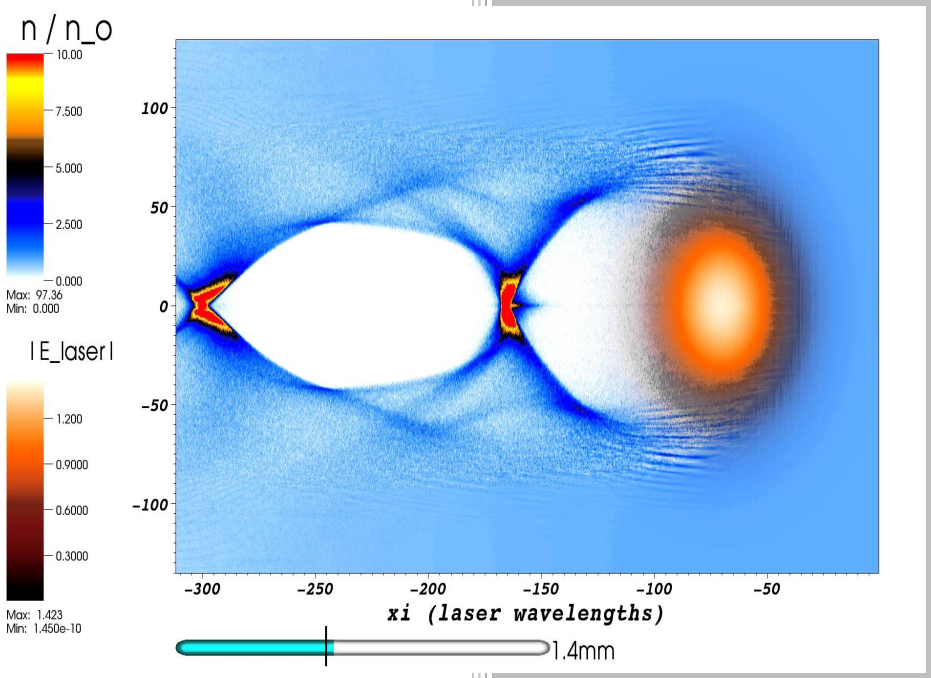
ORNL/TM-2020/1813

Performance Analysis of PIConGPU: Particle-in-Cell on GPUs using NVIDIA's NSight Systems and NSight Compute



Matthew Leinhauser
Jeffrey Young
Sergei Bastrakov
René Widera
Arghya Chatterjee
Sunita Chandrasekaran

Date: January 19, 2021

OAK RIDGE NATIONAL LABORATORY
MANAGED BY UT-BATTELLE FOR THE US DEPARTMENT OF ENERGY

ORNL/TM-2020/1813

Program: Center for Accelerated Application Readiness (CAAR)
Division: National Center for Computational Sciences (NCCS)

Performance Analysis of PIConGPU: Particle-in-Cell on GPUs using NVIDIA's NSight Systems and NSight Compute

Author(s)

Matthew Leinhauser[†], Jeffrey Young[‡], Sergei Bastrakov[Φ], René Widera[Φ],
Arghya Chatterjee[¶], Sunita Chandrasekaran[†]

[†] University of Delaware
[‡] Georgia Institute of Technology
[Φ] Helmholtz-Zentrum Dresden-Rossendorf
[¶] Oak Ridge National Laboratory

Date Published: January 19, 2021

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, TN 37831-6283
managed by
UT-Battelle, LLC
for the
US DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

CONTENTS

LIST OF FIGURES

ACRONYMS

ORNL	Oak Ridge National Laboratory
OLCF	Oak Ridge Leadership Computing Facility
PIC	Particle in Cell
PIConGPU	Particle in Cell on Graphics Processing Unit
CAAR	Center for Accelerated Application Readiness
AMD	Advanced Micro Devices
FOM	Figure of Merit
TWEAC	Traveling-Wave Electron Acceleration
LWFA	Laser-Wakefield Accelerators
PFLOPS	Peta Floating Point Operations per Second

ACKNOWLEDGMENTS

The authors first want to acknowledge the support from OLCF's User Assistance and Outreach (UAO) team members for working with us to get our application running optimally on Summit. We also want to thank Jeff Larkin and Robbie Searles from the NVIDIA COE team in helping us understand some of the nuances of the NVIDIA Performance profiling tool chain. Finally, the authors want to extend their gratitude towards Oscar Hernandez (ORNL) for his support in assisting us throughout the publication of the Technical Report.

We further acknowledge the Principal Investigator (PI) of the PIConGPU application, Dr. Michael Bussmannn for his support and allowing us to use both the application and key personnel from his team at the Helmholtz-Zentrum Dresden-Rossendorf (HZDR) laboratory.

ABSTRACT

PIConGPU, Particle In Cell on GPUs, is an open source simulations framework for plasma and laser-plasma physics used to develop advanced particle accelerators for radiation therapy of cancer, high energy physics and photon science. While PIConGPU has been optimized for at least 5 years to run well on NVIDIA GPU-based clusters (1), there has been limited exploration by the development team of potential scalability bottlenecks using recently updated and new tools including NVIDIA's NVProf tool and the brand-new NVIDIA NSight Suite (Systems and Compute) tools.

PIConGPU is a highly optimized application that runs production jobs at scale on a system Oak Ridge Leadership Facility's (OLCF) Summit supercomputer (using the full machine at 4600 nodes; at 98% of GPU utilization on all ~28000 NVIDIA Volta GPUs). PIConGPU has been selected as one of the the eight applications for OLCF's coveted Center for Accelerated Application Readiness (CAAR) program aimed at the facility's Frontier supercomputer (OLCF's first exascale system to launch in 2021), to partner with our vendors (primary vendors: AMD and Cray/HPE) ensuring that Frontier will be able to perform large-scale science when it opens to users in 2022 (2).

To this effect, performance engineers on the PIConGPU team wanted to dive deep into the application to understand at the finest granularity, which portions of the code could be further optimized to exploit the hardware on Summit at it's maximum potential and also to elucidate which key kernels should be tracked and optimized for the CAAR effort to port this code to Frontier. Any bottlenecks that are observed via performance profiling on Summit are likely to also impact scalability on the Frontier-dev system and the Frontier Early Access (EA) system. Additionally, the engineers wanted to take a closer look at the newest NVIDIA profiling tools which allows us to identify the most useful features on these tools and will provide an opportunity to compare it to new AMD and Cray's performance analysis tool releases and provide feedback to our vendor partners on what features are most important and mission critical for CAAR efforts.

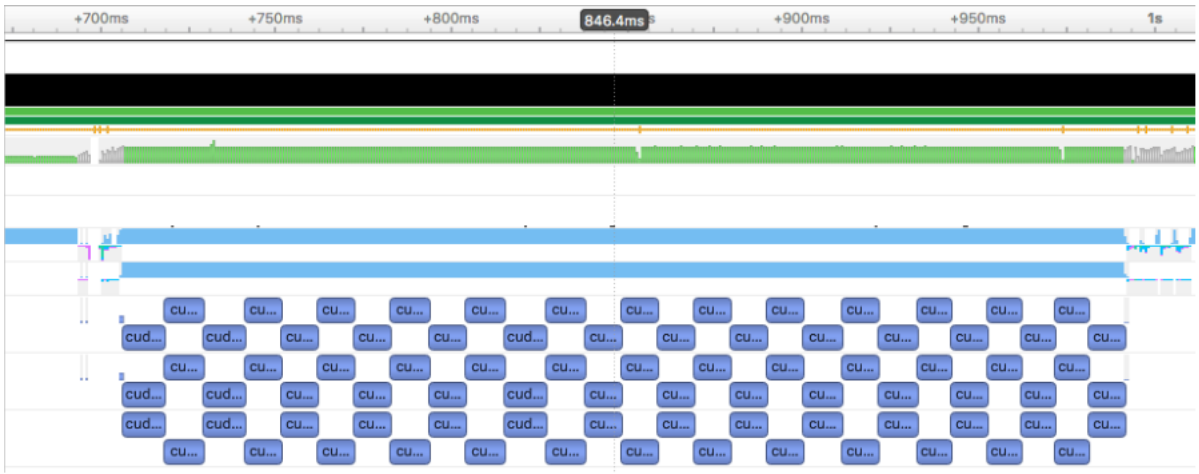


Figure 1. A timeline view of PIConGPU, captured using NSight Systems

The primary goal of this report is to focus on the evaluation of PIConGPU's most time-intensive kernels using NVProf and NSight Suite.

