Building Custom GPU Power Models With AccelWattch Project Report

Huy Dinh Tran
htran197@ucr.edu
University of California, Riverside

Daniel Wong
danwong@ucr.edu
University of California, Riverside

Abstract

Graphics Processing Units (GPUs) are the choice of accelerator that is widely utilized in data analytics and machine learning fields. Performance per watt is one of the most crucial evaluation metrics that determine the efficiency of a GPU. Performance and power models are equally important when researching and designing new technologies for GPU architects. However, while GPU performance modeling has seen great progression, power modeling has not been focused on to meet today's standards. Over the past years, researchers from Intel, Northwestern University, Purdue University, and University of British Columbia had developed and published an open-sourced power modeling framework named AccelWattch[4]. AccelWattch is an analytic model for modern GPUs that accurately estimated the constant, static, and dynamic power consumption. In this paper, we would be experimenting with creating our own power model of an NVIDIA Pascal GPU, the GTX 1050 Ti, using AccelWattch. The following sections will give a run-down procedure of the experiment which includes setting up validation benchmarks, hardware profiling for validation, running simulation with power model, and validating simulated estimated results with measured results from real GPU.

1 Introduction

1.1 Motivation

GPUs are widely used for many applications in data analytics and machine learning. The efficiency of a GPU is determined by the performance per watt. In designs, both performance and power models are important when researching and testing new technologies for new GPUs. While GPU performance modeling has progressed in great stride, power modeling has lacked behind and is not up-to-date with modern architecture. The previous state-of-art power modeling tool, GPUWattch, is outdated and does not accurately model modern GPU architecture. It is lack of cycle-level power model and is not able to capture constant and static power. To further aid the development of GPUs, researchers from Intel, Northwestern University, Purdue University, and University of British Columbia had developed and published an open-sourced power modeling framework named AccelWattch. AccelWattch is an analytic model for modern GPUs that accurately estimated the constant, static, and dynamic power consumption. AccelWattch was able to deliver promises that GPUWattch wasn't able to do with better accuracy and model modern technology better.

1.2 AccelWattch

AccelWattch is an analytic model that accurately captures constant, static, and dynamic power consumption in the presence of DVFS, thread divergence, intra-warp functional unit overlap, variable SM occupancy, and power gating. AccelWattch was specially designed for the NVIDIA Volta GPU architecture. AccelWattch resolves long-standing needs for modern GPU architectures: the lack of a cycle-level power model, and the inability to capture the constant and static power with existing methodologies. AccelWattch is integrated with GPGPU-Sim[1] and Accel-Sim[5] which enable it to be directed by emulation (PTX[6]) or trace-driven (SASS[7]) software performance models, by hardware performance counters, or by a combination of the above. AccelWattch can enable reliable design space exploration. Directly applying the Volta power model on a GPU configuration resembling the Pascal and Turing architectures result in accurate power models for these architectures without tuning specifically for them

1.3 Flowchart

Figure 1 shows the overall flowchart of how AccelWattch create a power model to estimate the power consumption of a GPU. There are three sections of modeling which are constant, static, and dynamic power modeling. Each of these sections follows the methodology to accurately model the power consumption. These will then create the power model which can be used for validation against real hardware power measurements.

1.4 Methodology

The methodology of AccelWattch consists of three modeling section which serves to estimate the constant, static, and dynamic power consumption. It takes the presence of DVFS, thread divergence, intra-warp functional unit overlap, variable SM occupancy, and power gating into account to accurately build a power model.

1.4.1 Constant Power Modeling. Figure 2 shows a high-level equation of the total power consumption of a GPU. Figure 3 shows a breakdown equation of Figure 2 with capacitance, voltage, and frequency. m and n are constant. However, while this equation can model older generations of

