Metrics and Design of an Instruction Roofline Model for AMD GPUs

MATTHEW LEINHAUSER, Center for Advanced Systems Understanding, and University of Delaware
RENÉ WIDERA, SERGEI BASTRAKOV, and ALEXANDER DEBUS, Helmholtz-Zentrum Dresden-Rossendorf Laboratory
MICHAEL BUSSMANN, Center for Advanced Systems Understanding, and Helmholtz-Zentrum Dresden-Rossendorf Laboratory
SUNITA CHANDRASEKARAN, University of Delaware

Due to the recent announcement of the Frontier supercomputer, many scientific application developers are working to make their applications compatible with AMD (CPU-GPU) architectures, which means moving away from the traditional CPU and NVIDIA-GPU systems. Due to the current limitations of profiling tools for AMD GPUs, this shift leaves a void in how to measure application performance on AMD GPUs. In this article, we design an instruction roofline model for AMD GPUs using AMD's ROCProfiler and a benchmarking tool, BabelStream (the HIP implementation), as a way to measure an application's performance in instructions and memory transactions on new AMD hardware. Specifically, we create instruction roofline models for a case study scientific application, PIConGPU, an open source particle-in-cell simulations application used for plasma and laser-plasma physics on the NVIDIA V100, AMD Radeon Instinct MI60, and AMD Instinct MI100 GPUs. When looking at the performance of multiple kernels of interest in PIConGPU we find that although the AMD MI100 GPU achieves a similar, or better, execution time compared to the NVIDIA V100 GPU, profiling tool differences make comparing performance of these two architectures hard. When looking at execution time, GIPS, and instruction intensity, the AMD MI60 achieves the worst performance out of the three GPUs used in this work.

CCS Concepts: • **General and reference → Performance;**

Additional Key Words and Phrases: Roofline model, instruction roofline model, AMD GPU, ROCProfiler, performance modeling

This work was partly funded by the Center for Advanced Systems Understanding (CASUS) which is financed by the German Federal Ministry of Education and Research (BMBF) and by the Saxon Ministry for Science, Art, and Tourism (SMWK) with tax funds on the basis of the budget approved by the Saxon State Parliament. This research partially used resources of the Oak Ridge Leadership Computing Facility (OLCF) at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. This material is based upon work supported by the National Science Foundation (NSF) under grant no. 1814609.

Authors' addresses: M. Leinhauser, Center for Advanced Systems Understanding, Görlitz, Germany, University of Delaware, Newark, Delaware, USA; email: mattl@udel.edu; R. Widera, S. Bastrakov, and A. Debus, Helmholtz-Zentrum Dresden-Rossendorf Laboratory, Dresden, Germany; emails: {r.widera, s.bastrakov, a.debus}@hzdr.de; M. Bussmann, Center for Advanced Systems Understanding, Görlitz, Germany, Helmholtz-Zentrum Dresden-Rossendorf Laboratory, Dresden, Germany; email: m.bussmann@hzdr.de; S. Chandrasekaran, University of Delaware, Newark, Delaware, USA; email: schandra@udel.edu.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2022 Association for Computing Machinery.

2329-4949/2022/01-ART1 $15.00
https://doi.org/10.1145/3505285

ACM Reference format:
Matthew Leinhauser, René Widera, Sergei Bastrakov, Alexander Debus, Michael Bussmann, and Sunita Chandrasekaran. 2022. Metrics and Design of an Instruction Roofline Model for AMD GPUs. *ACM Trans. Parallel Comput.* 9, 1, Article 1 (January 2022), 14 pages.
https://doi.org/10.1145/3505285

1 INTRODUCTION

One of the most invaluable resources for performance and analytical modeling of **high performance computing** (**HPC**) codes is the roofline model developed by Williams et al. in 2009 [22]. The roofline model offers a simple way to visually understand an application's performance (in FLOPS) and its bottlenecks. Understanding performance and bottlenecks helps in optimizing codes. Thus, the roofline model has proved itself as an invaluable resource for performance and analytical modeling of HPC codes. Using the roofline model, application developers easily know what to optimize within a code, although it might be challenging to implement those optimizations. However, the traditional roofline model only gives specific optimization insights into a code. Thus, researchers have developed several extensions, with perhaps the most useful being the **cache-aware roofline model** (**CARM**) [16] and the hierarchical roofline model [23, 24]. Both the CARM and hierachical roofline model extend the traditional roofline model beyond the traditional DRAM/HBM measurement to include the cache memories. Typically, only the L1 and L2 caches are included. To distinguish the CARM and hierarchical roofline model from other types of roofline models, in this work, we refer to them as the **roofline performance model** (**RPM**).

Another extension to the traditional roofline model, the **instruction roofline model** (**IRM**), was developed in 2019 [15]. This model offers additional performance insights for an application beyond the RPM such as access patterns and instruction throughput. Creating an IRM is very similar to constructing an RPM. Instead of calculating maximum achieved GFLOPs for the compute ceiling, the maximum achieved billions of instructions per second (GIPS) is calculated. For the memory bandwidth ceiling, instead of using the measured bandwidth in GB/s, the GB/s bandwidth is divided by the size of a transaction (32 bytes) to use billions of transactions per second (GTXN/s). For clarity, GIPS is preferred over GFLOPs because instruction-level performance is important for the IRM. Similarly, GTXN/s is favored over GB/s because, as Ding and Williams describe, an execution-level load, such as a warp-load, can create up to 32 transactions on NVIDIA GPUs depending on the memory patterns used. Therefore, making the transaction the preferred memory unit for analyzing memory access.

Given that most of the leading supercomputers in the world currently contain NVIDIA GPUs [2], porting an application to work with AMD GPUs is an important open challenge. Due to the architecture and terminology differences between NVIDIA and AMD GPUs, the metrics traditionally gathered by hardware profilers are not applicable to use to create roofline models for AMD GPUs. In this work, we design an IRM for AMD GPUs using metrics from the ROCProfiler [9], AMD's hardware profiler, translate equations meant for NVIDIA GPUs to AMD GPU components, and create formulas specifically for AMD GPUs. PIConGPU, the application we look at in this work, was selected as one of the eight teams to take part in the **Department of Energy's** (**DOE**) Frontier **Center for Accelerated Application Readiness** (**CAAR**) effort [1]. Frontier, the **Oak Ridge Leadership Computing Facility's** (**OLCF**) newest supercomputer, will reach completion and start running programs in early 2022, and will contain AMD CPUs and AMD GPUs in each node.