Three kernels, Current Deposition (also known as Compute Current), Particle Push (Move and Mark), and Shift Particles are known to be some of the most time-consuming kernels in PIConGPU. The Current

Deposition kernel and Particle Push kernel both set up the particle attributes for running any physics simulation with PIConGPU, so it is crucial to improve the performance of these two kernels. In this report, we measure single GPU metrics for the three kernels, offer high level takeaways from the conducted analysis, and compare the profiling data from NSight Compute to that of NVProf. This analysis was performed using a grid size of 240 x 272 x 224, and 10 time steps with the Mid-November Figure of Merit (FOM) run setup. The Traveling Wave Electron Acceleration (TWEAC) science case used in this run is a representative science case for PIConGPU. This execution can also be used for baseline analysis on AMD MI50/ MI60 systems. As of the time of writing, the PIConGPU application has limited use for features of NSight Systems, so this report will mainly focus on insights garnered from NSight Compute. For this analysis, we run the "full" metric set available in NSight Compute version 2020.1.2 and use NSight Systems version 2020.3.1 to generate the application timeline.

1. HIGH-LEVEL TAKEAWAYS FROM THIS ANALYSIS

1. NVProf showed us several application-level insights on how to possibly tweak PIConGPU for better performance on Summit.
    - Using NSight Compute showed us similar metrics and reinforced that our initial analysis with NVProf was valid. NVProf profiling has more overhead since it captures a larger number of metrics by default, but both tools can be used on Summit to do useful analysis within the available 2 hour runtime limit.
2. NSight Compute in general offers similar functionality to NVProf. Some of the high-level metrics, such as Speed of Light, are not extremely useful for analyzing a code that has already been optimized for Summit and NVIDIA GPUs in general.
    - Nsight can be run with less overhead by using its more robust customization options with default metric sets. New metric sets are easy to create and combine for analysis runs. 100 PIConGPU timesteps of profiling can be run with Nsight Compute in a 2 hr time limit compared to 1 time step with NVProf's default level profiling.
    - Nsight Systems provides aggregate-level statistics for applications but currently has some issues with separating long C++ templated kernel names that makes it tough to break these statistics down by kernel. See Section 5. for more details on possible profiling tool enhancements.
3. Comparison of profiles that support AMD hardware to NVIDIA profiles would be much more insightful than comparing NVProf and Nsight outputs.
    - PIConGPU has been optimized for CUDA GPUs for many years so further insights from CUDA profilers may be limited.
    - A useful comparison of profilers that support NVIDIA hardware and AMD hardware would look at whether the same metrics like shared memory utilization show up as important predictors for performance with AMD GPUs.
4. Roofline Analysis takeaways
    - We also anticipate NVIDIA's new roofline analysis capability in NSight Compute 2020.1 to be a useful tool for further analysis of PIConGPU.

2. EXPERIMENTAL SETUP

2.1 Summit

Oak Ridge Leadership Computing Facility's (OLCF), Summit supercomputer is a 4600 node, 200 PFLOPS IBM AC922 system [*]. Each node consists of 2 *IBM POWER9* CPUs with 512 GB DDR4 RAM and 6 *NVIDIA Volta V100* GPUs with a total of 96 GB high bandwidth memory (divided into 2 sockets), all connected together with NVIDIA's high-speed NVLink. For this research we have primarily used our allocation as a part of the Center for Accelerated Application Readiness (CAAR) program on Summit.

[*]Summit ranked the second place in the TOP500 list in June 2020 (3)

2.2 Experiment: The Science

One of the major applications of PIC codes is to simulate laser-driven particle acceleration. With respect to electrons, the main acceleration mechanism is Laser-Wakefield accelerators. In this simulation, the experimental setup used is the Traveling Wave Electron Acceleration (TWEAC) simulation. TWEAC aims to avoid limitations inherent to the Laser-Wakefield Accelerators (LWFA) (namely the dephasing and depletion limits). The goal of the TWEAC simulation, as stated by Debus et al., is "to create a laser focal region that moves ideally with exactly the vacuum speed of light and thus faster than the plasma group velocity" (4). Using LWFA, the laser pulse becomes "exhausted" over time and the "quailty" of acceleration worsens, thus introducing a limit on how much can be realistically accelerated. Using TWEAC, this issue is avoided and a longer acceleration takes place because the electrons will always see an "unexhausted" piece of the laser.

3. MORE DETAILED ANALYSIS OF KERNELS

In this section, we go over the metrics computed by NSight Compute from each of the three measured kernels, Compute Current, Move And Mark, and Shift Particles in detail.

3.1 COMPUTE CURRENT KERNEL

This kernel contains the back reaction of moving particles to fields. Each particle performs scatter operations to nearby fields and each thread block processes a super cell. Each CUDA block corresponds to a super cell. The super cell result, which is three floating point values per cell (including halo cells) is cached into shared memory using atomic floating point operations. The temporary shared memory data is then written back to global memory. The numerical scheme the algorithm uses causes warp divergence due to the particles' positions relative to the grid. This kernel took up 57.3% of runtime.

3.1.1 Launch Statistics

At launch:

- The size of the kernel grid is 2280
- The block size is 512
- There are 1,167,360 threads
- There are 14.25 waves per Streaming Multiprocessor (SM)
- There are 54 registers per thread
- The Static Shared Memory per block is 18.26 KB/block
- There is no Dynamic Shared Memory per block (0 KB/block)
- The Shared Memory Configuration Size is 65.54 KB

3.1.2 GPU Speed of Light

According to an application analysis done by NSight Compute:

- "Memory is more heavily utilized than Compute: Check memory replay (coalescing) metrics to make sure you're efficiently utilizing the bytes transferred. Also consider whether it is possible to do more work per memory access (kernel fusion) or whether there are values you can (re)compute."
- The SM Speed of Light (SOL) is 47.21%
  - This is also defined as the SM throughput (assuming ideal load balancing across SMSPs)
  - For each unit, the Speed Of Light (SOL) reports the achieved percentage of utilization with respect to the theoretical maximum.
- 66.43% of the theoretical maximum memory was used

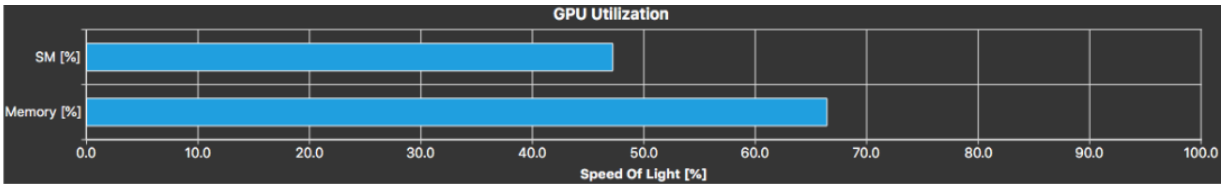


Figure 2. GPU Utilization of the Compute Current Kernel

3.2 Kernel Roofline

In the Compute Current kernel, all particle and field data are in single-precision. Due to this, only a single-precision roofline was generated by NSight Compute.

