Verified Instruction-Level Energy Consumption Measurement for NVIDIA GPUs

Yehia Arafa∗, Ammar ElWazir†, Abdelrahman ElKanishy*, Youssef Aly§, Ayatelrahman Elsayed†, Abdel-Hameed Badawy∗*, Gopinath Chennupati†, Stephan Eidenbenz†, and Nandakishore Santhi‡

Abstract—Graphics processor units (GPUs) are prevalent in modern computing systems at all scales. They consume a significant fraction of the energy in these systems. However, vendors do not publish the actual cost of the power/energy overhead of their internal microarchitecture. In this paper, we accurately measure the energy consumption of various instructions found in modern NVIDIA GPUs. We provide an exhaustive comparison of more than 40 instructions for four high-end NVIDIA GPUs from four different generations (Maxwell, Pascal, Volta, and Turing). Furthermore, we show the effect of the CUDA compiler optimizations on the energy consumption of each instruction. We use three different software techniques to read the GPU on-chip power sensors, which use NVIDIA’s NVML API and provide an in-depth comparison between these techniques. Additionally, we verified the software measurement techniques against a custom-designed hardware power measurement. The results show that Volta GPUs have the best energy efficiency of all the other generations for different categories of instructions. This work should give GPU architects and developers more concrete understanding of these representative NVIDIA GPUs’ microarchitecture. It should also make energy measurements of any GPU kernel both efficient and accurate.

Index Terms—GPU Power, NVML, PAPI, External Power Meters, Onboard Sensors, PTX, CUDA Optimizations

I. INTRODUCTION

Applications that rely on graphics processor units (GPUs) have increased exponentially over the last decade, mainly due to the need for higher compute resources to perform general-purpose calculations. Thus, general-purpose graphics processor units (GPGPUs) commonly appear in different computing systems. The recent TOP500 list [1] shows that more than a third of the 153 systems debuting on the list are GPU-accelerated. Adding powerful devices such as GPUs has lead to an increase in these systems’ overall performance. But this increase in performance has also led to a dramatic increase in their electrical power consumption. Furthermore, the top supercomputer in the Green500 machines [2] is the DGX SaturnV (NVIDIA built). SaturnV has a power efficiency of 15 GFlops/watts and consumes 97 kW compared to 10,096 kW for Summit, the most powerful supercomputer on the Top500 list.

Power consumption is now a primary metric for evaluating systems performance. Especially with the development of embedded/integrated GPUs and their application in edge/mobile computation. Researchers have shown that large power consumption has a significant effect on reliability GPUs [3]. Also, with the ongoing increase in the number of GPU cores, GPU power management is crucial [4], [5]. Hence, analyzing and predicting the power usage of the GPGPU’s hardware components remains an active area of research. Several monitoring systems (hardware & software) have been proposed in the literature to estimate the total power usage of GPUs [6]–[9]. However, estimating energy consumption of the GPGPU’s internal hardware component is particularly challenging as the percentage of updates in the microarchitecture can be significant from one generation to another. Moreover, the dominant vendor in the market, NVIDIA, has never published the data on the actual energy cost of their GPUs’ microarchitecture.

In this paper, we accurately measure the energy consumption of almost all the instructions that can execute on modern NVIDIA GPGPUs. We run specially-designed micro-benchmarks on the GPU, monitor the change in the GPU’s power usage, and compute the energy for each instruction. Since the optimizations provided by the CUDA NVCC compiler [10] can affect the latency of each instruction [11], we show the effect of the CUDA compiler’s high-level optimizations on the energy consumption of each instruction. We used NVIDIA’s assembly-like language, Parallel Thread Execution (PTX) [12], in writing these micro-benchmarks. With the machine independent PTX, we have control over the exact sequence of instructions executing in the code. Thus, the measurement technique introduced has minimum overhead and is portable across different architectures/generations.

The results show that Volta GPUs have the best energy efficiency among all the other generations for different categories of the instructions. On the other hand, Maxwell and Turing GPUs are power-hungry devices.

To compute the energy consumption, we use three different software techniques based on the NVIDIA Management Library (NVML) [6], which query the onboard sensors and read the power usage of the device. We implement two methods using the native NVML API, which we call the sampling monitoring approach (SMA), and Multi-Threaded...
Synchronized Monitoring (MTSM). The third technique uses the newly released CUDA component in the PAPI v.5.7.1 [13] API. Furthermore, we designed a hardware system to measure the power usage of the GPUs in real-time. The hardware measurement are considered as the ground truth to verify the different software measurement techniques. We used the hardware setup on Volta TITAN V GPU.

We compare the results of using MTSM and PAPI software techniques to the hardware measurement for each instruction. The comparison shows that the MTSM technique leads to the best results since it integrates the power readings and captures the start and the end of a kernel correctly.