This article makes the following contributions:

— Defines metrics and formulas needed to create an instruction roofline for state-of-the-art AMD GPUs. These metrics and formulas can provide a significant contribution to the design of AMD's profiling tool

— Provides a framework to model instruction-level performance of an HPC application on AMD GPUs
— Compares the performance of a plasma physics code, PIConGPU, using IRMs. Finding out which architecture PIConGPU performs better on is unclear due to profiling tool limitations for AMD GPUs

2 RELATED WORK

Of the existing research on IRMs, the most similar to our work comes from Ding et al. [15]. They defined the formulas necessary to create an IRM and to plot achieved performance. The most significant difference from our work is that Ding et al.'s model focuses on creating IRMs for NVIDIA GPUs. Specifically, some of the formulas are tied to NVIDIA's basic level of execution, the warp (32 threads). Since AMD's basic level of execution is the wavefront, all of the formulas defined in their work that use warps are unusable for AMD GPUs. It is worth noting that a wavefront can constitute 32 threads in some consumer GPUs (AMD RDNA2 GPUs), but this article focuses on AMD's HPC GPUs, where the size of a wavefront is 64 threads. Additionally, the metrics they gathered to create the roofline model come from NVIDIA's legacy profiler, nvprof. This profiling tool is compatible only with NVIDIA GPUs, thus leaving a gap on which metrics to use for AMD GPUs.

Recently, there have been a few efforts to create an IRM for AMD GPUs. Richards et al. [21] and Mehta et al. [20] created IRMs for the AMD Radeon Instinct MI60 GPU using proxy applications. Unfortunately, both papers lack sufficient details on how the roofline models were created. While some of the metrics used are explicitly mentioned, we do not know exactly how they were used to measure theoretical and achieved instruction-level performance. We build on these works by clearly defining which metrics are used to create an IRM and how they are used to plot achieved performance. Additionally, our work leverages AMD's state-of-the-art GPU, the AMD Instinct MI100. To the best of our knowledge, no other research uses the MI100 GPU (released in November 2020), or details formulas to create an IRM for AMD GPUs.

3 EXPERIMENTAL SETUP

In this section, we go over the specifications of the machines we ran the PIConGPU simulations on, Summit and an early access Frontier Center of Excellence machine. We also discuss the specifications of the next-generation supercomputer, Frontier, that will run PIConGPU once built.

3.1 Summit

The Summit supercomputer, built from IBM AC922 nodes, currently sits in the OLCF at Oak Ridge National Laboratory in Oak Ridge, Tennessee, USA. Summit, at the time of writing, is the world's second fastest supercomputer [2]. The machine features 4,608 nodes. Each node consists of two IBM Power9 CPUs (9,216 in total) and 6 NVIDIA Tesla V100 GPUs (27,648 in total). Each CPU contains 512 GB DDR4 RAM. In total, each node has 96 GB of **High Bandwidth Memory** (**HBM**) and uses NVIDIA's high-speed NVLink to communicate. A total of 256 cabinets are used to contain the machine. The machine can achieve 200 peta-floating-point operations per second (PFLOPs), or 200×10^{15} floating-point operations per second. The NVIDIA V100 GPUs use CUDA, which is a parallel computing platform and a programming model used exclusively for GPU computing on NVIDIA devices. On Summit we utilize Alpaka 0.6.0, CUDA 10.1.243, cupla 0.3.0-dev, Nsight Compute 2020.1, and Nsight Systems 2020.5.1.85.

3.2 Early Access Frontier COE Machine

The **Early Access Frontier COE Machine** (**EAFCOEM**) features eight nodes. Each node contains one HPC and artificial intelligence optimized AMD EPYC CPU. In addition to the CPU, two

nodes contain four NVIDIA Tesla V100 GPUs and the other six nodes contain four purpose built AMD Radeon Instinct GPUs. Of these six nodes, some contain Radeon Instinct MI60 GPUs while others hold AMD Instinct MI100 GPUs. As the release of Frontier nears, the nodes on this system will change to reflect the CPU and GPU devices the production Frontier machine will contain. Currently, the EAFCOEM supports both CUDA and HIP due to having both NVIDIA and AMD GPU devices. On the EAFCOEM, we utilize up to AMD ROCm 4.1.1 and HIP, Alpaka 0.6.0, and the latest stable commits of rocProf (up to and including commit 759f081) and BabelStream (up to and including commit 5182342).

3.3 Frontier (2021–2022)

Similar to the EAFCOEM, Frontier will hold one HPC and artificial intelligence optimized AMD EPYC CPU and four purpose built AMD Radeon Instinct GPUs per node. At the time of writing, we know Frontier will have over 100 cabinets to fit the nodes and use AMD Infinity Fabric to communicate. Frontier will use HIP, which is an open-source parallel computing model (an extension of C++) that can be used across multiple devices regardless of vendor. The peak performance is supposed to reach greater than 1.5 exa-FLOPs (EFLOPs), which is 1.5×10^{18} floating point operations per second.

4 CONSTRUCTING IRMS FOR AMD GPUS

In this section, we introduce AMD's ROCProfiler and explain which metrics are needed from it to measure an application's performance using an IRM. We then introduce the formulas used to create the IRM and show how the metrics gathered from the ROCProfiler can be applied to the IRM.

4.1 Gathering Metrics Using the ROCProfiler

The AMD ROCProfiler (rocProf) is a command line profiling analysis tool. This tool allows the user to get performance counters—and derived metrics from those counters—for an application. The tool works solely for applications using the ROCm accelerator backend. The metrics used for deriving IRMs on the NVIDIA V100 cannot be used on the AMD Radeon Instinct MI60 or AMD Instinct MI100 GPUs. The reason for this is because there is no way to extract the number of transactions from the L1 cache, L2 cache, or the DRAM/HBM using rocProf. Instead, we use rocProf to get the FETCH_SIZE, WRITE_SIZE, SQ_INSTS_SALU, and SQ_INSTS_VALU metrics, and the kernel runtimes to construct a roofline model. The FETCH_SIZE metric returns the total number of **kilobytes** (**KBs**) fetched from the GPU memory. Similarly, the WRITE_SIZE metric returns the total number of KBs written to the GPU memory. Before using these metrics, we convert each value from KBs to bytes. The SQ_INSTS_SALU metric tells how many scalar-ALU instructions are issued to the GPU. Similarly, the SQ_INSTS_VALU metric tells how many vector-ALU instructions are issued to the GPU.

