Impact of Memory Bandwidth on the Performance of Accelerators

Sambit Mishra
sambit98@tamu.edu
Department of Ocean Engineering,
Texas A&M University
College Station, Texas, USA

Dhruva K. Chakravorty
chakravorty@tamu.edu
High Performance Research
Computing, Texas A&M University
College Station, Texas, USA

Lisa M. Perez
perez@tamu.edu
High Performance Research
Computing, Texas A&M University
College Station, Texas, USA

Francis Dang
francis@tamu.edu
High Performance Research
Computing, Texas A&M University
College Station, Texas, USA

Honggao Liu
honggao@tamu.edu
High Performance Research
Computing, Texas A&M University
College Station, Texas, USA

Freddie David Witherden
fdw@tamu.edu
Department of Ocean Engineering,
Texas A&M University
College Station, Texas, USA

ABSTRACT

This study investigates the impact of memory bandwidth of accelerators on the performance of computational simulations, revealing the importance of bandwidth over computational power in scalable high-order numerical simulations. A detailed analysis performed on an NVIDIA H100 GPU and an Intel MAX 1100 GPU on the NSF ACES platform, demonstrates how matrix multiplication characteristics such as matrix size and sparsity influence the demand for memory bandwidth. Utilizing the open-source fluid flow solver PyFR for the study for its flexibility, efficiency, and alignment with expected performance, this work emphasizes the necessity for accelerator designs to prioritize memory bandwidth to enhance simulation efficiency, particularly in the case of workloads whose performance is bound by available memory bandwidth.

CCS CONCEPTS

• **Applied computing** → *Physics*; • **Hardware** → **Testing with distributed and parallel systems**; • **Computing methodologies** → **Simulation evaluation**; **Massively parallel and high-performance simulations**.

KEYWORDS

High-Performance Computing, Graphics Processing Unit, Memory Bandwidth, Discontinuous Spectral Element Methods, Matrix Multiplication Kernels, Performance Profiling, Roofline Model, Computational Fluid Dynamics, Simulation Scalability, Performance Portability, Science-Based Benchmarking

ACM Reference Format:
Sambit Mishra, Dhruva K. Chakravorty, Lisa M. Perez, Francis Dang, Honggao Liu, and Freddie David Witherden. 2024. Impact of Memory Bandwidth on the Performance of Accelerators. In *Practice and Experience in Advanced Research Computing (PEARC '24), July 21–25, 2024, Providence, RI, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3626203.3670540

PEARC '24, July 21–25, 2024, Providence, RI, USA

ACM ISBN 979-8-4007-0419-2/24/07
https://doi.org/10.1145/3626203.3670540

1 INTRODUCTION

Over the past decade, graphics processing units (GPUs) have emerged as a pivotal tool for high-performance computing (HPC). Its importance has been underscored by the increasing requirements of complexity, demands of computational tasks and shifting towards parallelism, particularly in fields such as computational fluid dynamics (CFD) [24], where the quest for higher simulation fidelity continues to push the boundaries of computational capabilities. This quest has also extended to developing different accelerator designs such as intelligence processing units [13] and cache architectures in GPUs such as those in the Intel MAX series Data Center GPUs [9]. At the time of writing, HPC focused GPUs are available from AMD, and NVIDIA, with Intel's recent entry marking a new era of innovation in the development of GPUs. With the ongoing developments in this field, evaluating and profiling their performance with industry-level software is essential.

For scientific applications, the performance of a GPU is typically evaluated based on three parameters: memory capacity, peak memory bandwidth, and peak floating point operations per second (FLOPs). Outside of linear algebra and machine learning applications, bandwidth is typically the limiting factor. As such when evaluating GPUs, it is important to consider (i) the *actual* amount of available bandwidth compared with the stated peak, (ii) the ease with which this bandwidth can be accessed by user code, and (iii) what tools are provided to measure bandwidth being achieved by kernels. In this paper, we consider these three questions within the context of CFD for two mainstream GPUs: NVIDIA H100 and Intel MAX 1100. Both the GPUs are available on the ACES testbed[14]. The performance characteristics of the Intel MAX GPUs which have not been as extensively tested as the GPUs provided by other vendors, further motivating this study.

Flux reconstruction (FR) is a numerical method that can be used for solving the compressible Navier–Stokes (NS) equations of fluid dynamics. Originally proposed by Huynh [6], FR belongs to the family of discontinuous spectral element methods, combining the superior accuracy of spectral methods with the geometric flexibility of finite volume methods. As a next-generation scheme, FR has several highly desirable attributes: (i) it is possible to select an arbitrary order of spatial accuracy, (ii) it operates naturally on mixed unstructured grids with curved boundaries, and (iii) when

paired with explicit time stepping exhibits a substantial degree of structured computation.

This third property makes FR an excellent candidate for modern computing architectures, including many-core CPUs with wide vector units, and GPUs. The arithmetic intensity of FR depends on a variety of factors. These include the *element type* and the chosen order of accuracy. In particular, on account of their tensor-product structure, hexahedral elements have a low arithmetic intensity whereas tetrahedral elements have an intensity that increases as a function of the order of accuracy. This variation makes FR a very good candidate for evaluating the bandwidth of modern accelerators.

In this manuscript, we use FR to evaluate GPUs from NVIDIA and Intel on the National Science Foundation (NSF) supported ACES testbed at Texas A&M University, with a particular focus on available memory bandwidth for numerical simulations. This will be accomplished through profiling a suite of FR simulations at orders two through six on both hexahedral and tetrahedral domains. The remainder of the paper is as follows: Section 2 outlines the FR formulation and the particular characteristics of the approach that render it suitable for evaluating the performance of accelerators across streaming architecture. Section 3 explains how the Python-based open-source CFD solver PyFR [25] is used to benchmark the NVIDIA H100 GPU and the Intel MAX 1100 GPU using the Taylor–Green vortex (TGV) breakdown as test case. Highlighting the performance contrast across the GPUs, this section further explains how the performance of TGV simulations was profiled. Section 4 discusses the key findings of the tests, exposes the cause of the observed performance contrast, and outlines potential areas of future work. Finally, conclusions are drawn in Section 5.