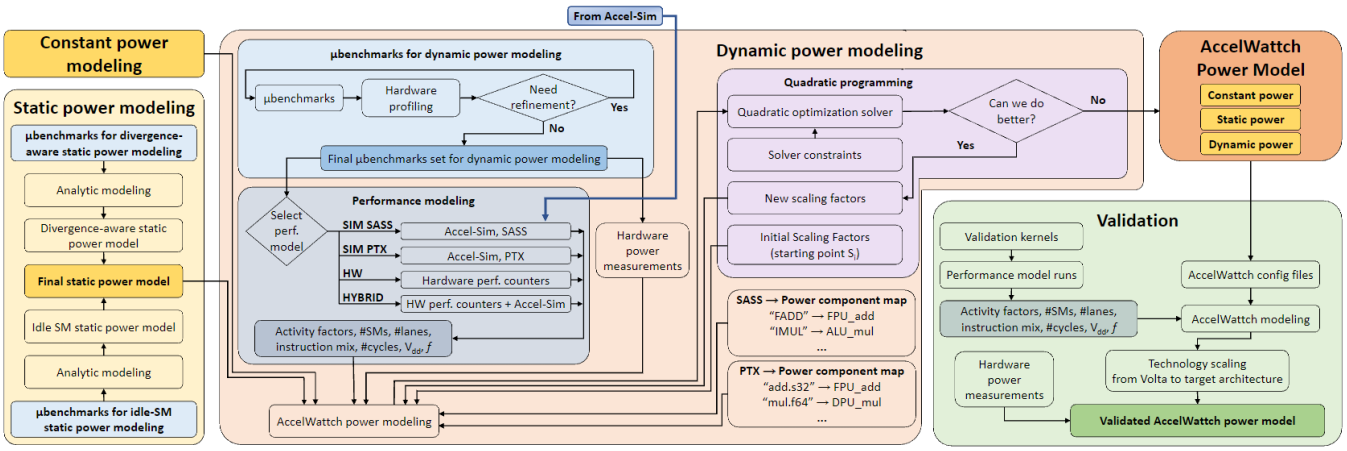


Figure 1. AccelWattch power modeling flowchart

$$P_{total} = P_{proc,\,dyn} + P_{mem,\,dyn} + P_{proc,\,static} + P_{mem,\,static} + P_{const}$$

Figure 2. High-level total power equation

$$P_{total} = \overbrace{mCV^2f}^{Dynamic} + \overbrace{nV}^{Static} + P_{const}$$

Figure 3. Older GPU high-level power equation

GPUs, recent technologies are built upon DVFS which will cause negative constant and static power estimations using this equation. To take DVFS into account, a new equation

$$P_{total} = \overbrace{\beta C f^3}^{Dynamic} + \underbrace{\tau f}_{Static} + P_{const}$$

Figure 4. Modern GPU high-level power equation

using recently-published data for fully-realized processors which better models GPUs with DVFS are shown in Figure 4. β and τ are constant.

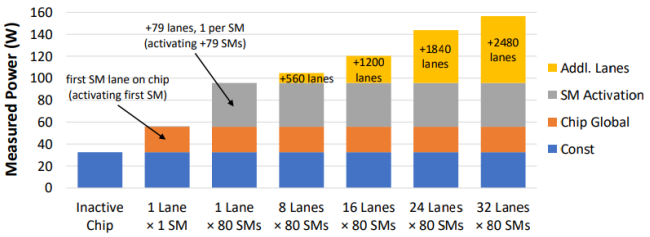


Figure 5. Inferring the power consumption of activating power-gated chip-wide and SM-wide components

1.4.2 Static Power Modeling. The static power modeling consists of three main awareness which are power gating, divergence, and idle SMs. For power gating, the model considers the impact on the power consumption of re-activating components on real hardware. It accurately captures the power consumption in the presence of power gating. For example, Figure 5 shows a chart that displays the power consumption if we just activate a local component (an SM with 1 lane), it will activate a shared global component (L2 cache) which results in a large amount of static power consumption.

$$P_{static,\,addLane} = \frac{P_{static,\,32Lanes} - P_{static,\,firstLane}}{31}$$

$$P_{static,\,yLanes} = \begin{cases} P_{static,\,firstLane} + P_{static,\,addLane} \cdot (y-1), & y \le 16 \\ P_{static,\,firstLane} + \frac{1}{2} P_{static,\,addLane} \cdot 15 + \frac{1}{2} P_{static,\,addLane} \cdot (y-17), & y > 16 \end{cases}$$

Figure 6. Warp-divergence based on number of active threads equation

For divergence, the model considers how a warp operates. In modern GPUs, a warp executes by running two 16-thread half-warps, one after the other. Figure 6 shows an equation that models the power consumption of divergence within a warp depending on the number of active threads. y is the number of active threads.

$$P_{perIdleSM} = \sqrt[n]{\prod_{i=1}^{n} \frac{P_{total,\,i} - P_{const} - \overbrace{\frac{P_{total,\,80SMs,\,i} - P_{const}}{80} \cdot N_{activeSM}}^{P_{dyn+static,perActiveSM,i}}}{N_{idleSMs}}}$$

Figure 7. Power consumption of per idle SM equation

For idle SMs, Figure 7 shows the power consumption per idle SM. The equation considers the dynamic plus static

power consumption per active SM when running microbenchmark i. Based on that, it will then derive the power consumption for idle SMs by eliminating all other powers and repeating it for n microbenchmarks to find the power per idle SM. n = # microbenchmarks, i = specific microbenchmark (microbenchmark i), 80 = NVIDIA Volta GV100 has 80 SMs, $N_{activeSM}$ = # active SMs, N_{idleSM} = # idle SMs.