4.2 Using Metrics to Create an IRM

The IRMs presented here for the AMD MI60 and MI100 GPUs are built off of the work of Richards et al. [21] from 2020. Instead of re-scaling the memory bandwidth to billions of transactions per second (GTXN/s), we leave the memory bandwidth in GB/s. Additionally, since there is no way to extract the number of transactions from rocProf, we use instructions per byte as the measurement unit on the horizontal axis, instead of instructions per transaction.

To calculate the achieved GIPS and instruction intensity performance, we need to find the number of instructions issued. We show how to get the number of instructions in Equation (1). We multiply the SQ_INSTS_VALU metric by four because this metric gives the number of instructions

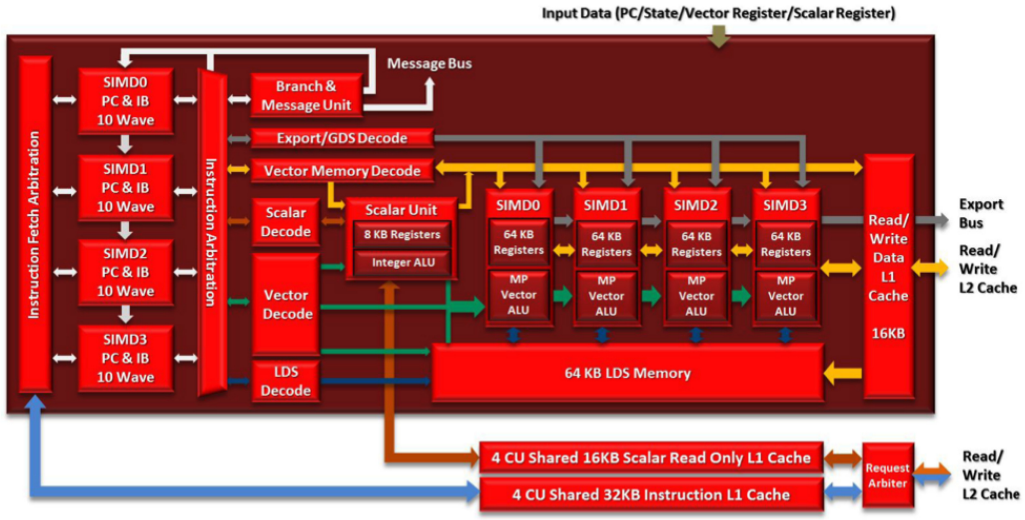

Fig. 1. The compute unit for all AMD GCN GPUs, which includes the MI60. The AMD CDNA GPUs (MI100) Compute Units are based off of the compute unit shown here. This image comes from the AMD GCN white paper [14].

issued per SIMD. For the AMD MI60 and MI100 GPUs, there are four SIMD vector units per **compute unit** (**CU**). Figure 1, which comes from the original AMD **Graphics Core Next** (**GCN**) GPU white paper [14], shows the four vector ALU units per SIMD. Similarly, we do not multiply the SQ_INSTS_SALU metric by anything because there is only one scalar unit per CU.

$$instructions = (SQ\_INSTS\_VALU \times 4) + SQ\_INSTS\_SALU. \tag{1}$$

To calculate the instruction intensity performance, measured in instructions per byte, the FETCH_SIZE and WRITE_SIZE metrics from rocProf are used. The sum of those metrics is then multiplied by the kernel runtime and that quantity divides the number of wavefront scaled instructions. This is shown in Equation (2). It is worth noting that because we normalize the instructions to the wavefront-level, we are actually calculating wavefront-level instruction intensity performance rather than a universal instruction intensity performance. Additionally, we calculate the instruction intensity performance rather than the instruction intensity because of the limited metrics available to use with rocProf. Instruction intensity performance can be easily determined by calculating the throughput of GPU HBM reads and writes.

$$Instruction\ Intensity\ Performance = \frac{\frac{instructions}{64}}{(bytes\ read + bytes\ written) \times runtime}. \tag{2}$$

To calculate the peak theoretical GIPS, we modify the peak GIPS equation from [15] to work with AMD architecture. AMD uses the term *CUs* instead of *streaming multiprocessors*. The MI60 and MI100 contain 64 and 120 CUs, respectively. Additionally, AMD GPUs use wavefronts instead of warps. The MI60 and MI100 GPUs each contain one wavefront scheduler per compute unit (WFS/CU). The theoretical **instructions per cycle** (**IPC**) variable is 1 (1 IPC) as stated in [10]. This is in unison with [15] despite focusing on a different GPU vendor. The frequency is measured in gigahertz as shown in the following equation:

$$GIPS_{peak} = CU \times WFS/CU \times IPC \times frequency. \tag{3}$$

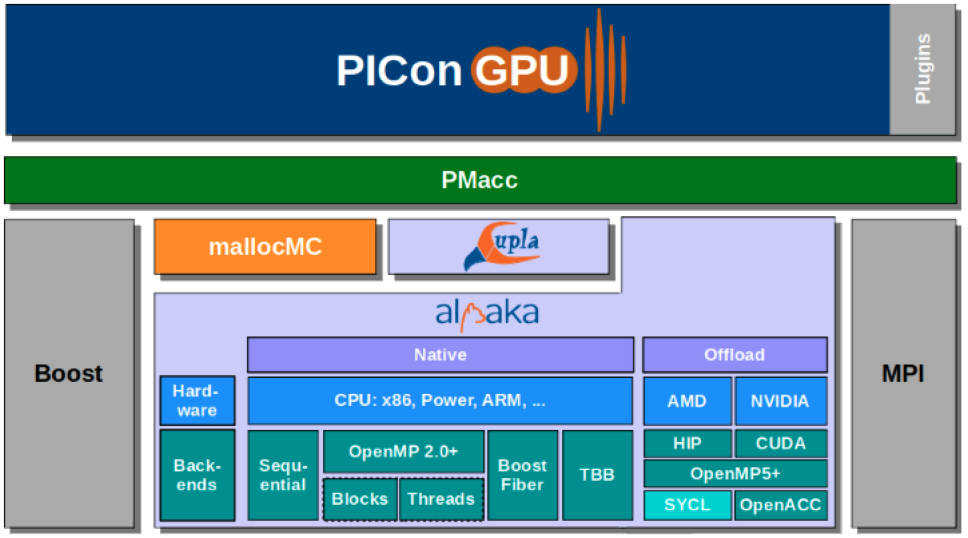

Fig. 2. The software stack of PIConGPU. Due to the abstraction Alpaka brings, only a few top level changes were made to get PIConGPU running on AMD GPUs via HIP.