2 THEORY

We start with a brief overview of the salient computational aspects of the FR approach. Consider a non-linear conservation law of the form

$$\frac{\partial \mathbf{v}}{\partial t} = -\nabla \cdot \mathbf{f}(\mathbf{v}, \nabla \mathbf{v}), \tag{1}$$

where the solution $\mathbf{v}$ is computed across time t in an arbitrary domain Ω. We first construct a *mesh* of the domain with non-intersecting elements. An example of such a mesh can be seen in Section 2. Inside each element, the solution to the conservation law at an arbitrary time is represented by a polynomial. This polynomial is defined as a nodal form with the nodal points being known as *solution points*. The number of solution points within the element depends on the element type and the desired order of accuracy of the solution. To communicate the fluxes $\mathbf{f}$ across elements, *flux points* are introduced along the interfaces between elements. Knowledge of these fluxes can then be used to compute the first derivative of the solution with respect to time and hence obtain a semi-discretized form of a conservation law. A detailed description of the FR approach, along with its implementation is provided by Witherden et al. [25]. A depiction of solution and flux points inside a pair of quadrilateral elements can be seen in Fig. 1.

In FR, the operations that constitute the computation towards the right–hand–side (RHS) evaluation of Eq. (1) can be divided into two broad categories: point-wise nonlinear kernels and matrix-matrix multiplication kernels. The former encodes the physics of the problem. These kernels operate on either individual solution points or pairs of flux points. Solution point operations result in kernels with a contiguous (streaming) memory access pattern whereas those involving flux point pairs—on account of the unstructured nature of the mesh—have a partially indirect (gather/scatter) memory access pattern. Point-wise nonlinear kernels always have a lower arithmetic intensity when compared with matrix multiplication kernels[20]. The matrix multiplication kernels take the form of $C = AB + \beta C$ where $\beta \in \{0, 1\}$, A is a constant *operator matrix*, and B and C are state matrices whose column counts are proportional to the number of elements in the domain. Depending on the element type these A operator matrices can be sparse, and depending on the desired order of accuracy these matrices can vary in size. The execution sequence of these kernels and the characteristics of the involved matrix multiplications are known a priori. As a result, the performance profiles of solvers that use the FR approach may be mapped to their overall performance in a straightforward manner, which in turn is useful when isolating bottlenecks and improving solver performance across streaming architecture.

PyFR is a cross-platform CFD solver written primarily in Python to perform high-fidelity scale resolving simulations [25] with the FR approach. A unique aspect of PyFR is its performance portability across a range of hardware platforms. This is accomplished through extensive use of a domain-specific language that can translate—at runtime—functions into either C, CUDA, HIP, OpenCL, or Metal kernels. The cross-platform framework has been shown to scale on heterogeneous multi-node systems [26] and on composable cyberinfrastructure [12], and has been a finalist in the ACM Gordon Bell Prize for high-performance computing [23].

Within PyFR, a domain-specific language is used to handle all pointwise kernels. Matrix multiplication kernels are offloaded to either dense vendor BLAS libraries or the GiMMiK sparse kernel generator. When no BLAS libraries are available for the accelerator — such as in the case of Intel GPUs on the OpenCL API — GiMMiK kernels are used. In cases where multiple kernels are available, as is the case with the NVIDIA cuBLASLt library and GiMMiK, PyFR employs run-time auto-tuning to select the most efficient kernel. Equipped with these attributes, PyFR is well suited to leverage performance improvements of actively developed libraries across the accelerators it supports.

3 METHODOLOGY

The performance of NVIDIA H100 PCIe GPU (henceforth called H100 GPU) and Intel MAX 1100 PCIe GPU (henceforth called MAX GPU) was tested using TGV simulations with 64-bit floating point arithmetic computations (FP64) on PyFR v2.0.2, both GPUs available on the ACES [14]. The essential specifications of the accelerators are outlined in Table 1. GiMMiK v3.2.1 and cuBLASLt available in CUDA v12.3.0 was used throughout this work.

The study used PyFR to simulate the breakdown of the TGV on a cubic flow domain $\Omega = [0, 2\pi]^3$ with periodic boundary conditions along the three coordinate axes $\{x, y, z\}$[22]. The test case is known for its scalability across numerical schemes and accelerators [1, 12]. The NS equations are solved for the field variables $\boldsymbol{v} = \{p, u, v, w, \rho\}$

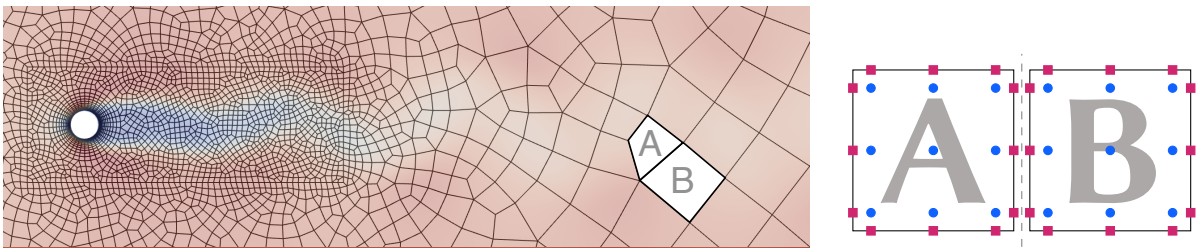


Figure 1: Example unstructured quadrilateral mesh along with the corresponding arrangement of solution points (blue circles) and flux points (orchid squares) inside a pair of neighboring elements. Adapted from Witherden [27].