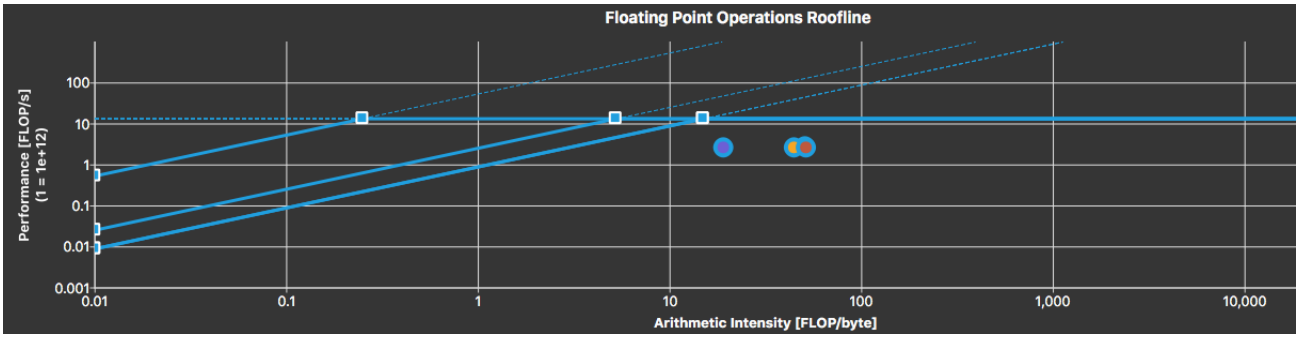


Figure 3. The Hierarchical Single Floating Point Roofline for the Compute Current Kernel within PIConGPU. From left to right, the diagonal lines represent the memory boundary for L1, L2, and DRAM. The purple dot represents the L1 cache achieved value; the orange dot represents the L2 cache achieved value, and the red dot represents the floating point achieved value.

Looking at the chart, we see that L1 cache, L2 cache, and achieved FLOP/s can all be optimized to reach the compute boundary. There are no issues with this kernel being memory-bound.

3.2.1 Compute Workload Analysis

- The SM was busy 48.99% of the time
- The issue slots were busy 48.99% of the time.

- The pipe was utilized most by Fused-Multiply-Add (FMA) instructions (39.02%) and least by Address Unit (ADU) instructions (0.04%). This is shown in Figure 3.
- 1.87 warp instructions were executed per cycle

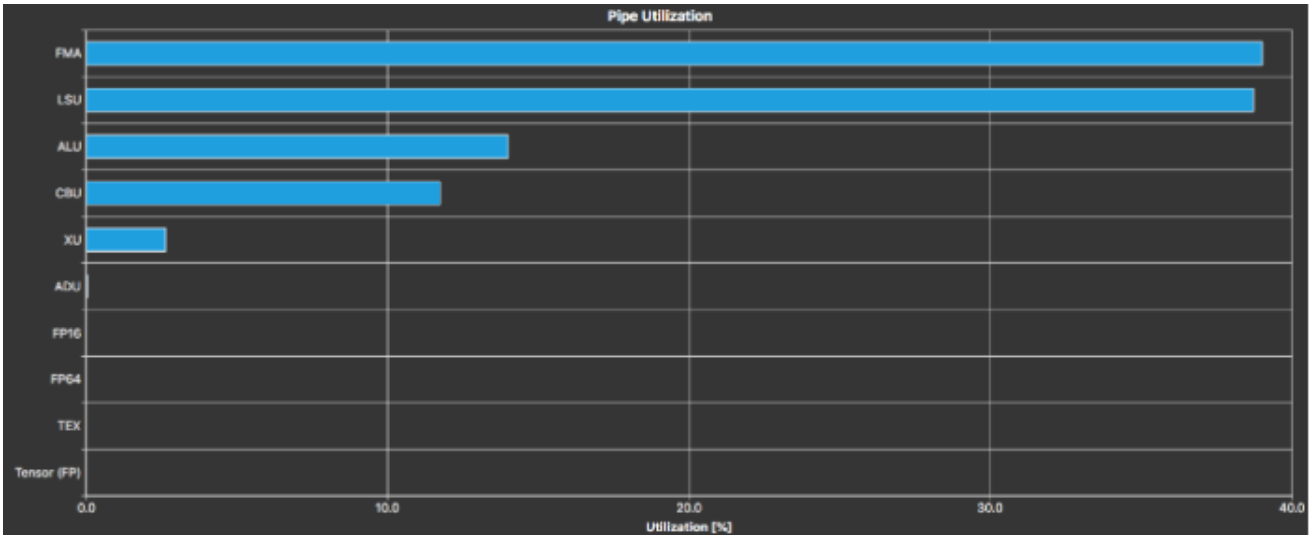


Figure 4. Pipe Utilization of the Compute Current Kernel

3.2.2 Memory Workload Analysis

- The memory throughput is 51.06 GB/second
- The available communication bandwidth between the SM, caches, and DRAM was 37.32%
  - This percentage does not necessarily limit the kernel's performance
- The maximum throughput of issuing memory instructions reached 39.71%
- Shared Memory Loads had 113,400,037 bank conflicts
- Shared Memory Loads had a peak utilization of 67.13%
- Shared Memory Stores had 0 bank conflicts
- Shared Memory Stores had a peak utilization of 38.25%

3.2.3 Occupancy

- The kernel's theoretical occupancy is 50%
- The kernel's achieved occupancy was 49.65%
- The kernel's theoretical active warps per SM is 32
- The kernel's active warps per SM was 31.78
- The registers block limit is 2

- The shared memory block limit is 5
- The warps block limit is 4
- The SM block limit is 32

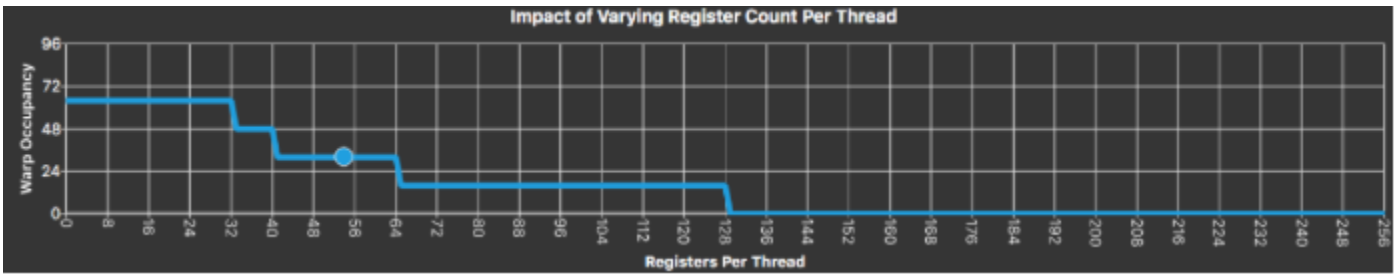


Figure 5. Looking at the graph, we see that we will achieve the highest amount of warp occupancy by decreasing the number of registers per thread to 0 - 31. We can also achieve better performance by decreasing this number in the range of 32 - 40 registers per thread.

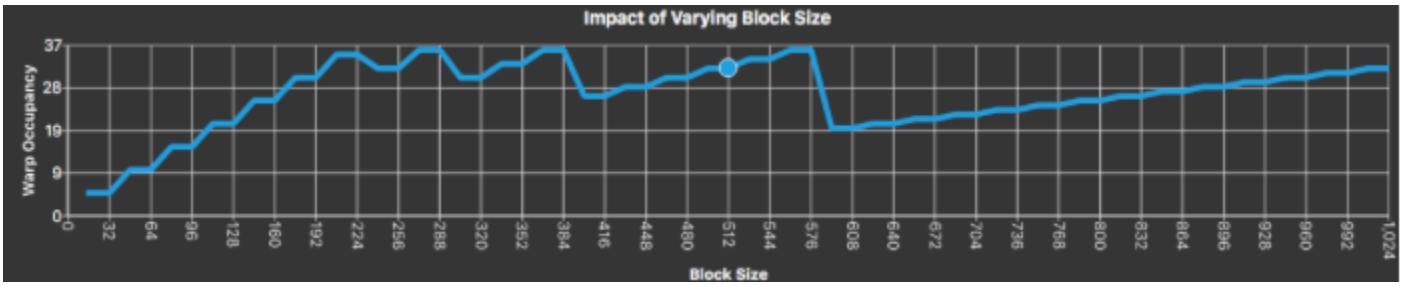


Figure 6. Looking at the graph, we see that increasing the block size from 512 to 576 will increase the number of warps. Similarly, decreasing the block size from 512 to 384 will also yield better warp occupancy.