The achieved instruction performance ($GIPS_{achieved}$) in GIPS is calculated by the formula shown in Equation (4). We divide by 64 because 64 threads constitute a wavefront in the AMD GPUs we target. The number of instructions is calculated as shown in Equation (1).

$$GIPS_{achieved} = \frac{\frac{instructions}{64}}{1 \times 10^9 \times runtime}. \tag{4}$$

The IRMs for AMD GPUs could easily re-scale the bandwidth into GTXN/s as shown for the V100 IRMs, and this might seem like a more equal comparison, but since we cannot get the number of transactions to use for the instruction intensity/instruction intensity performance, we did not want to offer a misleading comparison.

5 PICONGPU, A PLASMA PHYSICS APPLICATION

PIConGPU [11] is an open source **particle-in-cell** (**PIC**) simulations application that runs on general purpose GPUs for plasma and laser-plasma physics used to develop advanced particle accelerators for radiation therapy of cancer, high-energy physics, and photon science. PIConGPU utilizes the Alpaka [19] backend and the PIC algorithm for its science case simulations. Alpaka is an open-source abstraction library written in C++14 that aims at providing performance portability across accelerators through the abstraction of underlying levels of parallelism. It is platform independent and also supports concurrent and cooperative use between the host device and any attached accelerators. Alpaka is used on top of HIP and therefore, most of the porting is done in Alpaka rather than PIConGPU. Due to the software stack PIConGPU uses, only a few top level changes are made to support running on AMD GPUs via HIP. Figure 2 shows the entire PIConGPU software stack. More information about porting PIConGPU can be found in the 2016 ICHPC paper [25] and the Alpaka code repository [3]. As part of the ongoing CAAR for Frontier effort, we analyze the performance of PIConGPU on OLCF's Summit supercomputer, the second fastest in the world [2] as of the time of writing, and on an early access Frontier Center of Excellence machine.

In this work, we run PIConGPU's **Traveling Wave Electron Acceleration** (**TWEAC**) and **Laser Wakefield Acceleration** (**LWFA**) simulations. A deeper dive into the science of the TWEAC and LWFA simulations is explained by Debus et al. [13]. We profile the simulations using various state-of-the-art profiling tools and micro-kernel benchmarking suites. Using the metrics gathered from profiling, we construct roofline models for PIConGPU's MoveAndMark and ComputeCurrent kernels. These kernels are measured as the most computationally intensive [18] and are called in every PIConGPU simulation run. Therefore, optimizing these kernels will improve every simulation in PIConGPU. By constructing roofline models, we determine the future optimizations needed to exploit the best performance of PIConGPU.

6 PROFILING TOOLS USED

To profile the TWEAC and LWFA simulations on the NVIDIA V100 GPU, we use NVIDIA's legacy profiling tool, NVProf [8], and its state-of-the-art profiling tools, Nsight Compute [6] and Nsight Systems [7]. These profiling tools can only target NVIDIA GPUs. Hence we cannot use these tools to profile simulations on AMD GPUs.

To profile PIConGPU on the AMD devices, we initially used rocProf. However, we quickly found that rocProf did not suffice for acquiring all the metrics needed to construct IRMs and thus we pivoted to use various benchmarking suites. These benchmarking suites filled in most of the gaps needed to create roofline models for the AMD devices. As AMD architecture continues to grow in use, we hope to see new profiling tools with broader capabilities released in the future. The following subsections narrate findings using NVIDIA's tools and the micro-kernel benchmarking suites we used.

In making the roofline plots shown later in this report, we utilize the metrics the NERSC Roofline-on-NVIDIA-GPUs code repository uses [5]. The data gathered using rocProf, Nsight Compute, nvprof, and the HIP implementation of BabelStream can be found in a repository we created on GitHub [4]. Additionally, modifications we made to the NERSC code repository can be found in the same repository.

6.1 Nsight Compute and Nsight Systems

Nsight Compute is NVIDIA's latest application profiling tool. It boasts similar functionality to NVProf, but does not map the application runtime by kernel calls (as NVProf did). One of the biggest enhancements, the Roofline Analysis feature, came to Nsight Compute in version 2020.1. The Roofline Analysis feature automatically generates a roofline plot within the profiling report that is output. This work utilizes the Roofline Analysis feature and expands upon its initial use by utilizing Nsight Compute to create custom roofline plots.

Nsight Systems visually maps an application from execution to termination. The visualization, that is in the form of a timeline, is useful for deducing which kernels take up the most execution time, which bottlenecks exist in the code, and which kernels under-perform. This work utilized Nsight Systems to verify the developers' rationale about PIConGPU's most computationally intensive kernels and to find out the percentage of runtime those kernels took up. Figure 3 shows a visualization of the timeline from Nsight Systems to focus on our kernels of interest.

6.2 Micro-Kernel Benchmark Tools

To gather the memory bandwidth for the AMD MI60 and MI100 devices, we use a variety of micro-kernel benchmarking tools. First, we use the gpumembench [17] Benchmark Suite as a way to assess on-chip GPU memory bandwidth. Using the programs in the suite, we measure the instruction throughput, shared memory operations, and constant memory operations on the MI60 and MI100 GPUs. The other benchmark tool we use is BabelStream [12]. Formerly called GPU-Stream,

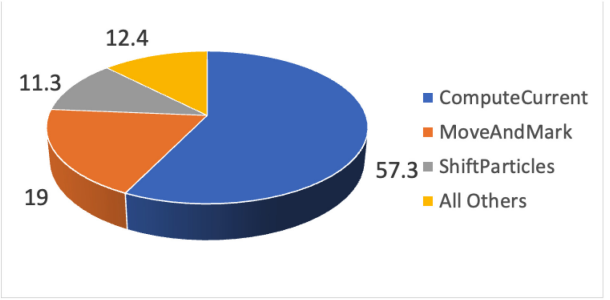

Fig. 3. Execution time (%) for different kernels within PIConGPU's TWEAC science case. The MoveAndMark and ComputeCurrent kernels take up over 75% of the overall runtime.

