eliminate idle time due to memory bandwidth and network limitations," in *Proceedings 14th International Parallel and Distributed Processing Symposium. IPDPS 2000*, 2000, pp. 171–180.

[12] ——, "Achieving scalable locality with time skewing," *International Journal of Parallel Programming*, vol. 30, 03 1999.

[13] S. Krishnamoorthy, M. Baskaran, U. Bondhugula, J. Ramanujam, A. Rountev, and P. Sadayappan, "Effective automatic parallelization of stencil computations," in *Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 235–244. [Online]. Available: https://doi.org/10.1145/1250734.1250761

[14] J. Holewinski, L.-N. Pouchet, and P. Sadayappan, "High-performance code generation for stencil computations on gpu architectures," in *Proceedings of the 26th ACM International Conference on Supercomputing*, ser. ICS '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 311–320. [Online]. Available: https://doi.org/10.1145/2304576.2304619

[15] T. Grosser, A. Cohen, P. H. J. Kelly, J. Ramanujam, P. Sadayappan, and S. Verdoolaege, "Split tiling for gpus: Automatic parallelization using trapezoidal tiles," in *Proceedings of the 6th Workshop on General Purpose Processor Using Graphics Processing Units*, ser. GPGPU-6. New York, NY, USA: Association for Computing Machinery, 2013, p. 24–31. [Online]. Available: https://doi.org/10.1145/2458523.2458526

[16] A. Nguyen, N. Satish, J. Chhugani, C. Kim, and P. Dubey, "3.5-d blocking optimization for stencil computations on modern cpus and gpus," in *SC'10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2010, pp. 1–13.

[17] NVIDIA, "Nvidia grace hopper superchip — nvidia," 2024, accessed: Mar 6, 2024. [Online]. Available: https://www.nvidia.com/en-us/data-center/grace-hopper-superchip/

[18] OpenACC-Standard.org, "Openacc," 2023, accessed: August 14, 2023. [Online]. Available: https://www.openacc.org/

[19] W. Lu, B. Shan, E. Raut, J. Meng, M. Araya-Polo, J. Doerfert, A. M. Malik, and B. Chapman, "Towards efficient remote openmp offloading," in *International Workshop on OpenMP*. Springer, 2022, pp. 17–31.

[20] B. Shan, M. Araya-Polo, A. M. Malik, and B. Chapman, "Mpi-based remote openmp offloading: A more efficient and easy-to-use implementation," in *Proceedings of the 14th International Workshop on Programming Models and Applications for Multicores and Manycores*, ser. PMAM'23, 2023, p. 50–59. [Online]. Available: https://doi.org/10.1145/3582514.3582519

[21] J. Huber, M. Cornelius, G. Georgakoudis, S. Tian, J. M. M. Diaz, K. Dinel, B. Chapman, and J. Doerfert, "Efficient execution of openmp on gpus," in *2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2022, pp. 41–52.

[22] S. Bak, C. Bertoni, S. Boehm, R. D. Budiardja, B. M. Chapman, J. Doerfert, M. Eisenbach, H. Finkel, O. R. Hernandez, J. Huber, S. Iwasaki, V. Kale, P. R. C. Kent, J. Kwack, M. Lin, P. Luszczek, Y. Luo, B. Pham, S. Pophale, K. Ravikumar, V. Sarkar, T. Scogland, S. Tian, and P. K. Yeung, "Openmp application experiences: Porting to accelerated nodes," *Parallel Comput.*, vol. 109, p. 102856, 2022. [Online]. Available: https://doi.org/10.1016/j.parco.2021.102856

[23] J. Doerfert, A. Patel, J. Huber, S. Tian, J. M. M. Diaz, B. Chapman, and G. Georgakoudis, "Co-designing an openmp gpu runtime and optimizations for near-zero overhead execution," in *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2022, pp. 504–514.