3.2.4 Scheduler Statistics

According to an application analysis done by NSight Compute:

- "Every scheduler is capable of issuing one instruction per cycle, but for this kernel each scheduler only issues an instruction every 2.0 cycles. This might leave hardware resources underutilized and may lead to less optimal performance. Out of the maximum of 16 warps per scheduler, this kernel allocates an average of 7.95 active warps per scheduler, but only an average of 0.76 warps were eligible per cycle. Eligible warps are the subset of active warps that are ready to issue their next instruction. Every cycle with no eligible warp results in no instruction being issued and the issue slot remains unused. To increase the number of eligible warps either increase the number of active warps or reduce the time the active warps are stalled."
- This kernel achieved 7.95 warps out of a 16 theoretical warps per scheduled time period
- 0.49 warps were eligible per scheduler, but only 0.76 were issued
- 50.97% of the time, 0 warps were eligible to execute

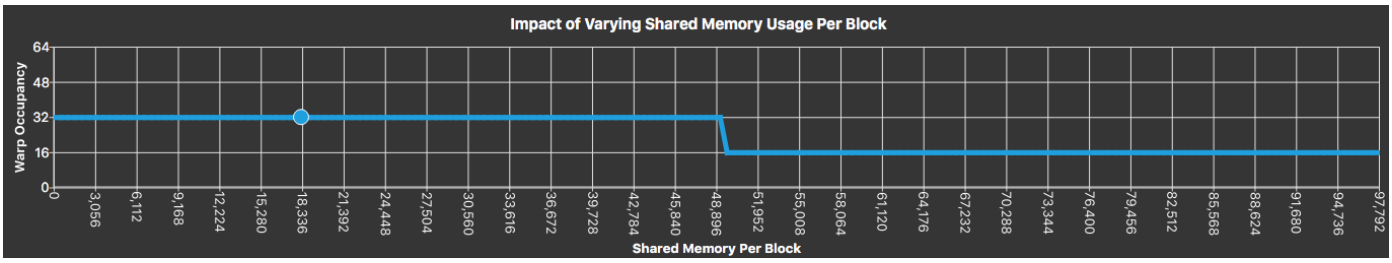


Figure 7. From this graph, we can see we are achieving the highest number of warps using 18,336 bytes of Shared Memory per Block.

- Conversely, 49.03% of the time, one or more warps were eligible to execute

3.2.5 Warp State Statistics

According to an application analysis done by NSight Compute:

- "Instructions are executed in warps, which are groups of 32 threads. Optimal instruction throughput is achieved if all 32 threads of a warp execute the same instruction. The chosen launch configuration, early thread completion, and divergent flow control can significantly lower the number of active threads in a warp per cycle. This kernel achieves an average of 20.6 threads being active per cycle. This is further reduced to 18.9 threads per warp due to predication. The compiler may use predication to avoid an actual branch. Instead, all instructions are scheduled, but a per-thread condition code or predicate controls which threads execute the instructions. Try to avoid different execution paths within a warp when possible. In addition, assure your kernel makes use of Independent Thread Scheduling, which allows a warp to reconverge after a data-dependent conditional block by explicitly calling __syncwarp()."
- "On average each warp of this kernel spends 5.1 cycles being stalled waiting for a scoreboard dependency on an MIO operation (not to TEX or L1). This represents about 31.2% of the total average of 16.2 cycles between issuing two instructions. The primary reason for a high number of stalls due to short scoreboards is typically memory operations to shared memory, but other contributors include frequent execution of special math instructions (e.g. MUFU) or dynamic branching (e.g. BRX, JMX). Consult the Memory Workload Analysis section to verify if there are shared memory operations and reduce bank conflicts, if reported."
- 16.21 Warp Cycles occurred per issued instruction
- 20.60 was the Average Number of Active Threads per Warp
  - According to NSight Compute, this number is caused by and if that is reduced or eliminated, this kernel will achieve a more optimal number of threads per warp.

3.3 MOVE AND MARK KERNEL

In this kernel, the momentum and position of a macro particle is updated based on the force calculated from the electric and magnetic fields. Each thread block processes a super cell. The fields of each super cell, three floating point values per cell, (including halo cells) is cached into shared memory. Each macro

particle performs a gather operation to interpolate the field to the position of the particle. Within this kernel, a particle accesses 64 cells. If a particle leaves a super cell, it is not physically moved, but rather only marked for removal. This kernel took up 19.0% of the runtime.

3.3.1 Launch Statistics

At launch:

- The size of the kernel grid is 57,120
- The block size is 256
- There are 14,622,720 threads
- There are 238 waves per SM
- There are 64 registers per thread
- The Static Shared Memory per block is 27.66 KB/block
- There is no Dynamic Shared Memory per block (0 KB/block)
- The Shared Memory Configuration Size is 98.30 KB

3.3.2 GPU Speed of Light

According to an application analysis done by NSight Compute:

- "The kernel is utilizing greater than 80.0% of the available compute or memory performance of the device. To further improve performance, work will likely need to be shifted from the most utilized to another unit."
- The SM SOL is 44.84%
- 90.93% of the theoretical maximum memory was used

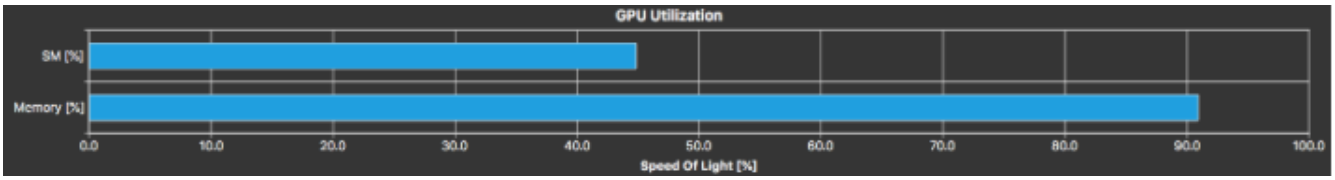


Figure 8. GPU Utilization of the Move and Mark Kernel

3.4 Kernel Roofline

Within the TWEAC simulation, the Move and Mark kernel of PIConGPU utilizes the Vay Pusher, which needs to perform some double precision operations. At a high level, particle data is loaded in single-precision, mixed-precision operations are performed while the particle push is occurring. Finally, a conversion back to single-precision takes place.

The graph shows that the L1 cache, L2 cache, and achieved FLOP/s are more compute-bound than memory-bound. All achieved values on the graph show that the architectural limit is close to being

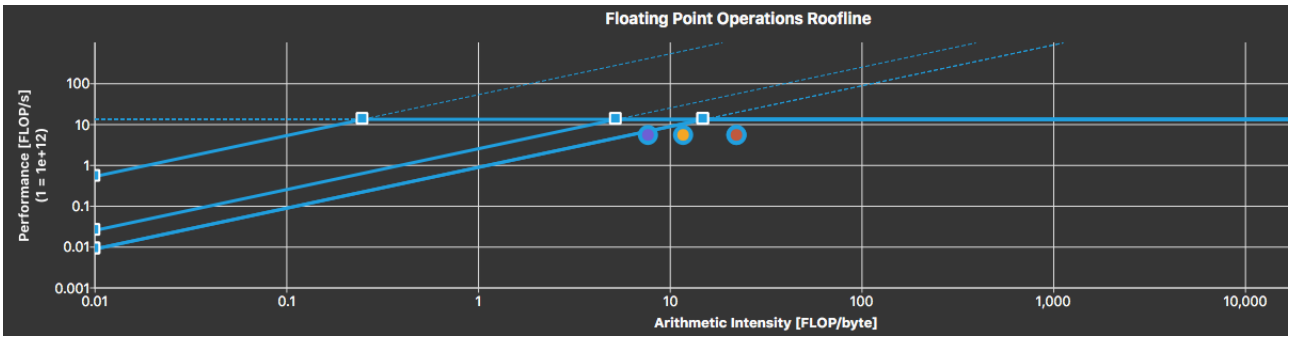


Figure 9. The Hierarchical Single Precision Roofline for the Move And Mark Kernel within PIConGPU. From left to right, the diagonal lines represent the memory boundary for L1, L2, and DRAM. The purple dot represents the L1 cache achieved value; the orange dot represents the L2 cache achieved value, and the red dot represents the floating point achieved value.

reached. To achieve the maximum compute, minor optimizations need to be made to reach the architectural maximum of the hardware.