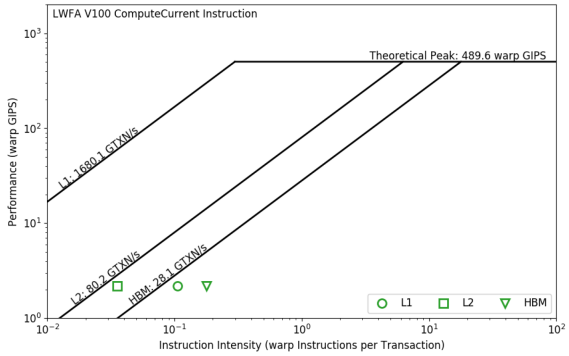

Fig. 4. The IRM for the ComputeCurrent kernel in the LWFA simulation on the **NVIDIA V100 GPU** on OLCF's Summit. Looking at the plotted points, one can see this kernel's instruction-level performance can be greatly improved.

BabelStream measures memory transfer rates to and from the global device memory on GPUs. BabelStream differs from other GPU memory bandwidth benchmarks and benchmarking suites in that it does not include PCIe transfer time in its results. BabelStream provides memory bandwidth results that are attainable. The output for the copy functions (808,975.476 MB/s for the MI60 GPU and 933,355.781 MB/s for the MI100 GPU) from BabelStream is used to represent the memory bandwidth for the AMD MI60 and MI100 GPU IRMs. It is worth noting that the measured bandwidth can vary when using an implementation of BabelStream other than the HIP implementation (for example the AOMP implementation). On the roofline plots, we convert each measurement to GB/s.

7 ROOFLINE MODEL COMPARISONS

In this section, we show IRMs generated on the NVIDIA V100, AMD MI60, and AMD MI100 for the PIConGPU LWFA science case, as seen in Figures 4–6, and compare the results in Table 1. Additionally, we show an IRM for the AMD MI60 and AMD MI100 for PIConGPU's TWEAC science case (Figure 7). Table 2 shows the same metrics that Table 1 did but for the TWEAC simulation. The IRM plots were created by modifying the scripts from the NERSC Roofline-On-NVIDIA-GPUs code repository. Finally, we dedicate a subsection to discussing the differences between AMD and NVIDIA GPU hardware and profiling tools to explain some of the differences outlined in this section.

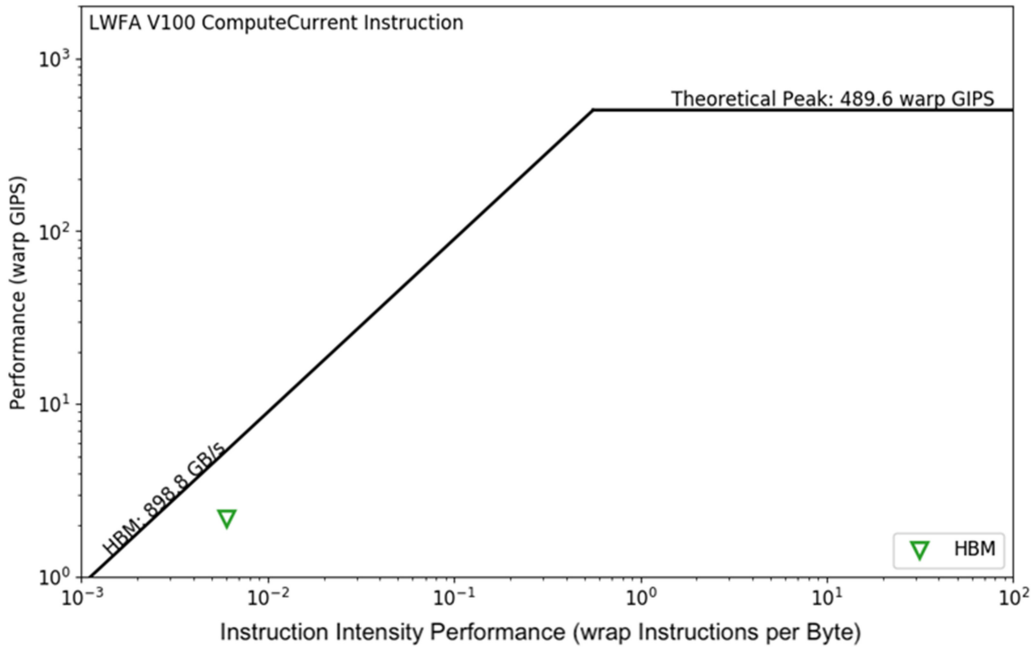

Fig. 5. The IRM for the ComputeCurrent kernel in the LWFA simulation on the **NVIDIA V100 GPU** on OLCF's Summit. Here, the instruction intensity is measured in instructions/byte rather than instructions/transaction. This plot shows that there is much room for improvement.

7.1 NVIDIA GPU IRMs

To construct the roofline models on the NVIDIA V100, we collected the same metrics from NVProf as mentioned in [15]. The roofline model shown in Figure 4 represents the IRM for an instance of the ComputeCurrent kernel ran during an LWFA science case simulation of PIConGPU. The plot shows the compute ceiling measured in GIPS and the memory bandwidth ceiling in GTXN/s, along with the achieved warp GIPS and instruction intensity for the HBM, and the L1 and L2 caches. We observe in [15] that L1 points appearing on the left side of the plot (i.e. L1 points that have low instruction intensity) are more likely to show that the kernel has strided memory access patterns. We confirmed this is true for Figure 4 by following the method outlined in [15] under "Global Memory Walls". Similarly, L2 points appearing more left on the plot represent 32-way bank conflicts. Using Nsight Compute, we confirm this kernel experiences many bank conflicts. We also see that it is HBM-bound and its computational performance cannot increase without addressing the memory issues present. A difference between the plot in Figure 4 and the IRMs shown in [15] is that the V100 GPU utilized to run the PIConGPU LWFA simulation achieved a greater memory bandwidth than the one Ding et al. used thus leading to higher GTXN/s for the L1 cache, L2 cache, and HBM. Additionally, we also created an IRM for this kernel measuring instruction intensity in *instructions/byte* to give a better comparison between NVIDIA and AMD. Figure 5 shows the instruction intensity measured in instructions/byte. We avoid including the L1 and L2 caches so that it is more similar to its AMD counterparts.