To the best of our knowledge, we are the first to provide a comprehensive comparison of the energy consumption of each instruction (more than 40 instructions) in modern high-end NVIDIA GPGPUs. Furthermore, the compiler optimizations effect on the energy consumption of each instruction has not been explored before in the literature. Also, we are the first to provide an in-depth comparison between different NVML power monitoring software techniques.

In summary, the followings are the contributions of this paper:

1) Accurate measurement of the energy consumption of almost all PTX instructions for four high-end NVIDIA GPGPUs from four different generations (Maxwell, Pascal, Volta, and Turing).
2) Show the effect of CUDA compiler optimizations levels on the energy consumption of each instruction.
3) Utilize three different software techniques to measure GPU kernels’ energy consumption.
4) Verified the different software techniques against a custom-designed in-house hardware power measurement on the Volta TITAN V GPU.

The rest of this paper is organized as follows: Section II provide a brief background on NVIDIA GPUs’ internal architecture. Section III describes the micro-benchmarks used in the analysis. Section IV depicts the differences between the three software techniques. While Section V shows the in-house direct hardware power measurement design. In Section VI we present the results. Section VII shows the related work and finally, Section VIII concludes the paper.

II. GPGPUs Architecture

GPUs consist of a large number of processors called Streaming Multiprocessor (SMX) in CUDA [14] terminology as shown in figure 1. These processors are mainly responsible for the computation part. They have several scalar cores, which has some computational resources, including fully pipelined integer Arithmetic Units (ALUs) for performing 32-bit integer instruction. Floating-Point units (FPU32) for performing floating-point operations, and Double-Precision Units (DPU) for 64-bit computations. Also, it includes Special Function Units (SPU) that executes intrinsic instructions, and Load and Store units (LD/ST) for calculations of source and destination memory addresses. In addition to the computational resources, each SMX is coupled with a certain number of warp schedulers, instruction dispatch units, instruction buffer(s), along with texture and shared memory units. Either each SMX has a private L1 memory, and all the SMXs share an L2 shared memory to cache the global addresses. The exact number of SMXs on each GPU varies with the generation and the computational capabilities of the GPU.

GPU applications typically consist of one or more kernels that can run on the device. All threads from the same kernel are grouped into a grid. The grid is made up of many blocks; each is composed of groups of 32 threads called warps. Grids and blocks represent a logical view of the thread hierarchy of a CUDA kernel. Warps execute instructions in a SIMD manner, meaning that all threads from the same warp execute the same instruction at any given time.

III. PTX Microbenchmarks

We designed special microbenchmarks to stress the GPU and expose its hidden characteristics to be able to capture the power usage of each instruction correctly.

We used Parallel-Thread Execution (PTX) [12] to write the micro-benchmarks. PTX is a virtual-assembly language used in NVIDIA’s CUDA programming environment. PTX provides an open-source machine-independent ISA. The PTX ISA itself does not run on the device but rather gets translated to another machine-dependent ISA named Source And Assembly (SASS). SASS is not open. NVIDIA does not allow writing native SASS instructions, unlike PTX, which provides a stable programming model for developers. There have been some research efforts [15], [16] to produce assembly tool-chains by reverse engineering and disassembling the SASS format to achieve better performance. Reading the SASS instructions can be done using CUDA binary utilities (cuobjdump) [17]. The use of PTX helps control the exact sequence of instructions executing without any overhead. Since PTX is a machine-independent, the code is portable across different CUDA runtimes and GPUs.
They executed the instruction only once, and read the clk presented a similar technique to find the instruction latency. For the unsigned div shows an example of the micro-benchmark of the unsigned div instructions which has the same header and the same name inside the SASS, used to transform the instrumented machine-independent PTX instructions and put that in a CUDA binary file (.cubin). This binary file is used to produce a fatbinary file, which gets embedded in the host C/C++ code. An empty kernel gets initialized in the host code, which is then gets replaced by the instrumented PTX kernel, which has the same header and the same name inside the (.fatbin.c). The kernel is executed with one block one thread.

Figure 2 shows an example of the instrumented PTX kernel for the unsigned Div instruction. Recently, Arafa et al. [11] presented a similar technique to find the instruction latency. They executed the instruction only once, and red the clk register before and after its execution. The design here is different since we need to capture the change in power usage, which would be unnoticeable if we execute the instruction only once. The key idea here is unrolling a loop and execute the same instruction millions of times and record the power then divide by the number of instructions to get the power consumption of the single instruction. The kernel in Figure 3 shows an example of the micro-benchmark of the unsigned div instruction. We begin by initializing the used registers, lines [13–27], is composed of 5 back-to-back unsigned div instructions with dependencies, to make sure that the compiler does not optimize any of them. We do a load-add-store operation on the output of the 5th div operation and begin the loop with new values each time to force the compiler to execute the instructions. Otherwise, the compiler would run the loop only the first time and squeeze the remaining iterations. We follow the same approach for all the instructions, and the kernel is the same, the only difference is the instruction itself.