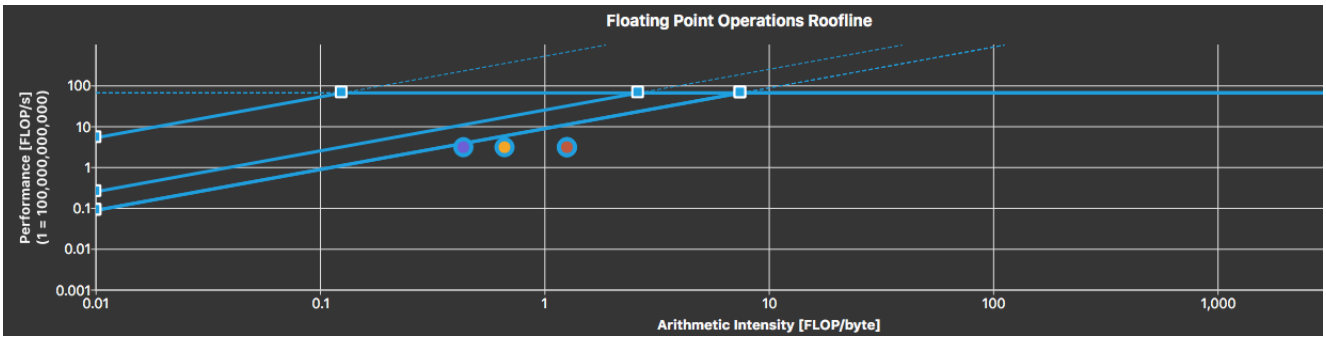


Figure 10. The Hierarchical Double Precision Roofline for the Move And Mark Kernel within PIConGPU. From left to right, the diagonal lines represent the memory boundary for L1, L2, and DRAM. The purple dot represents the L1 cache achieved value; the orange dot represents the L2 cache achieved value, and the red dot represents the floating point achieved value.

The graph shows that the L2 cache and achieved FLOP/s are more memory-bound than compute-bound. The L1 cache is more compute-bound. The compute performance of the L1 cache can be optimized to reach the maximum compute. To achieve the maximum compute within the L2 cache and achieved FLOP/s, optimizations need to be made to address the memory issues experienced during the Vay pusher process. After the achieved values become less memory-bound, we can make optimizations to reach the compute roofline.

3.4.1 Compute Workload Analysis

- The SM is busy 44.90% of the time.
- The issue slots were busy 43.34% of the time.
- The pipe was utilized most by Fused-Multiply-Add (FMA) instructions (44.90%) and least by

Address Unit (ADU) instructions (0.03%).

- 1.73 warp instructions were executed per cycle.

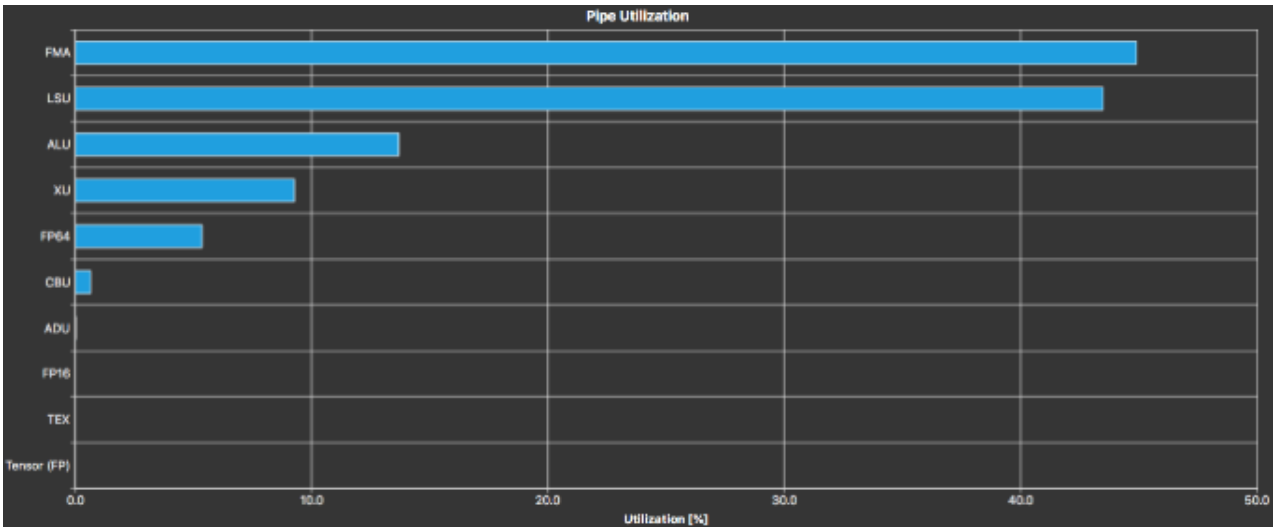


Figure 11. Pipe Utilization of the Move and Mark Kernel

3.4.2 Memory Workload Analysis

- The memory throughput is 218.53 GB/second
- The available communication bandwidth between the SM, caches, and DRAM was 43.41%
- The maximum throughput of issuing memory instructions reached 43.43%
- Shared Memory Loads had 4,931,034,721 bank conflicts
- Shared Memory Loads had a peak utilization of 87.50%
- Shared Memory Stores had 345,967 bank conflicts
- Shared Memory Stores had a peak utilization of 0.20%

3.4.3 Occupancy

- The kernel's theoretical occupancy is 37.50%
- The kernel's achieved occupancy was 37.44%
- The kernel's theoretical active warps per SM is 24
- The kernel's active warps per SM was 23.96
- The registers block limit is 4
- The shared memory block limit is 3
- The warps block limit is 8

- The SM block limit is 32

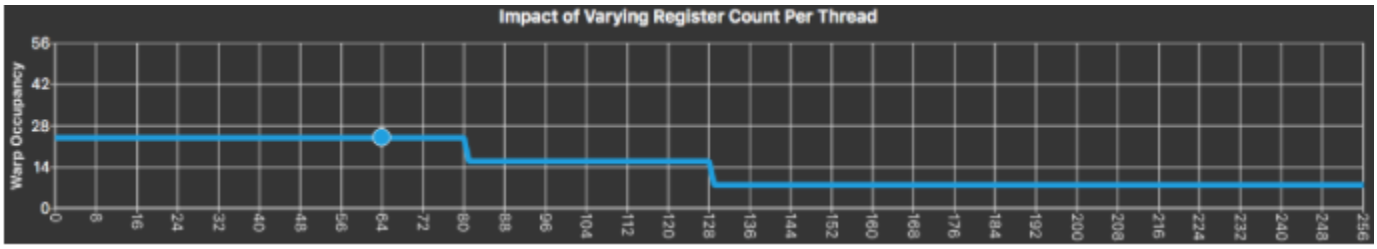


Figure 12. Looking at the graph, we see that we are achieving the highest amount of warp occupancy per the number of registers per thread