7.2 AMD GPU IRMs

We follow the process of constructing IRMs outlined in Section 4 and find that the AMD MI60 GPU has a theoretical peak GIPS of 115.2 while the MI100 boasts a theoretical peak GIPS of 180.24. The

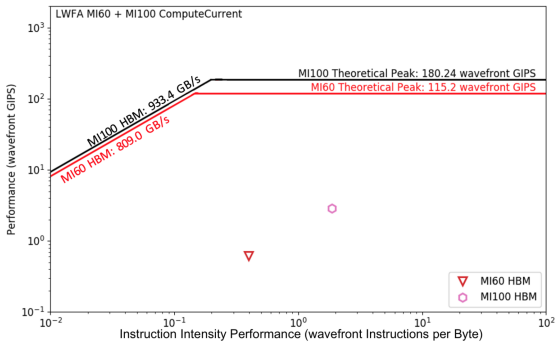

Fig. 6. The IRM for the ComputeCurrent kernel in the LWFA simulation on the **AMD MI60 and MI100 GPUs**. This simulation was run on an AMD/Cray early-access system at OLCF. The HBM point in this roofline plot appears in a much better position than than the HBM point for the V100's roofline plot.

Table 1. Execution Time, Achieved GIPS, and Instruction Intensity for the LWFA Simulation's ComputeCurrent Kernel on the NVIDIA V100 & AMD MI60 and MI100 GPUs

PIConGPU LWFA Simulation	**ComputeCurrent**		
GPU	V100	MI60	MI100
Execution Time (s)	0.0040	0.0127	0.0025
{Compute Units, Streaming Multiprocessors}	80	64	120
Instructions/Cycle	1	1	1
Frequency (GHz)	1.530	1.800	1.502
{Wavefront, Warp} Schedulers	4	1	1
Peak GIPS	489.60	115.20	180.24
Achieved GIPS	2.178	0.620	2.856
Instructions	279,498,240	502,440,960	449,796,480
Bytes Read	267,280,000,000	1,125,436,000	1,124,711,000
Bytes Written	97,329,000,000	432,711,000	408,483,000
{Wavefront, Warp}-Level Instruction Intensity (inst/byte)	0.006	0.398	1.863

The values in this table are rounded to three decimal points and therefore manually calculating Achieved GIPS and Instruction Intensity may vary slightly from the numbers shown here. For execution time, a lower number is better. For GIPS and Instruction Intensity, a higher number is better.

memory bandwidth was measured using BabelStream. Due to AMD profiling tool limitations, we are unable to measure performance for the L1 and L2 caches on any AMD GPU. Figure 6 shows the IRMs for both the AMD MI60 and MI100 on the LWFA science case's ComputeCurrent kernel. Even though we can only measure the HBM performance, we can assume that the L1 and L2 cache performance points would appear to the left of the HBM performance points. Following this assumption will give us synonymous information as the V100's IRM: the kernel has many bank conflicts and strided memory access. We want to stress the fact that we are *assuming* the AMD IRMs are showing bank conflicts and strided memory access. We are not able to confirm these assumptions because of the limited amount of performance counter metrics available in AMD profiling tools.

Table 1 shows the achieved GIPS performance, Instruction Intensity, and the runtime of the ComputeCurrent kernel on the NVIDIA V100, AMD MI60, and MI100 GPUs for the LWFA science case. The table shows the kernel executed significantly faster on the V100 and MI100 than the MI60, with the MI100 having the best execution time overall. The instruction intensities in the table are measured in $instructions/bytes$. According to [15], the instruction intensity should be measured in $instructions/transactions$. Due to performance counter limitations on rocProf, we could not

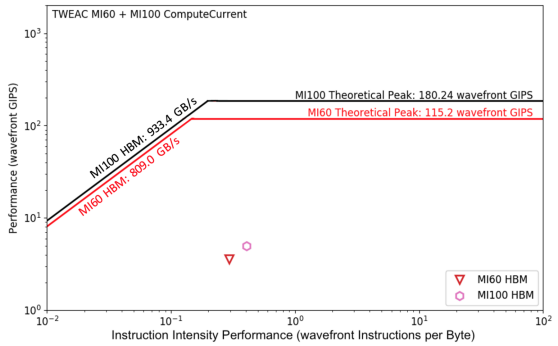

Fig. 7. The IRM for the ComputeCurrent kernel in the TWEAC simulation on the **AMD MI60 and MI100 GPUs**. This simulation was run on an AMD/Cray early-access system at OLCF. Unlike its NVIDIA V100 counterpart, this roofline plot offers no insight into the L1 and L2 cache performance due to a lack of performance counters on the AMD GPUs.

Table 2. Execution Time, Achieved GIPS, and Instruction Intensity for the TWEAC Simulation's ComputeCurrent kernel on the NVIDIA V100 & AMD MI60 and MI100 GPUs

PIConGPU TWEAC Simulation	**ComputeCurrent**		
GPU	V100	MI60	MI100
Execution Time (s)	0.283	0.394	0.246
{Compute Units, Streaming Multiprocessors}	80	64	120
Instructions/Cycle	1	1	1
Frequency (GHz)	1.530	1.800	1.502
{Wavefront, Warp} Schedulers	4	1	1
Peak GIPS	489.60	115.20	180.24
Achieved GIPS	6.634	3.586	4.993
Instructions	60,149,000,000	90,319,028,127	78,488,570,820
Bytes Read	40,931,000,000	11,451,009,000	11,460,394,000
Bytes Written	1,810,100,000	785,101,000	792,172,000
{Wavefront, Warp}-Level Instruction Intensity (inst/byte)	0.155	0.293	0.408

The values in this table are rounded to three decimal points and therefore manually calculating Achieved GIPS and Instruction Intensity may vary slightly from the numbers shown here. For execution time, a lower number is better. For GIPS and Instruction Intensity, a higher number is better.

extract the amount of transactions. On the NVIDIA V100, the instruction intensity was measured as 0.178 instructions per transactions (as shown in Figure 4). To make an equal comparison the V100's instruction intensity is measured in instructions per byte in Table 1.