We measure the energy of the kernel twice. First, with all the instructions and second, commenting out the instructions (lines [20–24]). We use Eq. 1 to calculate the energy of an instruction. Thus, we have eliminated both the steady-state power and any other overheads. Therefore, only the real energy of an instruction gets calculated.

\[ E_{\text{instruction}} = \frac{E_{\text{total}} - E_{\text{overhead}}}{\# \text{ of instructions}} \] (1)

A. NVCC Compiler Optimization

The kernel is compiled with (–O3) and without (–01) optimizations. This way, we capture the effect of the CUDA compiler’s higher levels of optimization flags on the energy consumption of each instruction. To make sure that in case of –O3, the compiler does not optimize the instructions and squeeze them, we performed three different validations of the code. First, we made sure that the output of the kernel is correct. Line 28 of Figure 3 stores the output of the loop. We read it and validate its correctness. Second, we validate the clk register for each instruction against the work of Arafa et al. [11]. Third, we inspect the SASS instructions. The loop bodies are small. The compiler unrolls the loop for –O3 and –O0. Furthermore, when commenting out the instructions, we dump the SASS code and verify that the difference is only the commented out instructions.

IV. SOFTWARE MEASUREMENT

NVIDIA provides an API named NVIDIA Management Library (NVML) [6], which offers direct access to the queries exposed via the command line utility, the NVIDIA System Management Interface or nvidia-smi. NVML allows the developers to query GPU device states such as GPU utilization, clock rates, GPU temperature etc. Additionally, it provides access to the board power draw by querying its instantaneous onboard sensors. The community has widely used NVML since its first release with the release of CUDA v4.1 in 2011. NVML comes with the NVIDIA display driver, and the SDK offers the API for its use.

We use NVML to read the device power usage while running the PTX micro-benchmarks and compute the energy use. Still, in the real SASS assembly, the number of registers is limited and will vary from one generation/architecture to another. When the limit exceeds, register variables will be spilled to memory, causing changes in performance. Line [10] sets the loop count to 1M iterations. The loop body, lines [13–27], is composed of 5 back-to-back unsigned div instructions with dependencies, to make sure that the compiler does not optimize any of them. We do a load-add-store operation on the output of the 5th div operation and begin the loop with new values each time to force the compiler to execute the instructions. Otherwise, the compiler would run the loop only the first time and squeeze the remaining iterations. We follow the same approach for all the instructions, and the kernel is the same, the only difference is the instruction itself.
of each instruction. There are several techniques for collecting power usage using NVML. We found that the methods do vary. Therefore, we provide an in-depth comparison of these techniques on the energy of the individual instructions.

A. Sampling Monitoring Approach (SMA)

The C-based API provided by NVML can query the power usage of the device and provide the instantaneous power measurement. Therefore, it can be programmed to keep reading the hardware sensor with a certain frequency. This basic approach is popular and was used in other related works [18]–[20]. The `nvmlDeviceGetPowerUsage()` function is used to retrieve the power usage reading for the device, in milliwatts. This function is called and executed by the CPU. We configured the sampling frequency of reading the hardware sensors to its maximum, 66.7 Hz [19] (15 ms window between each call to the function).

We read the power sensor according to the sample interval in the background while the micro-benchmarks are running. Example of the output using this approach are shown in Figures 4(a) and 4(b). The two figures show the power consumption over time for integer Add and unsigned integer Div kernels in the TITAN V (Volta) GPU. The power usage jumps shortly after the launch of the kernel and decreases in steps after the kernel finishes execution until it reaches the steady-state. This is done in 22 sec and 33 sec windows for Add and Div respectively. If we calculate the two kernels actual elapsed time, it takes only 0.28 sec and 13 sec for the Add and the Div kernels, respectively. That is, the GPU does something before and after the actual kernel execution. Hence, identifying the window of the kernel is hard and would affect the output as the power consumption varies through time. One solution is to take the maximum reading between the two steady states, but this would be misleading for some kernels, especially the bigger ones. Therefore, we ignore this approach from reporting owing to these issues.

B. PAPI API

Performance Application Programming Interface (PAPI) [13] provides an API to access the hardware performance counters found on most modern processors. We can get different performance metrics through either a simple programming interface from either C or Fortran programming languages. Researchers have used PAPI as a performance and power monitoring library for different hardware and software components [21]–[25]. It is also used as a middleware component in different profiling and tracing tools [26].

PAPI can work as a high-level wrapper for different components; for example, it uses the Intel RAPL interface [27] to report the power usage and energy consumption for Intel CPUs. Recently, PAPI version 5.7 added the NVML component, which supports both measuring and capping power usage on modern NVIDIA GPU architectures. It is essential to mention that installing PAPI with NVML is a tedious task and not straight forward.