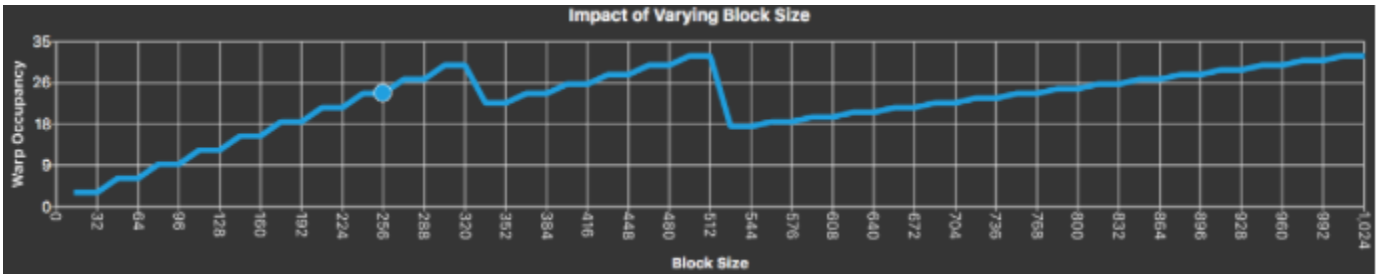


Figure 13. The graph shows that increasing the block size from 256 to 512 will increase the number of warps. However, this is tied to the number of cells per supercell/the number of particles in a buffer. This could be decoupled which could allow us to increase the number of threads per block, thus increasing the number of warps.

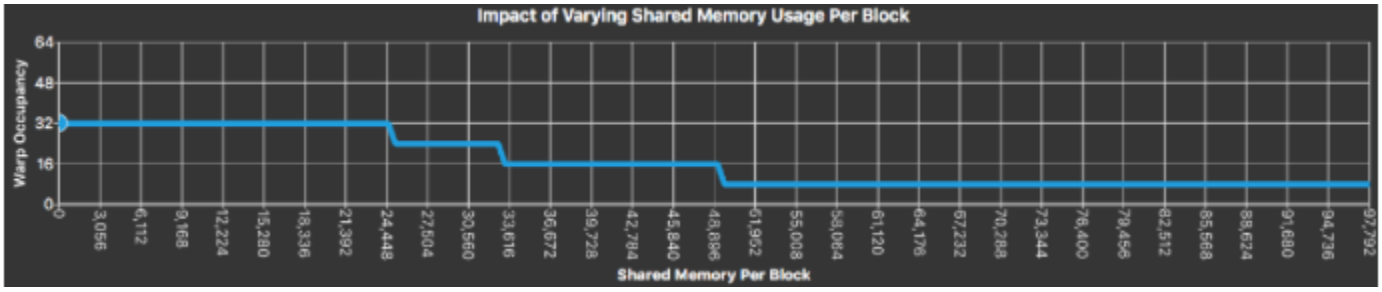


Figure 14. The graph incorrectly shows that this kernel is not using any shared memory.

NSight Compute shows that this kernel does not use shared memory in Figure 14. However, we know this is not correct. At the time of writing, we are still investigating why NSight Compute thinks this kernel does not use shared memory.

3.4.4 Scheduler Statistics

According to an application analysis done by NSight Compute:

- "Every scheduler is capable of issuing one instruction per cycle, but for this kernel each scheduler only issues an instruction every 2.3 cycles. This might leave hardware resources underutilized and

may lead to less optimal performance. Out of the maximum of 16 warps per scheduler, this kernel allocates an average of 5.99 active warps per scheduler, but only an average of 1.02 warps were eligible per cycle. Eligible warps are the subset of active warps that are ready to issue their next instruction. Every cycle with no eligible warp results in no instruction being issued and the issue slot remains unused. To increase the number of eligible warps either increase the number of active warps or reduce the time the active warps are stalled."

- This kernel achieved 5.99 warps out of a theoretical 16 warps scheduled time period
- 1.02 warps were eligible per scheduler, but only 0.43 were issued
- 56.66% of the time, 0 warps were eligible to execute
- Conversely, 43.34% of the time, one or more warps were eligible to execute

3.4.5 Warp State Statistics

According to an application analysis done by NSight Compute:

- "On average each warp of this kernel spends 4.5 cycles being stalled waiting for the MIO instruction queue to be not full. This represents about 32.9% of the total average of 13.8 cycles between issuing two instructions. This stall reason is high in cases of extreme utilization of the MIO pipelines, which include special math instructions, dynamic branches, as well as shared memory instructions."
- 13.82 Warp Cycles occurred per issued instruction
- 31.87 was the Average Number of Active Threads per Warp

3.5 SHIFT PARTICLES KERNEL

In this kernel, the particle storage data structure is updated so that it matches the particles' new positions after the push is performed. This kernel took up 11.3% of the runtime.

3.5.1 Launch Statistics

At launch:

- The size of the kernel grid is 2280
- The block size is 256
- There are 583,680 threads
- There are 7.12 waves per SM
- There are 56 registers per thread
- The Static Shared Memory per block is 2.66 KB/block
- There is no Dynamic Shared Memory per block (0 KB/block)
- The Shared Memory Configuration Size is 16.38 KB

3.5.2 GPU Speed of Light

According to an application analysis done by NSight Compute:

- "This kernel exhibits low compute throughput and memory bandwidth utilization relative to the peak performance of this device. Achieved compute throughput and/or memory bandwidth below 60.0% of peak typically indicate latency issues."
- The Streaming Multiprocessor (SM) Speed of Light (SOL) is 10.51%
- 14.02% of the theoretical maximum memory was used

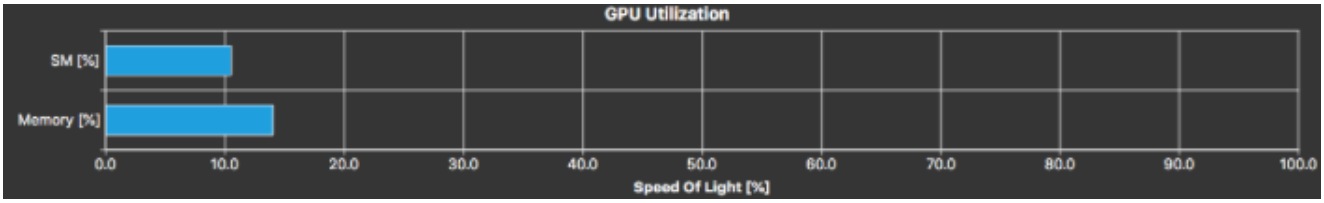


Figure 15. GPU Utilization of the Shift Particles Kernel

3.6 Kernel Roofline

There is no roofline for this kernel as it does not execute any FLOP/s. The Shift Particles kernel mainly copies data around and performs a small number of integer operations.

3.6.1 Compute Workload Analysis

- The SM is busy 11.11% of the time.
- The issue slots were busy 11.11% of the time.
- The pipe was utilized most by Load/Store Unit (LSU) instructions (10.16%) and least by Texture Memory (TEX) instructions (0.01%).
- 0.40 warp instructions were executed per cycle

3.6.2 Memory Workload Analysis

- The memory throughput is 120.01 GB/second
- The available communication bandwidth between the SM, caches, and DRAM was 14.02%
- The maximum throughput of issuing memory instructions reached 9.61%
- The Shared Memory Loads had 798,480 bank conflicts
- The Shared Memory Loads had a peak utilization of 2.45%
- The Shared Memory Stores had 781,119 bank conflicts
- The Shared Memory Stores had a peak utilization of 0.85%