Looking at the ComputeCurrent kernel for PIConGPU's TWEAC simulation, we see similar results to its LWFA simulation for the same kernel. Table 2 shows that the MI100 still has the fastest execution time of the kernel, but its achieved GIPS is much lower than the V100. To measure instructions on AMD GPUs, we use the vector-ALU and scalar-ALU instructions stated by rocProf. The V100 uses an nvprof metric (*inst_executed*) that measures all the different types of instructions the profiled kernel uses. Just as in the table above, the instruction intensity for the V100 in Table 2 is reported in instructions per byte to make an equal comparison with AMD GPUs. The instruction intensity on the V100 in instructions per transaction is 4.931. In Tables 1 and 2, the NVIDIA V100 processes a much lesser number of instructions and reads and writes a significantly higher number of bytes to the DRAM than the AMD GPUs. At the time of writing, we are unsure of why this behavior is occurring and continue to look into it. Figure 7 shows the IRM for PIConGPU's TWEAC simulation ComputeCurrent kernel.

7.3 Discussion of Rooflines, Profiling Tools, and GPU Hardwares

As we alluded to earlier, there are many differences between AMD and NVIDIA GPUs. The first main difference is the size of a warp and a wavefront. One possible explanation for the difference between achieved GIPS on NVIDIA GPUs and AMD GPUs could be the size of a wavefront (64 threads) vs. the size of a warp (32 threads). Since a wavefront is twice as big as a warp, achieved GIPS on AMD GPUs are at a significant disadvantage when compared head to head with NVIDIA GPUs due to how instructions are scaled. If each GPU issued 100,000 compute instructions, the achieved GIPS for an NVIDIA GPU would be twice as high as the AMD GPU's solely due to the idea of scaling instructions to the warp-/wavefront-level. However, this is necessary in order to figure out the instruction-level performance of the GPU. Additionally, nvprof's *inst_executed* metric measures *all* types of instructions issued by the NVIDIA GPU whereas the vector-ALU and scalar-ALU instructions comprise *only* the compute instructions for AMD GPUs. We decided to utilize the *inst_executed* metric for the experiments conducted in this work because both nvprof and Nsight Compute do not offer a single metric to extract only the compute instructions issued to an NVIDIA GPU like rocProf does. This means that the NVIDIA GPU might contain instructions that have nothing to do with the compute instructions of the kernel being measured as opposed to the instructions metrics we collect for AMD GPUs, which only contain compute instructions. However, Nsight Compute does allow users to obtain detailed per-instruction-type metrics which can then be aggregated across all compute instructions.

Looking beyond what constitutes the achieved points and instead focusing on the base rooflines themselves, we also see a few differences. The first noted difference is the memory bandwidth for each GPU. Using Nsight Compute, we see the achieved bandwidth of the NVIDIA V100 GPU (shown in Figure 5) is over 99% of its theoretical bandwidth (900 GB/s). With rocProf, we cannot measure the achieved bandwidth for AMD GPUs so we use the HIP implementation of BabelStream to do so. Using BabelStream, the AMD MI60 achieves 81% of its theoretical bandwidth and the MI100 achieves 78%. The compute ceilings also show different stories. The theoretical GIPS ceiling for the V100 is about 2.7x higher than the MI100's and 4.25x higher than the MI60's. As Ding and Williams described in [15], the theoretical GIPS ceiling is calculated using a GPU's hardware components. The biggest differences that lead to the lower theoretical GIPS ceiling for the AMD MI60 are the number of CUs (64 vs. the V100's 80 and MI100's 120) and the 1 wavefront scheduler per CU (vs. the V100's 4 warp schedulers per streaming multiprocessor). Despite having the highest frequency out of all three GPUs used, these two differences lead the MI60 to have the lowest theoretical GIPS ceiling. Similar to the MI60, the MI100's GIPS ceiling is low compared to the V100's because the MI100 only has 1 wavefront scheduler per CU. The warp/wavefront schedulers per CU/streaming multiprocessor are what drive the theoretical GIPS up. For example, if the V100 only had 1 warp scheduler per streaming multiprocessor, its theoretical GIPS ceiling would be only 122.4, a quarter of what it is now. Due to these fundamental differences between GPU hardware and their profiling tools, it is hard to create an instruction-level performance model that will equally relate GPUs made by different vendors.

Finally, the last point we want to make is that while roofline models offer many insights into a kernel's performance, they do not explicitly show the performance limiters of that kernel. While showing memory bandwidth, instruction throughput or FLOPS under-utilization is helpful, the real performance limiters (memory stalls, instruction dependencies, etc.) must be analyzed separately. Adding on to the difficulty of optimizing kernels is that some performance limiters can be architecture dependent. In this article, all three of the GPUs used have many similarities and therefore improvements and refactoring to reduce and/or avoid performance limiters have positive effects for all devices. Alpaka provides the infrastructure necessary to write unified code, but

how memory is mapped to workers/threads depends on the architecture and how it is abstracted for PIConGPU using PMacc (see Figure 2).

8 CONCLUSIONS AND FUTURE WORK

In this article, we showed how to construct IRMs for AMD GPUs using metrics from rocProf and micro-kernel benchmarking suites. We also highlighted the existing limitations that make measuring performance on AMD GPUs a challenge. Finally, we built and compared IRMs for PIConGPU's TWEAC and LWFA simulations. Specifically, we looked at the LWFA's Compute-Current kernel on different hardware (NVIDIA vs. AMD) and compared the performances of the hardware used. We find that the result of which hardware PIConGPU performs better on is unclear because of the profiling tool limitations for AMD GPUs. Additionally, comparing the achieved instruction-level performance between AMD and NVIDIA is difficult due to the limitations of each vendor's profiling tool(s).

In the future, we hope to expand upon this work by designing and constructing roofline models, along with analyzing the performance of PIConGPU, on future AMD GPUs found in the Frontier supercomputer. We will continue to look into why the AMD MI100 is processing more instructions than the NVIDIA V100, and why the V100 is reading and writing more bytes to the DRAM than the AMD GPUs. Similarly, we wish to identify how many additional instructions are added due to the intrusion of profiling tools. Finally, we will investigate how to extract the achieved FLOPs and the number of L1 and L2 cache, and HBM/DRAM read and write transactions from AMD GPUs to allow for more equal comparisons between NVIDIA and AMD GPUs.