Specification	NVIDIA H100 PCIe [5, 16]	Intel MAX 1100 PCIe [7–9]
API	CUDA	OpenCL
Device memory	80 GB HBM3	48 GB HBM2e
Peak FP64 Performance	51.2 TFLOPS	22.22 TFLOPS
Memory bandwidth	2039 GB/s	1228.8 GB/s
L2 cache	50 MB	108 MB
Kernel libraries	cuBLASLt, GiMMiK	GiMMiK

Table 1: Technical specifications of GPUs

at Reynolds number $Re = 1600$ with initial conditions

$$p = 1 + \frac{U^2}{16}\left(cos\left(\frac{2x}{L}\right) + cos\left(\frac{2y}{L}\right)\right)\left(cos\left(\frac{2z}{L}\right) + 2\right),$$
$$u = -Usin\left(\frac{x}{L}\right)cos\left(\frac{y}{L}\right)cos\left(\frac{z}{L}\right), v = -U\ cos\ \left(\frac{x}{L}\right)sin\left(\frac{y}{L}\right)cos\left(\frac{z}{L}\right),$$
$$w = 0, \rho = \frac{p}{RT}, \tag{2}$$

where $L = 2\pi$ is the length of the cubic domain, and R and T are constant fluid parameters. An image of the flow field is shown in

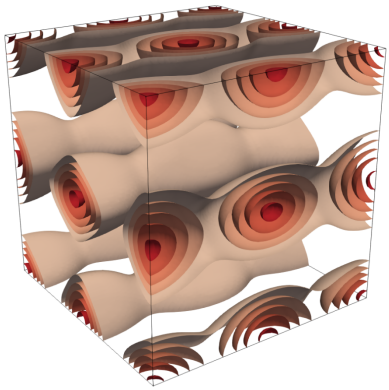


Figure 2: TGV breakdown simulation: Q-Criterion iso-contours colored by the magnitude of velocity at $t = 1$.

Fig. 2.

The matrix multiplication kernels employed for PyFR simulations depend on the number of solution points and flux points (see Fig. 1), which in turn depend on the polynomial order and element type of each element in the mesh. Considering the evaluation of a field variable at a solution point as a degree of freedom (DOF), the performance of the GPUs was measured in units of giga-degrees of freedom per second (GDoF/s). The overall performance of the TGV simulation for a given polynomial order and element type is measured as

$$P_{TGV} = P_{TGV}(N_e, e, O_{rk}, \mathcal{P}) = \frac{N_e \cdot DoF_p^e \cdot O_{rk} \cdot \mathcal{N}}{T_{\mathcal{N}}}, \tag{3}$$

where N_e is the total number of elements, O_{rk} is the Runge-Kutta order, and $T_{\mathcal{N}}$ is the wall-time taken for the simulation to run $\mathcal{N}$ timesteps. In the case of hexahedral elements and tetrahedral elements, the DOF of computations per element is given by

$$DoF_{\mathcal{P}}^{e=\text{HEX}} = 5(\mathcal{P}+1)^3, \tag{4}$$
$$DoF_{\mathcal{P}}^{e=\text{TET}} = 5\frac{(\mathcal{P}+1)(\mathcal{P}+2)(\mathcal{P}+3)}{6}, \tag{5}$$

where $\mathcal{P}$ is the polynomial order, $e = \text{HEX}$ and $e = \text{TET}$ are the hexahedral and tetrahedral element types respectively, and the factor 5 corresponds to the RHS computation of the five field variables at each solution point.

Finally, libraries essential to the execution of PyFR for NVIDIA GPUs — GCC v12.3.0, OpenMPI v5.0.1, OpenCL v3.0, and CUDA v12.3.0 — were built from source on each compute node containing H100 GPUs on ACES[14]. Libraries from Intel oneAPI toolkit were used to set up PyFR on the MAX 1100 GPUs. Python scripts used to create the meshes were provided [19]. The single-GPU scaling tests and profiling tests closely followed the procedure outlined by

Mishra et al. [12], while the scripts used for building and setting up PyFR along with its dependent libraries from the source are made available [21].

3.1 Single-GPU scaling tests

To measure the optimal loading conditions on the GPUs, TGV simulations were performed across polynomial orders $\mathcal{P} \in \{2, 3, 4, 5, 6\}$ with $e \in \{\text{TET}, \text{HEX}\}$ element types. To perform scaling tests with similar mesh sizes across all the polynomial orders and element types, six sets of meshes were created, ranging from around 3.2×10^5 DOF to 1.7×10^8 DOF.

PyFR benchmarks the performance of GiMMiK kernels against other available open-source and vendor-provided kernels prior to running simulations, such as cuBLASLt [15] for the CUDA platform. With the lack of performant BLAS libraries for Intel GPUs on OpenCL platform, GiMMiK library was used to create bespoke matrix multiplication kernels. A reliable comparison of the performance of PyFR on the GPUs warranted isolating the overall performance effects of using only GiMMiK kernels on the H100 GPUs. Towards this, additional functionality was added to the source code to selectively choose kernels from the GiMMiK library and cuBLASLt library. The average simulation performance across 1000 time steps is plotted in Fig. 3. PyFR simulations on the H100 GPUs performed equal to or better than those restricted to only GiMMiK kernels or only cuBLASLt kernels.

In the case of the MAX 1100 GPU, simulations run on the H100 GPUs were found to give a *not-a-number* (NaN) error. Upon further analysis, it was found that certain kernels performing robustly on the NVIDIA H100 failed on the MAX 1100 GPU[1]. A trial-and-error approach isolated the set of performant kernels on the MAX 1100 GPUs for all the simulations in this paper. A modified PyFR branch was set up to identify robust kernels, available as `pearc24revision` branch in the first author's Github repository [18].