The advantage of using PAPI is that the measurements are by default synchronized with the kernel execution. The target kernel is invoked between the `papi_start`, and the `papi_end` functions, and a single number, representing the power event
we need to measure, is returned. The NVML component implemented in PAPI uses the function, `getPowerUsage()` which query `nvmlDeviceGetPowerUsage()` function. According to the documentation, this function is called only once when the `papi_end` is called. Thus, the power returned using this method is an instantaneous power when the kernel finishes execution. Although synchronizing with the kernel solves the SMA issues, taking the instantaneous measurement when the kernel finishes execution can provide non-accurate results especially, for large and irregular kernels as shown in Section VI. Note that PAPI provides an example that works like the SMA approach, which we refrain from this paper.

C. Multi-Threaded Synchronized Monitoring (MTSM)

In MTSM, we identify the exact window of the kernel execution. We modified the SMA to synchronize the kernel execution. This way, only the power readings of the kernel are recorded. Since the host CPU monitors the NVML API, we use Pthreads for synchronization where one thread calls and monitors the kernel while the other thread records the power.

Algorithm 1 shows the MTSM. We initialize a volatile atomic variable (`flag`) to zero, which we use later to record the power readings according to the start and end of the target kernel. On line 6 we create a new thread (`th1`) which executes a function (`func1`) [line 17] in parallel. This function completes the power monitoring, depending on the atomic `flag`. This uses the NVML function, `nvmlDeviceGetPowerUsage()` which returns the device power in milli-watts. The readings of the power during the kernel window are recorded and saved in an array (`power_readings`), which is used later in computing the kernel energy. In lines [7–12], flip the `flag` value and start computing the elapsed time and the launch kernel, which means starting the power monitoring. At the end of the kernel execution, we record the elapsed time and change the flag. We use the CUDA synchronize function to make sure that the power is recorded correctly. We do not specify any reading sampling frequency for the NVML functions. Although this would give us redundant values, it would be more accurate. With this setup, we found that the power reading frequency is nearly 2kHz.

Figures 5(a) and 5(b) show the corresponding kernels in Figures 4(a) and 4(b) after identifying the exact kernel execution window. The new graphs are annotated with the start and end of the kernel. We observe that the kernel does not start after the sudden rise in the power from the steady-state, rather after a couple of ms from this sudden increase in power consumption (see `add` kernel in Figure 5(a) for clarity). After the kernel finishes execution, the power remains high for a small-time,
and then it starts descending in steps until it reaches the steady-state again. To compute the kernel’s energy, we calculated the area under the curve for the kernel using Eq. 2. We believe that this approach would provide the most accurate measurement since the power readings of only the kernel are recorded. Computing the energy as the area under the curve is more rigorous than just taking the last power reading multiplied by the time elapsed for the kernel, as is done in PAPI.

\[
E_{(mj)} = \frac{\text{elapsed time (sec)}}{\text{# of power readings (N)}} \times \sum_{i=0}^{N-1} \text{Power} (i) \text{(mev)} \quad (2)
\]

V. HARDWARE MEASUREMENT

GPUs drain power as leakage power and dynamic power. The leakage power is a constant power that the GPU consumes to maintain its operation. However, dynamic power is affected by the kernel’s instructions and operations. We take into account the two power components in this study.

The modern graphics cards have two primary power sources. The first power source is the direct DC power (12V) supply, provided through the side of the card. The second one is the PCI-E (3.3V and 12V) power, provided through the motherboard, we have designed a system to measure each power source in real-time. The hardware measurement are considered as the ground truth to verify the different software measurement techniques.

Figure 6 shows the experimental hardware setup with all the components. To capture the total power, we measure the current and voltage for each power source simultaneously. A clamp meter and a shunt series resistor are used for the current measurement. For voltage measurement, we use a direct probe on the voltage line using an oscilloscope to acquire the signals. Equation 3 is used to calculate the total hardware power drained by the GPU from the two different power sources.

**Direct DC Power Supply Source:** Power supply provides a 12V voltage through a direct wired link. We use both a 6-pin and 8-pin PCI-E power connectors to deliver a maximum of 300W. Thus, the direct DC power supply source is the main contributor to the card’s power. Figure 6 shows a clamp meter measuring the current of the direct power supply connection. The voltage of the power supply is measured using an oscilloscope probe. The current and voltage are acquired using an oscilloscope, as shown in Figure 6. Therefore, the Direct DC power supply source is calculated using simple multiplication. The third addition term in Eq. 3 shows the calculation of the power which is multiplying \( I_{clamp} \) by \( V_{DPS} \). In which, \( V_{DPS} \) is the voltage of the direct power supply.

**PCI-E Power Source:** Graphics cards are connected to the motherboard through the PCI-E x16 slot connection. 3.3V and 12V voltages are provided through this slot. To accurately measure the power that goes through this slot, an intermediate power sensing technique should be installed between the card and the motherboard. We designed a custom made PCI-E riser board that measures the power supplied through the riser board.