$$Pest. = \underbrace{\sum_{i=1}^{N} \frac{a_i \cdot \hat{E}_i}{T_{elapsedTime}} \cdot x_i}_{Dynamic} + \underbrace{\frac{P_{static\_yLanes}}{80} \cdot k + P_{perIdleSM} \cdot (80 - k)}_{Static} + \underbrace{P_{const}}_{Constant}$$

Figure 8. Total estimated power consumption with dynamic power equation

1.4.3 Dynamic Power Modeling.

Figure 8 shows an equation that overall estimates the power consumption of a GPU where it consists of dynamic, static, and constant power. The equation to estimate the dynamic power is the sum of each initial estimate of component i's energy per access $\hat{E}_i$, its activity factor a_i (the number of accesses to it during execution), and the run time $T_{elapsedTime}$ for N microarchitectural components. x_i is the unknown variable caused by inaccuracies that depend on some microarchitectural components. a_i = activity factor, k = # SMs, $\hat{E}_i$ = initial estimate of component i's energy consumption per access, x_i = unknown variable (caused by inaccuracy), i = specific component (component i), N = # microarchitectural components.

2 Implementation

For this project, we followed the "AccelWattch MICRO'21 Artifact Appendix Manual" which is a procedure of steps to produce the results similar to the original AccelWattch's paper. Our GPU for the implemenation is the NVIDIA GeForce GTX 1050 Ti which is based on the Pascal architecture. There are three main steps which consist of setting up validation benchmarks, hardware profiling to create a power model, and running simulations with the power model to estimate the power consumption of benchmarks.

2.1 Setting up validation benchmarks for AccelWattch

Firstly, we compiled the microbenchmarks, validation set benchmarks for simulator runs, and power profiling that were provided. The datasets of the benchmarks were also provided which needed to be extracted from a compressed file for later use. Table 1 shows the list of benchmarks used for AccelWattch. Figure 9 shows a screenshot of all the microbenchmark binaries that were compiled and ready to use for hardware profiling and simulations.

List of benchmarks	
backprop-rodinia-3.1	binomialOptions
b+tree-rodinia-3.1	dct8x8
fastWalshTransform	histogram
hotspot-rodinia-3.1	kmeans-rodinia-3.1
mergeSort	parboil-mri-q
parboil-sad	parboil-sgemm
pathfinder-rodinia-3.1	quasirandomGenerator
sobolQRNG	srad_v1-rodinia-3.1

Table 1. List of benchmarks for AccelWattch

Figure 9. All microbenchmark binaries

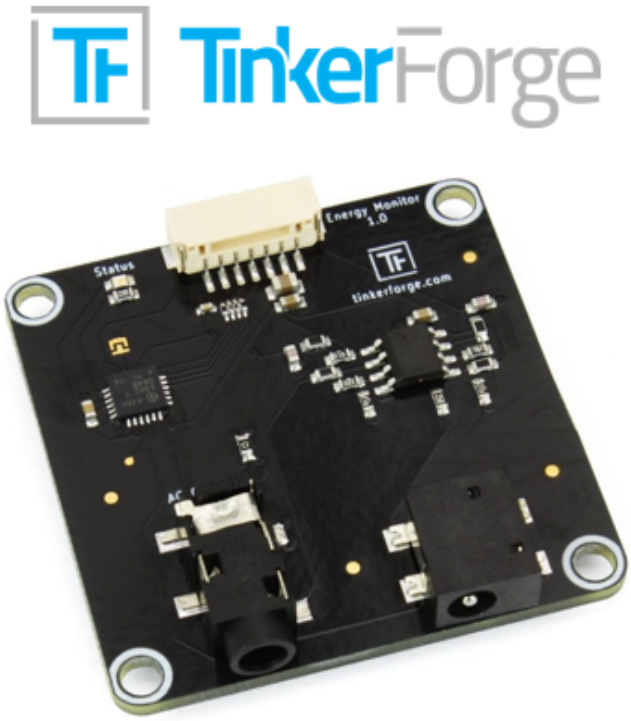


Figure 10. TinkerForge Energy Monitor Bricklet

2.2 Hardware profiling for AccelWattch validation

Unfortunately, the NVIDIA GTX 1050 Ti was not equipped with an internal power consumption monitor out of the box. Alternatively, we bought an Energy Monitor Bricklet[9] from TinkerForge, shown in Figure 10, which is a small current

sensor that we connected to the GPU to measure the total power consumption. We then integrated the Bricklet into AccelWattch to measure the power. Figure 12 shows a screenshot of the source code files of the Bricklet. Figure 13 shows the screenshot of AccelWattch's power measuring source code which we integrated the Bricklet code into it to measure the power consumption for the hardware profiling. After setting up the Bricklet, we then performed hardware profiling for validation. This step measured power five times and calculated the average for the runs for each validation set kernel on real hardware. It will first heat up the GPU chip to ≥65°C before taking power measurements. Temperature variability affects static power exponentially. Keeping a constant temperature during hardware measurements eliminates this noise. Figure 11 shows a screenshot of the hardware profiling runs.