3.2 Kernel profiling

The kernels were profiled to isolate the cause of the performance differences across the GPUs with the GiMMiK kernels. As the simulation performances tapered beyond 10^7 DOF in most of the cases, a mesh size of $\sim 7\times10^7$ DOF was considered for profiling the simulations. The overall performance of this set of simulations is provided in Table 2. TGV simulations were profiled across polynomial orders $\mathcal{P} \in \{2, 3, 4, 5, 6\}$, and for TET and HEX element types using the GiMMiK library.

NVIDIA NSight Compute 2023.3.0.0 available on the CUDA v12.3.0 toolkit was used to profile simulations on the H100 GPU. The profiler is designed to profile applications running on NVIDIA GPUs with a particular focus on the CUDA API. The profiler was set to profile the Python executible that ran PyFR, with the 'full' metrics option enabled. Each kernel call was executed about 500 times to obtain a detailed analysis of each kernel's execution, along with its corresponding roofline model.

To profile simulations on the MAX 1100 GPU, we utilized Intel V-Tune 2024.1 from the oneAPI toolkit available through ACES. 'GPU Compute/Media Hotspots' profiling was performed on the Python executable that ran PyFR, with 'Trace GPU programming APIs' enabled. To the authors' knowledge, the oneAPI toolkit could not provide adequate details to plot a roofline model for the kernels. Hence, PyFR's existing performance benchmarking routines for eligible matrix multiplication kernels were employed to obtain the roofline for each kernel. The run time (Δt) averaged over 1000 iterations and the number of non-zero entries ($\mathfrak{n}_A$) was obtained for the operator matrix A for each GiMMiK kernel. The performance of each kernel was obtained as

$$C_m = \frac{2 \times \mathfrak{n}_A \times \mathcal{R}s}{\Delta t}, \tag{6}$$

where the measured performance C_m equals the number of floating point operations performed per unit time. The arithmetic intensity ($\mathcal{I}$) is thus calculated as

$$\mathcal{I} = \frac{C_m}{BW}, \tag{7}$$

where the bandwidth BW was obtained from the Nsight Compute and Intel V-Tune profilers. The performance and arithmetic intensity for each kernel, along with the roofline model for both the GPUs are plotted in Fig. 4. The memory bandwidth reported from BabelSTREAM v4.0 [3] benchmarks were included on the plots.

4 DISCUSSION

PyFR measured the performance on the NVIDIA H100 and the Intel MAX 1100 GPUs with a series of simulations. While all GiMMiK kernels performed on the H100 GPUs, a subset of the GiMMiK kernels were robust on the Intel MAX 1100 GPUs. Observing a stark difference in performance across the GPUs, GiMMiK kernels were chosen for profiling the GPUs via PyFR.

For simulations performed on H100 GPUs, GiMMiK-enabled simulations outperformed cuBLASLt-enabled simulations for e = HEX simulations, while the reverse is true for e = TET simulations. A closer look at the kernels involved in the corresponding simulations revealed that the arithmetic intensity of e = HEX kernels was lower than that of e = TET kernels. Further, the performance difference between the GiMMiK and cuBLASLt kernels widened with polynomial order. This contrast is in line with the expected result of sparsity in the corresponding matrices and the relative inefficiency of the GiMMiK library in performing dense matrix operations. To ensure the best kernels are chosen with both of the available libraries, PyFR includes auto-tuning logic to compare their performance and select the best-performing kernel. No analogue of the cuBLASLt library is available specific to Intel GPUs with the OpenCL API, forcing PyFR to use GiMMiK kernels for all element types. MAX 1100 GPUs presented challenge in ensuring maximum possible performance was realised, as well as robust kernels were used to benchmark and profile the GPUs. The procedure followed to identify the invalid kernels is provided in Appendix B.3. Consequently, the performance of MAX 1100 GPUs was evaluated on basis of the subset of GiMMiK kernels that were found robust across all the simulations.

Upon profiling the GiMMiK kernels across the TGV simulations performed across polynomial orders and element types, the kernels are found to be limited by the available memory bandwidth on both the GPUs, as can be seen in Fig. 4. The performance of the kernels on the H100 GPUs was limited by the memory bandwidth of 2039 GB/s which was reported in the vendor's specification sheet

[1]This issue has been replicated via private correspondence with Intel.