ACKNOWLEDGMENTS

The authors thank Ronnie Chatterjee, the former PIConGPU CAAR liaison for his invaluable support. Additionally, we extend gratitude to Nicholas Malaya, the AMD Technical Lead for the Frontier Center of Excellence for his help with understanding AMD GPUs and to Felix Schmitt of NVIDIA for his help on understanding Nsight Compute metrics. We extend thanks to Nan Ding & Sam Williams at LBNL for their insights into instruction roofline models. Our thanks to Shirley Moore, of UTEP, and David Richards, of LLNL, for their help in understanding rocProf. Finally, our thanks to David Rogers of ORNL and our current CAAR liaison for the valuable discussions.

REFERENCES

[1] Frontier Center for Accelerated Application Readiness (CAAR). 2019. Retrieved 11 Mar 2021 from https://www.olcf.ornl.gov/caar/frontier-caar/.
[2] The Top500 List. 2020. Retrieved from https://www.top500.org/.
[3] Alpaka Code Repository. 2021. Retrieved from https://github.com/alpaka-group/alpaka.
[4] AMD-Instruction-Roofline-using-rocProf-Metrics Code Repository. 2021. Retrieved from https://github.com/Techercise/AMD-Instruction-Roofline-using-rocProf-Metrics.
[5] NERSC Roofline-on-NVIDIA-GPUs Code Repository. 2021. Retrieved from https://gitlab.com/NERSC/roofline-on-nvidia-gpus.
[6] NVIDIA Nsight Compute. 2021. Retrieved from https://developer.nvidia.com/nsight-compute.
[7] NVIDIA Nsight Systems. 2021. Retrieved from https://developer.nvidia.com/nsight-systems.
[8] NVIDIA NVProf. 2021. Retrieved from https://docs.nvidia.com/cuda/profiler-users-guide/index.html#nvprof-overview.
[9] ROC-profiler. 2021. Retrieved from https://github.com/ROCm-Developer-Tools/rocprofiler.
[10] Paul Bauman, Noel Chalmers, Nick Curtis, Chip Freitag, Joe Greathouse, Nicholas Malaya, Damon McDougall, Scott Moe, René van Oostrum, and Noah Wolfe. 2019. Intro to AMD GPU Programming with HIP.
[11] M. Bussmann, H. Burau, T. E. Cowan, A. Debus, A. Huebl, G. Juckeland, T. Kluge, W. E. Nagel, R. Pausch, F. Schmitt, U. Schramm, J. Schuchart, and R. Widera. 2013. Radiative signatures of the relativistic Kelvin-Helmholtz instability. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM, New York, NY. DOI: https://doi.org/10.1145/2503210.2504564

[12] Tom Deakin, James Price, Matt Martineau, and Simon McIntosh-Smith. 2016. GPU-STREAM v2. 0: Benchmarking the achievable memory bandwidth of many-core processors across diverse parallel programming models. In *Proceedings of the International Conference on High Performance Computing*. Springer, 489–507.

[13] Alexander Debus, Richard Pausch, Axel Huebl, Klaus Steiniger, René Widera, Thomas E. Cowan, Ulrich Schramm, and Michael Bussmann. 2019. Circumventing the dephasing and depletion limits of laser-wakefield acceleration. *Physical Review X* 9, 3 (Sep 2019), 031044. DOI: https://doi.org/10.1103/PhysRevX.9.031044

[14] AM Devices. 2012. AMD Graphics Cores Next (GCN) Architecture (whitepaper).

[15] Nan Ding and Samuel Williams. 2019. An instruction roofline model for GPUs. In *Proceedings of the 2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*. IEEE, 7–18.

[16] Aleksandar Ilic, Frederico Pratas, and Leonel Sousa. 2013. Cache-aware roofline model: Upgrading the loft. *IEEE Computer Architecture Letters* 13, 1 (2013), 21–24.

[17] Elias Konstantinidis and Yiannis Cotronis. 2016. A quantitative performance evaluation of fast on-chip memories of gpus. In *Proceedings of the 2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. IEEE, 448–455.

[18] Matthew Leinhauser, Jeffrey Young, Sergei Bastrakov, René Widera, Ronnie Chatterjee, and Sunita Chandrasekaran. 2021. *Performance Analysis of PIConGPU: Particle-in-Cell on GPUs using NVIDIA's NSight Systems and NSight Compute*. Technical Report. Oak Ridge National Lab.(ORNL), Oak Ridge, TN.

[19] Alexander Matthes, René Widera, Erik Zenker, Benjamin Worpitz, Axel Huebl, and Michael Bussmann. 2017. Tuning and optimization for a variety of many-core architectures without changing a single line of implementation code using the Alpaka library. In *Proceedings of the International Conference on High Performance Computing*. Springer, 496–514.

[20] Neil A. Mehta, Rahulkumar Gayatri, Yasaman Ghadar, Christopher Knight, and Jack Deslippe. 2020. Evaluating performance portability of OpenMP for SNAP on NVIDIA, Intel, and AMD GPUs using the Roofline Methodology. In *Proceedings of the WACCPD@ SC*.

[21] D. F. Richards, O. Aaziz, J. Cook, S. Moore, D. Pruitt, and C. Vaughan. 2020. *Quantitative Performance Assessment of Proxy Apps and Parentsreport for ecp Proxy App Project Milestone adcd-504-9*. Technical Report. Lawrence Livermore National Lab.(LLNL), Livermore, CA.

[22] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: An insightful visual performance model for multicore architectures. *Communications of the ACM* 52, 4 (2009), 65–76.

[23] Samuel W. Williams. 2010. The roofline model. In *Performance Tuning of Scientific Applications*, David H. Bailey, Robert F. Lucas, and Samuel W. Williams (Eds.). CRC Press, 205–226.

[24] Charlene Yang, Thorsten Kurth, and Samuel Williams. 2020. Hierarchical Roofline analysis for GPUs: Accelerating performance optimization for the NERSC-9 Perlmutter system. *Concurrency and Computation: Practice and Experience* 32, 20 (2020), e5547.

[25] Erik Zenker, René Widera, Axel Huebl, Guido Juckeland, Andreas Knüpfer, Wolfgang E. Nagel, and Michael Bussmann. 2016. Performance-portable many-core plasma simulations: Porting PIConGPU to openpower and beyond. In *Proceedings of the International Conference on High Performance Computing*. Springer, 293–301.

Received March 2021; revised July 2021; accepted September 2021