Two in-series shunt resistors are used as a power sensing technique. As shown in Figure 7, each shunt resistor \( R_S \) is connected in series with \( V_{S1} \) and \( V_{G1} \) which are across \( R_S \) using oscilloscope. We then divide it with the \( R_S \) value. The voltage level is measured using the riser board. We duplicate the same calculation technique for the 3.3V voltage level, as shown in Eq. 5

\[
P_{HW} = \frac{V_{S1} - V_{g1}}{R_s} \times V_{g1} + \frac{V_{S2} - V_{g2}}{R_s} \times V_{g2} + I_{clamp} \times V_{DPS} \quad (3)
\]
VI. Results

We run each instruction found in the latest PTX version, 6.4 [12], and show its energy consumption. We report the results of using MTSM and PAPI on four different NVIDIA GPGPUs from four different generations/architectures:

- **GTX TITAN X**: A GPU from Maxwell architecture [28] with a compute capability of 5.2. It has 3584 cores that run on 151 MHz clock frequency.
- **GTX 1080 Ti**: A GPU from Pascal architecture [29] with a compute capability of 6.1. It has 3584 cores that run on 1481 MHz clock frequency.
- **TITAN V**: A GPU from Volta architecture [30] with a compute capability of 7.0. It has 5120 cores that run on 1200 MHz clock frequency.
- **TITAN RTX**: A GPU from Turing architecture [31] with a compute capability of 7.5. It has 4608 cores that run on 1350 MHz clock frequency.

We used CUDA NVCC compiler version 10.1 [10] to compile and run the codes. CUDA compiler comes equipped with the C-based programmatic interface for monitoring and managing different GPU states and NVML library [6].

Table I shows an enumeration of the energy consumption of the various ALU instructions for the different GPUs. For simplicity, we used each GPU generation to refer to the GPUs. We denote the (–O3) version as Optimized and the (–O0) version as Non-Optimized.

Overall Volta GPUs have the lowest energy consumption per instruction among all the tested GPUs. Pascal preceded the Volta while Maxwell and Turing are very power hungry devices except for some categories of the instructions. Furthermore, Maxwell has very high energy consumption in Floating Single and Double Precision instructions.

In Half Precision (FP16) instructions, Volta and Turing have much better results than Pascal. Hence, this confirms that both architectures are suitable for approximate computing applications (e.g., deep learning and energy-saving computing). We did not run FP16 instructions on Maxwell as Pascal architecture was the first GPU that offered FP16 support. The same trend can be found in Multi Precision (MP) instructions where Volta and Pascal have better energy consumption compared to the other two generations. MP [32] instructions are essential in a wide variety of algorithms in computational mathematics (i.e., number theory, random matrix problems, experimental mathematics). Also, it is used in cryptography algorithms and security.

Overall, the energy of Non-Optimized is always more than the Optimized. One reason is that the number of cycles at the O0 optimization level are more than the O3 level [11]. Thus, the instruction takes more time to finish execution.

PAPI vs. MTSM: The dominant tendency of the results is that PAPI readings are always more than the MTSM. Although the difference is not significant for small kernels, it can be up to 1 μj for bigger kernels like Floating Single and Double Precision div instructions.

A. Verification against the Hardware Measurement

Since Volta GPUs are the primary GPUs in data-centers. We verified the different software techniques (MTSM & PAPI) against the hardware setup on the Volta TITAN V GPU.

Compared to the ground truth hardware measurement, for all the instructions (each run ten times), the average Mean Absolute Percentage Error (MAPE) of MTSM Energy is 6.39, while the average Root Mean Square Error (RMSE) is 3.97. In contrast, PAPI average MAPE is 10.24 and the average RMSE is 5.04. Figure 8 shows the error of MTSM and PAPI relative to the hardware measurement for some of the instructions. The verification’s results prove that MTSM is more accurate than PAPI as it is closer to what has been measured using the hardware.

VII. Related Work

In this section, we discuss some of the related work. Additional details can be found in [33]. Several works in the literature have proposed to analyze and estimate the energy consumption and the power usage of GPUs. Researchers have proposed several analytical methods [18], [34], [35] to indirectly model and predict the total GPU's kernel power/energy. Likewise, several methods rely on cycle-accurate simulators [36].

The power/energy measurement can be carried out in two different approaches, a software-oriented solution, where the internal power sensors are queried using NVML, and the hardware-oriented solutions using an external hardware setup.

Software-oriented approaches: Arunkumar et al. [18] used a direct NVML sampling motoring approach running in the background while using a special micro-benchmark to calculate basic compute/memory instructions energy consumption and feed that to their model. They run their evaluation on (Tesla K40) GPU. They intentionally disabled all compiler optimizations and compiled their micro-benchmark with S-O0 flag. Hence, they do not report any results for energy consumption if the optimization flags are working. Burtscher et al. [19] analyzed the power reported by NVML for (Tesla K20) GPU.