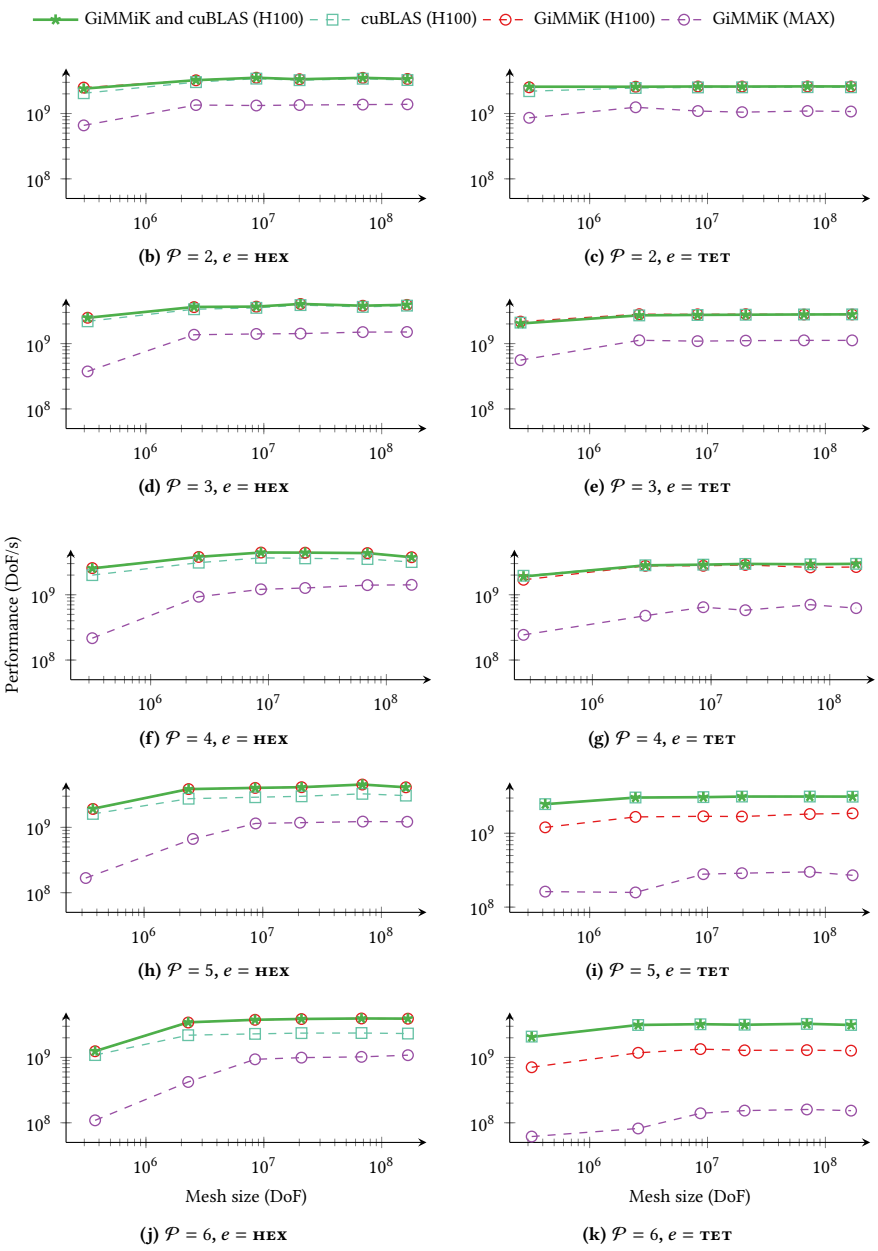


Figure 3: Performance of TGV simulations with various kernel configurations. Tests were performed on H100 and MAX GPUs for $e \in \{\text{TET}, \text{HEX}\}$ and $\mathcal{P} \in \{2, 3, 4, 5, 6\}$.

[16] and was in line with the roofline model from Nsight Compute. Interestingly, the performance on the Intel MAX GPUs was mostly limited by the memory bandwidth of 800 GB/s reported from the STREAM benchmark, and *not* the bandwidth of 1228.8 GB/s reported by the vendors [7]. The kernels generally performed worse in terms of both the arithmetic intensity as well as the performance. Finally, the roofline of high order TET elements exhibited significantly lower performance compared to the reported bandwidth limit and the FLOP limit, signaling a dire need for efficient dense matrix multiplication kernels.

While a clear performance contrast is observed across the GPUs, it may be noted that the unique architecture of the Intel MAX series GPUs in comparison with their competitors is potentially untapped. In particular, the substantial L2 cache on the Intel MAX 1100 GPU is 108 MiB, while on the NVIDIA H100, this is only 50 MB. This large cache size of the MAX series GPUs opens the possibility of

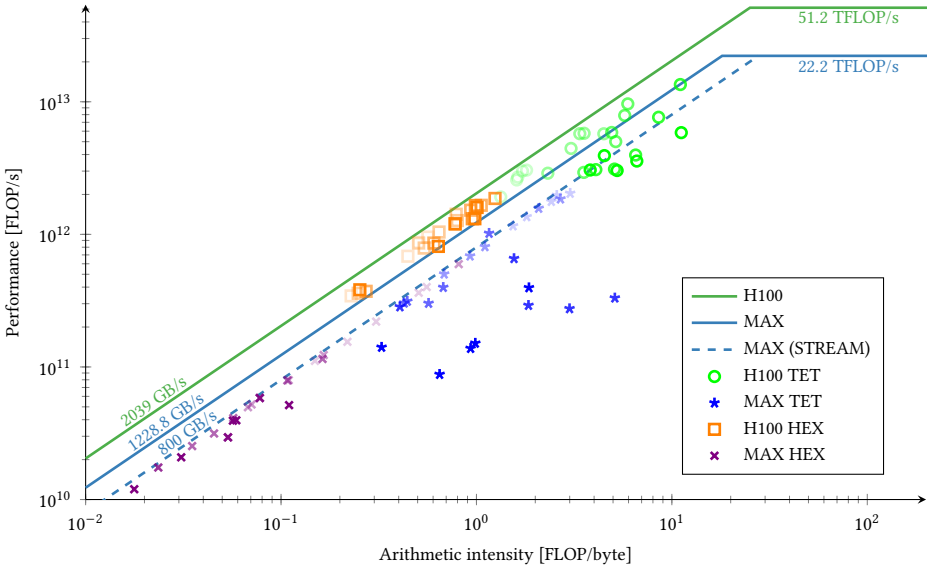


Figure 4: Roofline plot for kernels used in the TGV simulations on H100 and MAX GPUs. A higher intensity of scatter plot color represents a higher order polynomial.

incorporating cache-blocking strategies that are traditionally used on CPUs [2].

All the analysis was performed via PyFR, making the flow solver an ideal science-based tool for benchmarking the performance of accelerators on the cluster. The alignment of estimated performance with the actual performance as core to the development of the solver greatly helped boil down computations to the execution of kernels. Leveraging vendor-provided profilers' capabilities, the matrix multiplication kernels' performance allowed us to isolate the causes of performance discrepancy across the NVIDIA and Intel GPUs. Apart from highlighting the potential of the Intel GPUs and the limitations on bandwidth on the performance, this analysis also exposed potential bugs in compiling and running the code, further bolstering the potential of PyFR as a benchmarking tool for stress-testing the cluster. Finally, this in-depth analysis of Intel MAX 1100 GPUs is pivotal to the ongoing developments on the ACES testbed with cutting-edge resources such as Intelligence Processing Units[11], container adoption[10] and workforce development[4].