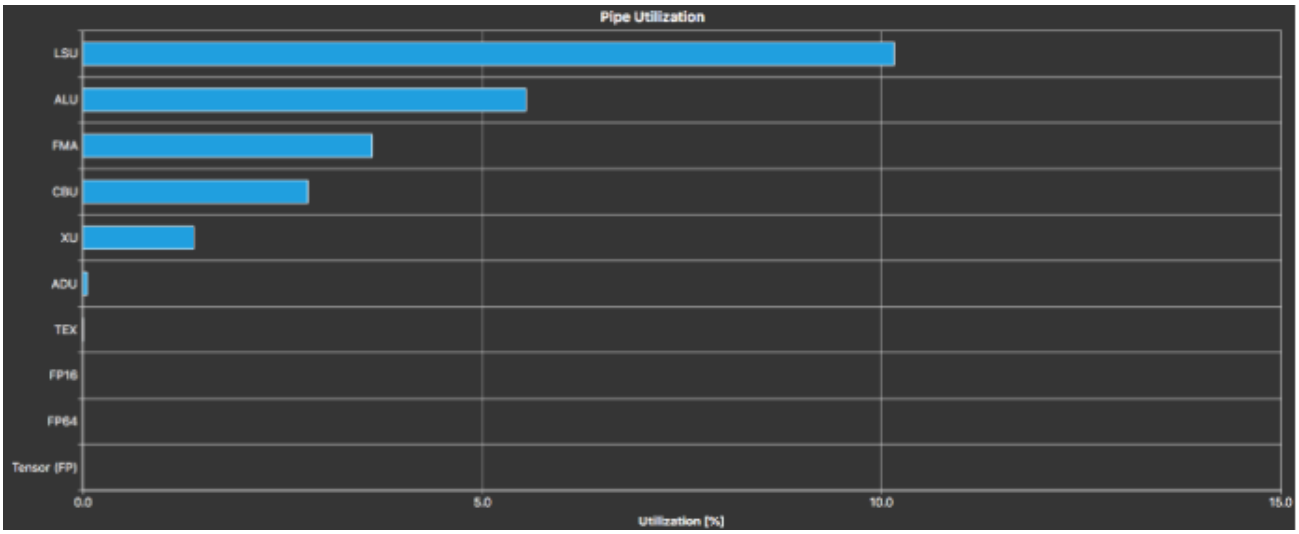


Figure 16. Pipe Utilization of the Shift Particles Kernel

3.6.3 Occupancy

- The kernel's theoretical occupancy is 50%
- The kernel's achieved occupancy was 47.92%
- The kernel's theoretical active warps per SM is 32
- The kernel's active warps per SM was 30.67
- The registers block limit is 4
- The shared memory block limit is 34
- The warps block limit is 8
- The SM block limit is 32

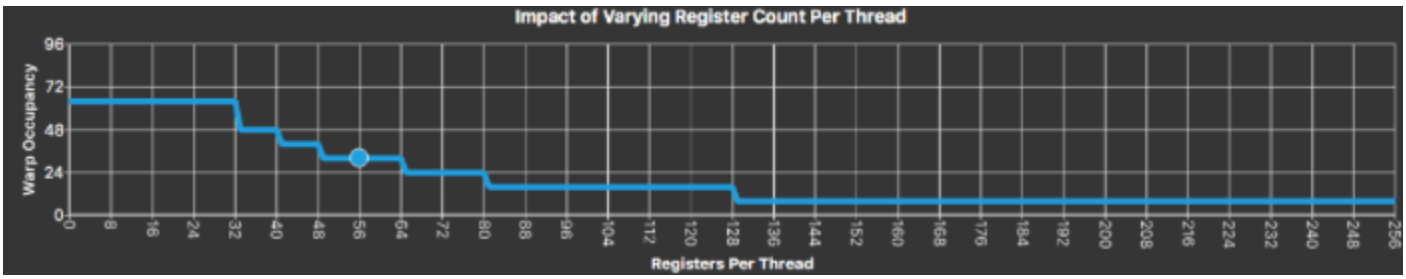


Figure 17. Looking at the graph, we see that decreasing the number of registers per thread will yield higher warp occupancy. We can achieve the highest amount of warp occupancy by decreasing the number of registers per thread to no more than 32.

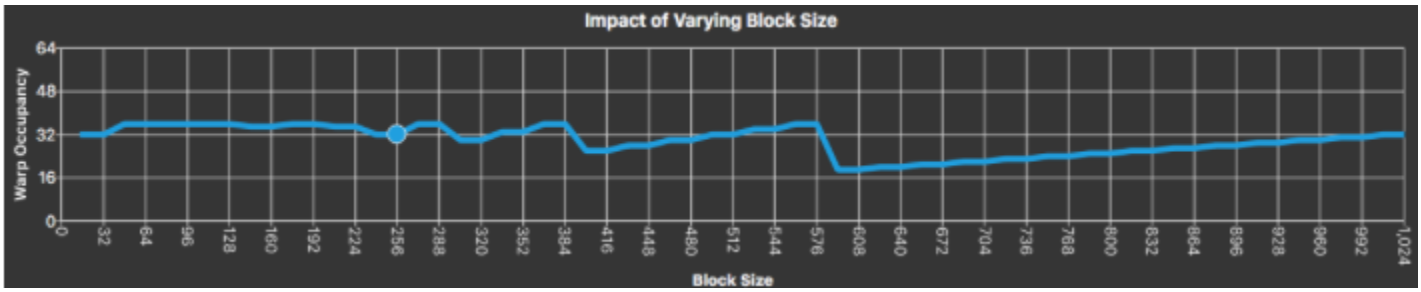


Figure 18. Looking at the above graph, we see that increasing the block size from 256 to 288 will increase the number of warps. Since we are already achieving close to the theoretical max warp occupancy, this does not seem like a priority.

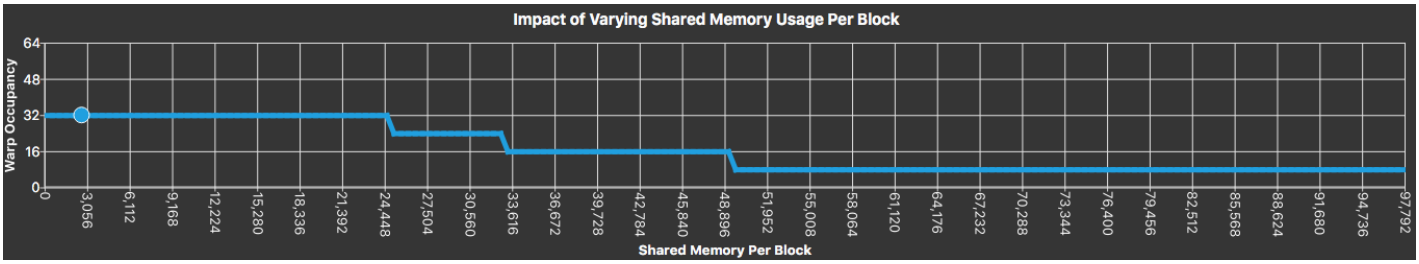


Figure 19. From the graph, we see that we are achieving the greatest warp occupancy by using 2,656 bytes of shared memory per block. To maintain this high warp occupancy, we should try not to use more than 24,448 bytes per block.