Hardware-oriented approaches: Zhao et al. [37] used an external power meter on an old GPU from Fermī [39]
| Instruction | \( s \) | \( u \) |
|------------|------|------|
| add / sub / min / max | \( .0942, .0461 \) | \( .0277, .0200 \) |
| mul / mad | \( .0206, .1789 \) | \( .1876, .1849 \) |
| {s} div | \( 10.5921 \) | \( 6.2983 \) |
| {s} rem | \( 7.8512 \) | \( 6.3597 \) |
| abs | \( .800, .747 \) | \( .2927, .2349 \) |
| {u} div | \( 7.44783 \) | \( 6.2899 \) |
| {u} rem | \( 7.5357 \) | \( 6.4006 \) |

(2) Logic and Shift Instructions

| Instruction | \( s \) | \( u \) |
|------------|------|------|
| and / or / not / xor | \( .3622, .0343 \) | \( .3227, .2423 \) |
| cnot | \( .0942, .0461 \) | \( .0277, .0200 \) |
| shl / shr | \( .7778, .2008 \) | \( .0021, .0014 \) |
| {s} div | \( 10.6203 \) | \( 9.4351 \) |

(3) Floating Single Precision Instructions

| Instruction | \( s \) | \( u \) |
|------------|------|------|
| add / sub / min / max | \( .0206, .1789 \) | \( .1876, .1849 \) |
| mul / mad / fma | \( .3023, .2739 \) | \( .2778, .2008 \) |
| div | \( 50.3810 \) | \( .2899 \) |

(4) Double Precision Instructions

| Instruction | \( s \) | \( u \) |
|------------|------|------|
| add / sub / mul | \( NA \) | \( NA \) |
| subc | \( .6094, .3593 \) | \( .3655, .3599 \) |
| madc / madc | \( .11575 \) | \( .7697 \) |

(5) Half Precision Instructions

| Instruction | \( s \) | \( u \) |
|------------|------|------|
| rcp | \( 6.4492 \) | \( 5.3416 \) |
| sqrt | \( 6.3630 \) | \( 5.3923 \) |
| approx.sqrt | \( 0.8527 \) | \( 0.4961 \) |
| rsqrt | \( 0.5174 \) | \( 0.303 \) |
| sin / cos | \( 0.3410 \) | \( 0.1507 \) |
| lg2 | \( 0.5075 \) | \( 0.3098 \) |
| exp2 | \( 0.5147 \) | \( 0.3094 \) |
| copy | \( 0.2009 \) | \( 0.1700 \) |

(6) Special Mathematical Instructions

| Instruction | \( s \) | \( u \) |
|------------|------|------|
| mul24() / mad24() | \( 0.3915, .2939 \) | \( 0.2853, .2727 \) |
| sadi() | \( 0.0316, .015 \) | \( 0.2523, .1243 \) |
| popc() | \( 0.074, .057 \) | \( 0.1347, .2674 \) |
| clr() | \( 0.0729, .0479 \) | \( 0.2644, .3339 \) |
| blind() | \( 0.0488, .0374 \) | \( 0.2326, .3081 \) |

(7) Integer Intrinsics Instructions

### TABLE I

Energy Consumption of GPU Instructions. \( \{ s \} \) & \( \{ u \} \) denote signed and unsigned instructions respectively. The first number in the results is PAPI and the second one is the MTSM. All the numbers are in \( (\mu) \).
architecture, *(GeForce GTX 470)* where they designed a micro-benchmark to compute the energy of some PTX instructions and feed that into their model. The authors of *(38)* validates their roofline model by using PowerMon 2 *(9)* and a custom PCIe interposer to calculate the instantaneous power of *(GTX 580)* GPU.

Recently, Sen et al. *(40)* provided an assessment to rate the quality and performance of the power-profiling mechanisms using hardware and software techniques. They compared a hardware approach using PowerInsight *(8)*, a commercial product from Penguin Computing *(41)* to the software NVML approach on a developed matrix multiplication CUDA benchmark. Kasichayanula et al. *(42)* used NVML to calculate the energy consumption of some GPU units which drive their model and validate it with a Kill-A-Watt power meter. While these types of hardware power meters are cheap and straightforward to use, they do not give an accurate measurement, especially in HPC settings *(33)*.

In a similar spirit, we follow the same line of research. Nevertheless, we focus on the energy consumption of individual instructions and the effect of CUDA compiler optimizations on the instructions. We also provide an assessment of the quality of the different software techniques and verify software results to a custom in-house hardware setup.

**VIII. CONCLUSIONS**

In this paper, we accurately measure the energy consumption of various instructions that can be executed in modern NVIDIA GPUs. We also show the effect of different optimization levels found in the CUDA (NVCC) compiler on the energy consumption of each instruction. We provide an in-depth comparison of varying software techniques used to query the onboard internal GPU sensors and compare that to a custom-designed hardware power measurement. Overall, the paper provides an easy and straightforward way to measure the energy consumption of any NVIDIA GPU kernel. Additionally, our contributions will help modeling frameworks *(43)*, and simulators *(35)* have a more precise predictions of GPUs’ energy/power consumption.