5 CONCLUSIONS

This work investigated the performance of NVIDIA H100 and Intel MAX 1100 GPUs using the open-source fluid flow solver PyFR on the ACES testbed. Single-GPU scaling tests were performed on both GPUs across different orders of accuracy and element types, which correspond to performing matrix multiplications with differing matrix sizes and sparsities. Upon observing the consistently better performance of the H100 GPU over MAX 1100 GPU across various configurations of the Taylor–Green Vortex simulations, the performance of GiMMik kernels on the H100 GPU and the MAX GPU were profiled with NVIDIA NSight Compute and Intel V-Tune respectively. The performance of kernels observed on the roofline plots exposed the strong relation between the overall performance of the simulations with the bandwidth-bound nature of the kernels. Finally, the untapped potential of the unique architecture of the Intel MAX series GPUs was discussed, and potential directions for improving the performance of the MAX 1100 GPUs were outlined. The findings in this work inform the researchers of the expected performance of the NVIDIA H100 and the Intel MAX 1100 GPUs available on the ACES facility and promote the development of OpenCL libraries for the Intel GPUs. The studies showed that the NVIDIA H100 GPUs performed better than the Intel MAX 1100 GPUs by a factor of up to 6× for dense matrix multiplications, and up to 3.7× for sparse matrix multiplications part of hexahedral and tetrahedral element meshes, owing to the bandwidth-bound computations performed by PyFR and the lack of efficient robust kernels for the Intel MAX 1100 GPUs.

ACKNOWLEDGMENTS

All simulations performed in this paper used the Accelerating Computing for Emerging Sciences (ACES) cyberinfrastructure testbed hosted by Texas A&M University with the support of NSF award numbers 2112356 and 1925764. SM and FDW were partially supported by the Air Force Office of Scientific Research via grants FA9550-23-1-0232 ("Enabling next-generation heterogeneous computing for massively parallel high-order compressible CFD").

REFERENCES

[1] Abouelmagd Abdelsamie, Ghislain Lartigue, Christos E. Frouzakis, and Dominique Thévenin. 2021. The Taylor–Green vortex as a benchmark for high-fidelity combustion simulations using low-Mach solvers. *Computers and Fluids* 223 (June 2021), 104935. https://doi.org/10.1016/j.compfluid.2021.104935

[2] Semih Akkurt, Freddie Witherden, and Peter Vincent. 2022. Cache blocking strategies applied to flux reconstruction. *Computer Physics Communications* 271 (Feb. 2022), 108193. https://doi.org/10.1016/j.cpc.2021.108193

[3] BabelSTREAM for benchmarking memory bandwidth 2024. . Retrieved March 8, 2024 from https://github.com/uob-hpc/babelstream