3.6.4 Scheduler Statistics

According to an application analysis done by NSight Compute:

- "Every scheduler is capable of issuing one instruction per cycle, but for this kernel each scheduler only issues an instruction every 8.9 cycles. This might leave hardware resources underutilized and may lead to less optimal performance. Out of the maximum of 16 warps per scheduler, this kernel allocates an average of 7.71 active warps per scheduler, but only an average of 0.13 warps were eligible per cycle. Eligible warps are the subset of active warps that are ready to issue their next instruction. Every cycle with no eligible warp results in no instruction being issued and the issue slot remains unused. To increase the number of eligible warps either increase the number of active warps or reduce the time the active warps are stalled."
- This kernel achieved 7.71 warps out of a theoretical 16 warps scheduled time period
- 0.13 warps were eligible per scheduler, and 0.11 were issued
  - This shows we executed eligible warps fairly well in this scenario
- 88.83% of the time, 0 warps were eligible to execute
- Conversely, 11.17% of the time, one or more warps were eligible to execute

3.6.5 Warp State Statistics

According to an application analysis done by NSight Compute:

- "On average each warp of this kernel spends 46.2 cycles being stalled waiting for sibling warps at a CTA barrier. This represents about 67.0% of the total average of 69.0 cycles between issuing two instructions. A high number of warps waiting at a barrier is commonly caused by diverging code paths before a barrier that causes some warps to wait a long time until other warps reach the synchronization point. Whenever possible try to divide up the work into blocks of uniform workloads. Use the Source View's sampling columns to identify which barrier instruction causes the most stalls and optimize the code executed before that synchronization point first."
- "Instructions are executed in warps, which are groups of 32 threads. Optimal instruction throughput is achieved if all 32 threads of a warp execute the same instruction. The chosen launch configuration, early thread completion, and divergent flow control can significantly lower the number of active threads in a warp per cycle. This kernel achieves an average of 16.9 threads being active per cycle. This is further reduced to 16.0 threads per warp due to predication. The compiler may use predication to avoid an actual branch. Instead, all instructions are scheduled, but a per-thread condition code or predicate controls which threads execute the instructions. Try to avoid different execution paths within a warp when possible. In addition, assure your kernel makes use of Independent Thread Scheduling, which allows a warp to reconverge after a data-dependent conditional block by explicitly calling __syncwarp()."
- 69.00 Warp Cycles occurred per issued instruction
- 16.88 was the Average Number of Active Threads per Warp
  - According to NSight Compute, this number is caused by and if that is reduced or eliminated, this kernel will achieve a more optimal number of threads per warp.

4. NVPROF COMPARISON

Previously, we had generated an analysis report using NVProf. This analysis was performed using a grid size of 240 x 272 x 224, and 10 time steps on the Mid-November Figure of Merit (FOM) run setup. The Traveling Wave Electron Acceleration (TWEAC) science case used in this run, is a representative science case for PIConGPU. This execution can be used for baseline analysis on AMD MI50/ MI60 systems. Our initial thoughts are that we should see similar, if not the same results for metrics, we saw using NVProf. We point out similarities and differences found across both applications for the three kernels below.

4.1 SIMILARITIES

Across all kernels, the GPU Utilization percentages and the impact of varying block size figures and numbers calculated by NVProf and NSight Compute were the same across all kernels. Below we highlight kernel-specific similarities.

4.1.1 Compute Current Kernel

In the Compute Current kernel, the Occupancy numbers calculated by NVProf and NSight Compute were identical. Additionally, the impact of varying shared memory usage per block figures and numbers were

the same.

4.1.2 Move And Mark Kernel

Both NVProf and NSight Compute stated the Move And Mark kernel's performance is bound by memory bandwidth. NSight Compute did not explicitly state this ("The kernel is utilizing greater than 80.0% of the available compute or memory performance of the device."), it seems reasonably clear that this is implied by looking at the charts for this kernel. The numbers calculated for Occupancy were virtually the same across applications. We did notice that NVProf rounded these numbers.

4.1.3 Shift Particles Kernel

Both NVProf and NSight Compute stated the Shift Particles Kernel's performance was bound by Instruction and Memory Latency.

4.2 DIFFERENCES

The differences we saw between the NVProf analysis and NSight Compute analysis were all kernel-specific. We go over those differences below.

4.2.1 Compute Current Kernel

NVProf stated the Compute Current kernel's performance is bound by Instruction and Memory Latency. NSight Compute did not explicitly mention instruction and memory latency. Regarding memory it did say to check memory replay (coalescing) metrics and consider whether it is possible to do more work per memory access (kernel fusion) or whether there are values we can (re)compute.

4.2.2 Move And Mark Kernel

The impact of varying shared memory usage per block graphs were the same, however the amount of warp occupancy we were achieving per shared memory differed. NVProf stated we were achieving 24 warps at 27,000 Bytes of Shared Memory per block while NSight Compute said we were achieving 32 warps using 0 Bytes of Shared Memory per block.

4.2.3 Shift Particles Kernel

The Occupancy numbers across applications differed slightly. In NVProf, the Achieved Active Warps is 30.59 vs. 30.67 for NSight Compute. The same was seen for Achieved Occupancy where NVProf showed the Achieved Occupancy as 47.8% vs. 47.92% for NSight Compute. Additionally the impact of varying shared memory usage per block graphs were the same, however the amount of warp occupancy we were achieving per shared memory differed. NVProf stated we were achieving 32 warps at 2,000 Bytes of Shared Memory per block while NSight Compute said we were achieving 32 warps using 0 Bytes of Shared Memory per block. Finally, NVProf showed that this kernel exhibits low compute throughput and memory bandwidth utilization relative to the peak performance of this device. Achieved compute throughput and/or memory bandwidth below 60.0% of peak typically indicate latency issues. However, NSight Compute did not offer any message similar to this.

5. POSSIBLE PROFILING TOOL ENHANCEMENTS

From comparing the findings of the profiling reports generated by both applications, we propose the following as possible enhancements for NSight Compute and NSight Systems.

5.1 NSIGHT COMPUTE

- For the GPU Utilization graphs, break down the utilization percentage by operation (memory, control-flow, arithmetic, etc.) like NVVP did. Below, we provide an example of what NVProf's GPU Utilization graph looked like. The other enhancement we would like to see with NSight Compute is a way to explicitly navigate to the application's documentation (5) within the application itself. Specifically, it would prove helpful to have each metric listed in a given report hyperlink to where it appears in the documentation or link to a documentation section within NSight Compute. As of the the time of writing, the documentation can be accessed through the Help menu by selecting either "Documentation" or "Documentation (Local)". By selecting "Documentation", a web browser will open the documentation for NSight Compute. If "Documentation (Local)" is selected, an HTML page containing the NSight Compute documentation is pulled up from the local machine. Currently it is only possible to view the profiling tool's documentation via an external source. We think providing documentation internal to the application would lead to easier analysis of the reports generated and would help the tool adopt a higher user-base.

5.2 NSIGHT SYSTEMS

For NSight Systems, we propose the following enhancements:

- Within PIConGPU, a typical kernel name has over 10,000 characters and knowing which kernel is which is crucial for performance analysis. If possible, we want to have the ability for the full name of the kernel to pop up when hovering over the kernels in the aggregate stream total (All Streams). Currently, the full name of the kernel is available after expanding the individual streams, but is not available for the aggregate stream total. There is a workaround for viewing the full kernel name in under All Streams in the latest version of NSight Systems. We additionally suggest adding an option where the user can add multiple 'sed' regex rules to describe how to extract a useful kernel name out of the mangled name.
- For templated kernel instantiations, the ability to limit the call stack trace would be useful. Currently the entire instantiation trace is listed in the popup which is not that useful.
- Additionally, we want the ability to reset the screen to the default view after fully extending the kernel names view, or have the kernel names text wrap. Currently, if the kernel names view is fully extended to the right, there's no way to view the timeline again without restarting the application. Finally, we want to be able to infinitely extend the width of the window. Right now, after extending it so far, the application turns black.

References

[1] 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, "Radiative signatures of the relativistic kelvin-helmholtz instability," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. New York, NY, USA: ACM, 2013, pp. 5:1–5:12. [Online]. Available: http://doi.acm.org/10.1145/2503210.2504564

[2] "Frontier Center for Accelerated Application Readiness (CAAR)," 2019. [Online]. Available: https://www.olcf.ornl.gov/caar/frontier-caar/

[3] "The Top500 List," 2020. [Online]. Available: https://www.top500.org/

[4] A. Debus, R. Pausch, A. Huebl, K. Steiniger, R. Widera, T. E. Cowan, U. Schramm, and M. Bussmann, "Circumventing the dephasing and depletion limits of laser-wakefield acceleration," *Phys. Rev. X*, vol. 9, p. 031044, Sep 2019. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevX.9.031044

[5] "NSight Compute Metrics Guide," 2020. [Online]. Available: https://docs.nvidia.com/nsight-compute/ProfilingGuide/index.html#metrics-guide