**REFERENCES**

1. Top500. [Online]. Available: https://www.top500.org/

2. Green500. [Online]. Available: https://www.top500.org/green500/

3. L. B. Gomez, F. Cappello, L. Carro, N. DeBardeleben, B. Fang, S. Gurumurthi, K. Pattabiraman, P. Rech, and M. S. Reorda. “Gpppus: How to combine high computational power with high reliability,” in *Proceedings of the Conference on Design, Automation & Test in Europe*, ser. DATE 2014. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2014, pp. 341:1–341:9.

4. A. Pathania, Qing Jiao, A. Prakash, and T. Mitra, “Integrated cpu-gpu power management for 3d mobile games,” in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2014, pp. 1–6.

5. O. Kayiran, A. Jog, A. Pattnaik, R. Ausavarungnirun, X. Tang, M. Kandemir, G. Loh, O. Mutlu, and C. Das, “uc-states: Fine-grained cpu datapath power management,” in *Parallel Architectures and Compilation Techniques - Conference Proceedings*, PAC, pp. 17–30, 2016.

6. NVIDIA Management Library (NVML). 2019. [Online]. Available: https://docs.nvidia.com/pdf/NVMLAPIReferenceGuide.pdf

7. J. W. Romein and B. Veenboer, “Power sensor 2: A fast power measurement tool,” in *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, April 2018, pp. 111–113.

8. J. H. Laros, P. Pokorny, and D. DeBonis, “Powerinsight - a commodity power measurement capability,” in *2013 International Green Computing Conference*, June 2013, pp. 1–6.

9. D. Bedard, M. Y. Lim, R. Fowler, and A. Porterfield, “Powermon: Finegrained and integrated power monitoring for commodity computer systems,” in *Proceedings of the IEEE SoutheastCon 2010 (Southeast-Con)*, March 2010, pp. 479–484.

10. CUDAbench Drakk Drakk Driver (NVCC) v10.1, 2019. [Online]. Available: [https://docs.nvidia.com/cuda/pdf/CUDACompilerDriverNVCC.pdf](https://docs.nvidia.com/cuda/pdf/CUDACompilerDriverNVCC.pdf)

11. Y. Arafah, A. A. Badawy, G. Chemnupati, N. Santhi, and S. Eidenbenz, “Low overhead instruction latency characterization for nvidia gpgpus,” in *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, Sep. 2019, pp. 1–8.

12. Parallel Thread Execution ISA v6.4, 2019. [Online]. Available: [https://docs.nvidia.com/pdf/ptxisa6.4.pdf](https://docs.nvidia.com/pdf/ptxisa6.4.pdf)

13. Performance Application Programming Interface (PAPI), 2009. [Online]. Available: [https://icl.utk.edu/papi/index.html](https://icl.utk.edu/papi/index.html)

14. CUDAC Programming Guide, 2018. [Online]. Available: [https://docs.nvidia.com/cuda/archives/8.0/cuda-c-programming-guide/index.html](https://docs.nvidia.com/cuda/archives/8.0/cuda-c-programming-guide/index.html)

15. X. Zhang, G. Tan, S. Xue, J. Li, K. Zhou, and M. Chen, “Understanding the gpu microarchitecture to achieve bare-metal performance tuning,” in *Proceedings of the 22Nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP ’17. New York, NY, USA: ACM, 2017, pp. 31–43.

16. S. Gray, MaxAs: Assembler for NVIDIA Maxwell architecture, 2011. [Online]. Available: [https://github.com/NervanaSystems/maxas](https://github.com/NervanaSystems/maxas)

17. CUDA Binary Utilities, 2019. [Online]. Available: [https://docs.nvidia.com/cuda/pdf/CUDABinaryUtilities.pdf](https://docs.nvidia.com/cuda/pdf/CUDABinaryUtilities.pdf)

18. A. Arunkumar, E. Bolotin, D. Nellans, and C.-J. Wu, “Understanding the future of energy efficiency in multi-module gpus,” in *Proceedings - 25th IEEE International Symposium on High Performance Computer Architecture, HPCA 2019*, ser. HPCA ’19, pp. 519–532.

19. M. Burscher, I. Zecena, and Z. Zong, “Measuring gpu power with the k20 built-in sensor,” in *Proceedings of Workshop on General Purpose Processing Using GPUs*, ser. GPGPU-7, 2014, pp. 28:28–28:36.

20. Mariza Ferro et al., “Analysis of GPU Power Consumption Using Internal Sensor,” Zenodo July 2017. [Online]. Available: [http://doi.org/10.5281/zenodo.833347](http://doi.org/10.5281/zenodo.833347)

21. D. Terpstra, H. Jagode, H. You, and J. Dorangara, “Collecting performance data with papi-cc,” in *Tools for High Performance Computing*, 2010, pp. 157–173.