[4] Wesley A. Brashear, Lisa M. Perez, Elizabeth Leake, Sandra B. Nite, Marinus Pennings, Sheri Stebenne, Honggao Liu, and Dhruva K. Chakravorty. 2024. Cultivating Cyberinfrastructure Careers through Student Engagement at Texas A&M University High Performance Research Computing.. In *Practice and Experience in Advanced Research Computing (PEARC '24)*. ACM.
[5] Jack Choquette. 2023. NVIDIA Hopper H100 GPU: Scaling Performance. *IEEE Micro* 43, 3 (May 2023), 9–17. https://doi.org/10.1109/mm.2023.3256796
[6] H. T. Huynh. 2007. A Flux Reconstruction Approach to High-Order Schemes Including Discontinuous Galerkin Methods. American Institute of Aeronautics and Astronautics. https://doi.org/10.2514/6.2007-4079
[7] Intel Data Center GPU MAX 1100 Specifications 2024. . Retrieved Feb 22, 2024 from https://www.intel.com/content/www/us/en/products/sku/232876/intel-data-center-gpu-max-1100/specifications.html
[8] Intel Data Center GPU MAX Series 2024. . Retrieved March 8, 2024 from https://www.intel.com/content/www/us/en/developer/articles/technical/intel-data-center-gpu-max-series-overview.html
[9] Intel Data Center GPU MAX Series Technical Overview 2024. . Retrieved March 5, 2024 from https://www.intel.com/content/www/us/en/products/details/discrete-gpus/data-center-gpu/max-series.html
[10] Richard Lawrence, Dhruva K. Chakravorty, Lisa M. Perez, Wesley A. Brashear, Zhenhua He, and Joshua Winchell. 2024. Container Adoption in Campus High Performance Computing.. In *Practice and Experience in Advanced Research Computing (PEARC '24)*. ACM.
[11] Hieu T. Le, Zhenhua He, Mai Le, Dhruva K. Chakravorty, Akhil Chilumuru, Yan Yao, and Jiefu Chen. 2024. Performance Benchmarking and Lessons Learned from Porting AI/ML Workloads to Intelligence Processing Units. In *Practice and Experience in Advanced Research Computing (PEARC '24)*. ACM.
[12] Sambit Mishra, Freddie Witherden, Dhruva Chakravorty, Lisa Perez, and Francis Dang. 2023. Scaling Study of Flow Simulations on Composable Cyberinfrastructure. In *Practice and Experience in Advanced Research Computing (PEARC '23)*. ACM. https://doi.org/10.1145/3569951.3597565
[13] Abhinand Nasari, Lujun Zhai, Zhenhua He, Hieu Le, Suxia Cui, Dhruva Chakravorty, Jian Tao, and Honggao Liu. 2023. Porting AI/ML Models to Intelligence Processing Units (IPUs). In *Practice and Experience in Advanced Research Computing (PEARC '23)*. ACM. https://doi.org/10.1145/3569951.3603632
[14] NSF Category II: ACES - Accelerating Computing for Emerging Sciences 2024. . Retrieved March 6, 2024 from https://hprc.tamu.edu/aces/
[15] NVIDIA cuBLASLt library user guide 2024. . Retrieved Feb 24, 2024 from https://docs.nvidia.com/cuda/cublas/index.html#using-the-cublaslt-api
[16] NVIDIA H100 Specifications 2024. . Retrieved Feb 16, 2024 from https://resources.nvidia.com/en-us-tensor-core
[17] Paper data and results 2024. . Retrieved June 7, 2024 from https://github.com/sambitmishra98/GiMMiK-profiling-on-GPU.git
[18] PyFR branch with functionality to choose specific kernels 2024. . Retrieved Jun 12, 2024 from https://github.com/sambitmishra98/PyFR.git
[19] Python scripts to create GMSH 2024. . Retrieved March 7, 2024 from https://github.com/WillTrojak/basic_gmsh.git
[20] J. Romero, J. Crabill, J.E. Watkins, F.D. Witherden, and A. Jameson. 2020. ZEFR: A GPU-accelerated high-order solver for compressible viscous flows using the flux reconstruction method. *Computer Physics Communications* 250 (May 2020), 107169. https://doi.org/10.1016/j.cpc.2020.107169
[21] Single-GPU test scripts 2024. . Retrieved March 7, 2024 from https://github.com/sambitmishra98/benchmark
[22] Geoffrey Ingram Taylor and Albert Edward Green. 1937. *Proceedings of the Royal Society of London. Series A - Mathematical and Physical Sciences* 158, 895 (Feb. 1937), 499–521. https://doi.org/10.1098/rspa.1937.0036
[23] P. Vincent, F. Witherden, B. Vermeire, J. Park, and A. Iyer. 2016. Towards Green Aviation with Python at Petascale. In *SC16: International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE Computer Society, Los Alamitos, CA, USA, 1–11. https://doi.org/10.1109/SC.2016.1
[24] Z.J. Wang, Krzysztof Fidkowski, Rémi Abgrall, Francesco Bassi, Doru Caraeni, Andrew Cary, Herman Deconinck, Ralf Hartmann, Koen Hillewaert, H.T. Huynh, Norbert Kroll, Georg May, Per-Olof Persson, Bram van Leer, and Miguel Visbal. 2013. High-order CFD methods: current status and perspective. *International Journal for Numerical Methods in Fluids* 72, 8 (Jan. 2013), 811–845. https://doi.org/10.1002/fld.3767
[25] F.D. Witherden, A.M. Farrington, and P.E. Vincent. 2014. PyFR: An open source framework for solving advection–diffusion type problems on streaming architectures using the flux reconstruction approach. *Computer Physics Communications* 185, 11 (Nov. 2014), 3028–3040. https://doi.org/10.1016/j.cpc.2014.07.011
[26] F.D. Witherden, B.C. Vermeire, and P.E. Vincent. 2015. Heterogeneous computing on mixed unstructured grids with PyFR. *Computers and Fluids* 120 (Oct. 2015), 173–186. https://doi.org/10.1016/j.compfluid.2015.07.016
[27] Freddie D. Witherden. 2021. Python at Petascale With PyFR or: How I Learned to Stop Worrying and Love the Snake. *Computing in Science and Engineering* 23, 4 (July 2021), 29–37. https://doi.org/10.1109/mcse.2021.3080126

A KERNEL PERFORMANCE, PROFILE AND BENCHMARK DATA

The performance of simulations using the mesh set of mesh sizes $\sim 7 \times 10^7 DoF$ is given in Table 2.

B OBSERVATIONS ON INTEL MAX 1100 GPU

The setup of PyFR on the Intel MAX 1100 GPUs was not as straightforward as that on the NVIDIA H100 GPUs. Significant observations upon working with the MAX 1100 GPUs are noted in this section, while the rest of the observations are mentioned in [17].

B.1 Intel MPI and OpenCL installation on compute node

The Intel oneAPI toolkit was used to set up PyFR on the cluster.

Following issues around setting up MPI and OpenCL libraries on the Intel GPUs on ACES, all libraries were installed from the source, on the compute nodes containing the MAX GPUs. Build instructions to set up the libraries and results are available at the first author's Github repository Paper data and results [17], Single-GPU test scripts [21].

B.2 Strong-scaling test on MAX GPUs

To evaluate the potential benefits of performing single-GPU scaling tests and kernel profiling studies on the Intel MAX 1100 GPUs, strong scaling tests were performed on the MAX GPUs on ACES. ACES is set to host 120 Intel MAX 1100 GPUs upon its deployment. Four MAX GPUs are connected to a PCIe switch chip, which, in turn, is connected to the CPU. The nodes were connected across the NDR Infiniband fabric with an inter-node bandwidth of at least 200 GB/s. The libraries were set up on a compute node containing the MAX GPUs (see Appendix B.1), and multi-node multi-GPU scaling tests successfully ran on the cluster with OpenCL backend. A TGV simulation was performed with $e = \text{HEX}$ and $\mathcal{P} = 3$, and the generated meshes were partitioned to distribute workload across the GPUs with METIS v5.2.0. It may be noted that despite our use of a subset of kernels found to perform robustly on a single Intel MAX 1100 GPU, the strong-scaling tests found simulations to result in a NaN error beyond 32 GPUs.

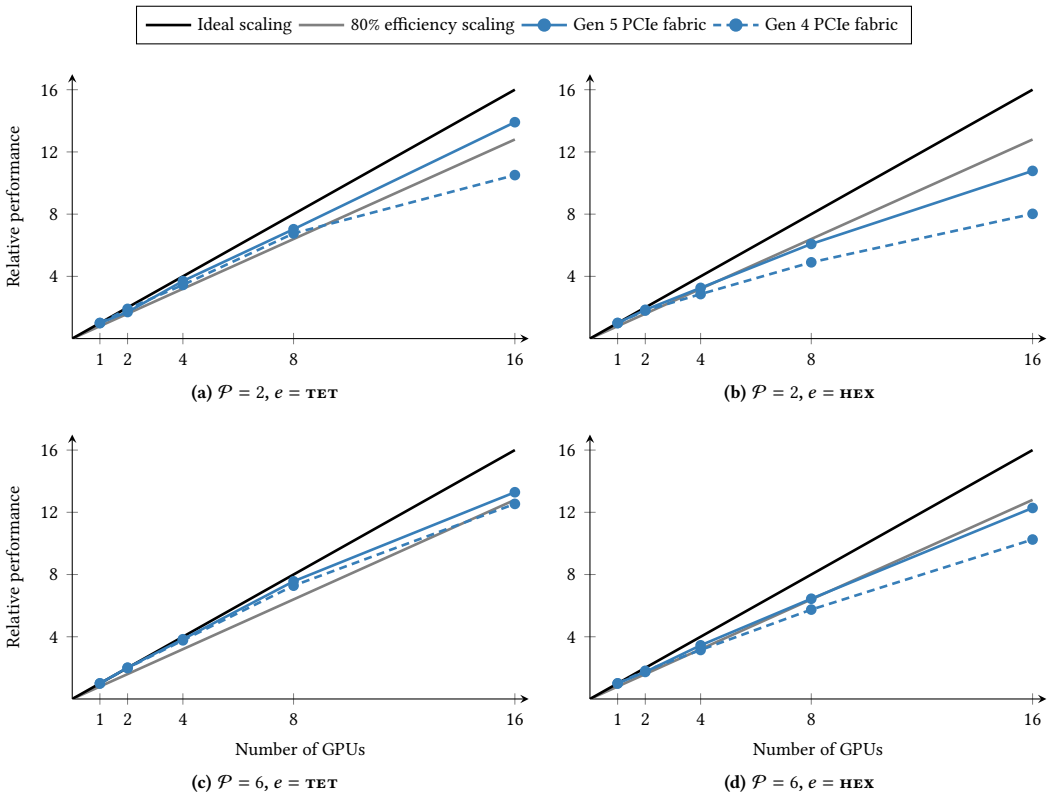
B.3 Isolation of faulty kernels

Simulations failed in a probabilistic manner, with the chance of a NaN increasing with simulation mesh size and polynomial order. Upon observing the NaN error, each matrix multiplication kernel was isolated and tested iteratively for large matrix sizes. For matrix multiplications of the form $C = AB$ where A performed in PyFR (see Section 2) when matrix C was expected to be a matrix with unit entries, the authors found that a few of the matrix entries deviated from 1 every few iterations, particularly for large matrices. Since this issue is only isolated as a result of the physics of the simulation failing, it is possible that many simulations would result in invalid physics but not deviate from the valid solution enough to cause the NaN error.

In addition, a few simulations resulted in a page fault on the nodes with the MAX 1100 GPUs. This warning did not seem to visibly affect the progress of the simulation every time it was observed,

Element type	Order	Mesh size [MDoF]	Performance [GDoF/s] H100 GiMMiK	H100 GiMMiK+cuBLAS	MAX GiMMiK
TET	2	70	2.596	2.602	1.092
	3	65	2.788	2.791	1.120
	4	68	2.621	2.935	0.703
	5	73	1.821	3.140	0.300
	6	69	1.294	3.276	0.160
HEX	2	69	3.513	3.517	1.369
	3	69	3.793	3.801	1.499
	4	69	4.342	4.344	1.403
	5	69	4.523	4.529	1.227
	6	67	3.950	3.951	1.020

Table 2: Comparison of the overall performance of PyFR simulations for the mesh set used to profile the GPUs.



(a) $\mathcal{P} = 2$, $e =$ TET **(b)** $\mathcal{P} = 2$, $e =$ HEX

(c) $\mathcal{P} = 6$, $e =$ TET **(d)** $\mathcal{P} = 6$, $e =$ HEX

Figure 5: Strong-scaling test performed with Intel MAX 1100 GPUs across multiple nodes on ACES, with four GPUs connected to each node on the composable Gen 4 and Gen 5 PCIe fabric. The performance was normalized with respect to corresponding simulations performed on the four meshes in the largest mesh set tested for the paper, i.e. $1.7 \times 10^8 DoF$.

but may be a warning for the error observed with the matrix multiplication. The Intel V-Tune profiler failed to profile simulations that displayed this page-fault error.

```
[2024-03-04 08:22:58][u.sm121949@sdp-pvc ~]$ dmesg --time
    -format=iso | grep "2024-02-23T11:20:39"
```

```
2024-02-23T11:20:39,752448-06:00 i915 0000:9a:00.0: page
    fault @ 0x01000010d6241000, ccs0 in python3
    [2609731]
2024-02-23T11:20:39,752451-06:00 i915 0000:9a:00.0: EU
    debugging disabled, EUs not interrupted, dumping
    error state to /sys/class/drm/card0/error
```

The above issues were addressed by employing a subset of kernels offered by the GiMMiK package. While the GiMMiK package typically provides four kernels per matrix multiplication routine, the following subset of kernels were found robust across all simulations:

```
...
[backend-opencl]
gimmik-nkern-M0 = [0,1]
gimmik-nkern-M3 = [0,1]
gimmik-nkern-M6 = [0,1]
gimmik-nkern-M132 = [0,1]
gimmik-nkern-M460 = [0,1,2,3]
...
```

The PyFR branch modified from the develop branch to isolate the kernels is available as `pearc24revision` branch in the first author's Github repository[18].