22. A. D. Malony, S. Biersdorff, S. Shende, H. Jagode, S. Tomov, G. Juckeland, R. Dietrich, D. Poole, and C. Lamb, “Parallel performance measurement of heterogeneous parallel systems with gpus,” in *2011 International Conference on Parallel Processing*, Sep. 2011, pp. 176–185.

23. V. M. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra, and S. Moore, “Measuring energy and power with papi,” in *2014 41st International Conference on Parallel Processing Workshops*, Sep. 2012, pp. 262–268.

24. H. McCraw, D. Terpstra, J. Dongarra, K. Davis, and R. Musselman, “Beyond the cpu: Hardware performance counter monitoring on blue gene/l,” in *Supercomputing*, 2013, pp. 213–225.

25. A. Haidar, H. Jagode, A. Yarkhan, P. Vacciaro, S. Tomov, and J. Dongarra, “Power-aware computing: Measurement, control, and performance analysis for intel xeon phi,” in *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, Sep. 2017, pp. 1–7.

26. A. Agelastos, B. Allan, J. Brandt, P. Cassella, J. Enos, J. Fulpol, A. Gentile, S. Monk, N. Nakshinhaboo, J. Ogden, M. Rajan, M. Showmerman, J. Stevenson, N. Taeat, and T. Tucker, “The lightweight distributed metric service: A scalable infrastructure for continuous monitoring of large scale computing systems and applications,” in *SC ’14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2014, pp. 154–165.

27. H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, “Rapl/memory power estimation and capping,” in *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*, Aug 2010, pp. 189–194.

28. NVIDIA Maxwell GPU Architecture, 2014. [Online]. Available: [https://international.download.nvidia.com/geforce-com/international/ pdfs/GeForce-GTX-750Ti-Whitepaper.pdf](https://international.download.nvidia.com/geforce-com/international/ pdfs/GeForce-GTX-750Ti-Whitepaper.pdf)
[29] NVIDIA Pascal GPU Architecture, 2016. [Online]. Available: https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf

[30] NVIDIA Volta GPU Architecture, 2017. [Online]. Available: https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf

[31] NVIDIA Turing GPU Architecture, 2018. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/

[32] N. Emmart, “A study of high performance multiple precision arithmetic on graphics processing units,” Ph.D. dissertation, UMASS, 2018. [Online]. Available: https://scholarworks.umass.edu/dissertations_2/1164

[33] R. A. Bridges, N. Imam, and T. M. Mintz, “Understanding gpu power: A survey of profiling, modeling, and simulation methods,” in ACM Comput. Surv., vol. 49, no. 3, pp. 41:1–41:27, Sep. 2016.

[34] X. Ma, M. Dong, L. Zhong, and Z. Deng, “Statistical power consumption analysis and modeling for gpu-based computing,” IEEE Micro, vol. 31, no. 2, p. 50–59, Mar. 2011.

[35] X. Ma, M. Dong, L. Zhong, and Z. Deng, “An integrated gpu power and performance model,” in Proceedings of the 37th Annual International Symposium on Computer Architecture, ser. ISCA ’10. New York, NY, USA: ACM, 2010, pp. 280–289.

[36] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M.Aamodt, and V. J. Reddi, “Gpuwatch: Enabling energy optimizations in gpgpus,” in Proceedings of the 40th Annual International Symposium on Computer Architecture, ser. ISCA ’13, 2013.

[37] Q. Zhao, H. Yang, Z. Luan, and D. Qian, “Poigem: A programming-oriented instruction level gpu energy model for cuda program,” in Proceedings of the 13th International Conference on Algorithms and Architectures for Parallel Processing, Springer, 2013, pp. 129–142

[38] C. M. Wittenbrink, E. Kilgariff, and A. Prabhui, “Fermi gf100 gpu architecture,” IEEE Micro, vol. 31, no. 2, p. 50–59, Mar. 2011.

[39] J. Choi, D. Bedard. R. Fowler, and R. Vuduc, “A roofline model of energy,” in 2013 IEEE 27th International Symposium on Parallel and Distributed Processing, May 2013, pp. 661–672.

[40] S. Sen, N. Imam, and C. Hsu, “Quality assessment of gpu power profiling mechanisms,” in 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), May 2018, pp. 702–711.

[41] Penguin Computing, 2012. [Online]. Available: https://www.penguincomputing.com/company/press-releases

[42] K. Kasichayanula, D. Terpstra, P. Luszczek, S. Tomov, S. Moore, and G. D. Peterson, “Power aware computing on gpus,” in 2012 Symposium on Application Accelerators in High Performance Computing, July 2012, pp. 64–73.

[43] Y. Arafa, A. A. Badawy, G. Chennupati, N. Santhi, and S. Eidenbenz, “Ppt-gpu: Scalable gpu performance modeling,” IEEE Computer Architecture Letter, vol. 18, no. 1, pp. 55–58, Jan. 2019.