Figure 11. Hardware profiling for validation runs

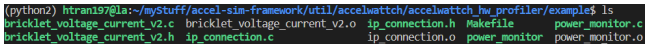


Figure 12. Bricklet source files

Figure 13. Integrating Bricklet into AccelWattch power measuring source code

2.3 Running AccelWattch and collecting power model results

Before running simulations and estimating the power, we need to recollect traces to use for the SASS simulation. We then launched AccelWattch jobs using Slurm[8] which the simulation will be run on GPGPU-Sim. For Pascal architecture, only PTX and SASS simulation modes are supported which we ran. After the jobs are finished, we just need to collect the simulation results for analysis later on. Figure 14 shows a screenshot of a log file containing the simulation results from b+tree-rodinia-3.1 in SASS simulation mode.

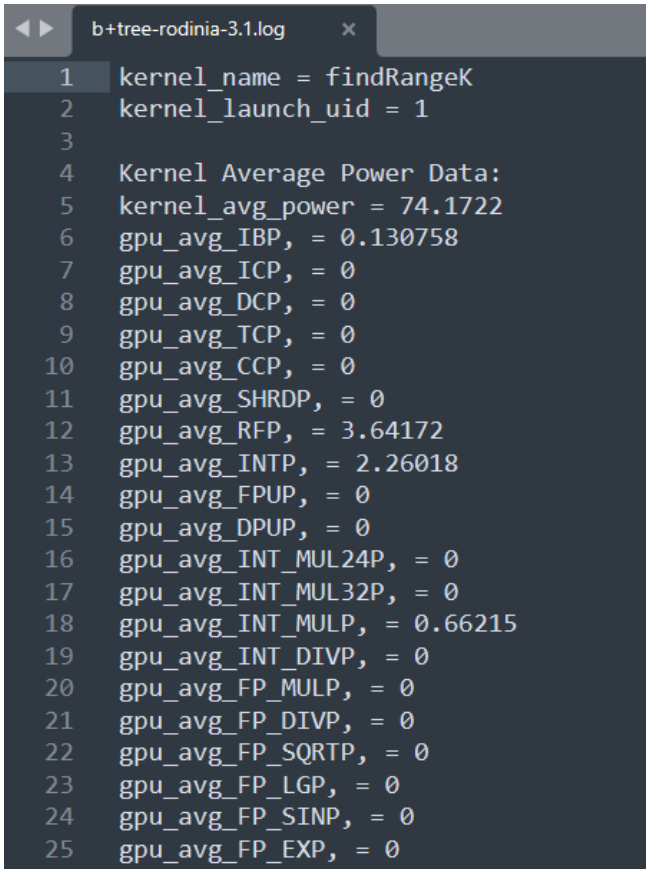
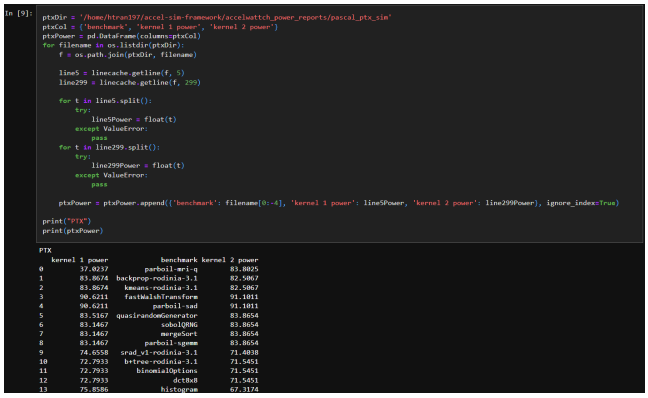


Figure 14. Simulation results

3 Result

To compare and analyze results, we created a Jupyter Notebook[3] script to collect the real and simulated results from all the runs. We create bar and scatter plots to visualize the correlation between the real and simulated power readings using seaborn[10]. We also calculated the average mean absolute percentage error (MAPE) to better understand how far or close the results from the power model are to the actual power. Figure 15, Figure 16, and Figure 17 are screenshots of the Jupyter Notebook of collecting data, plotting, and calculating MAPE.

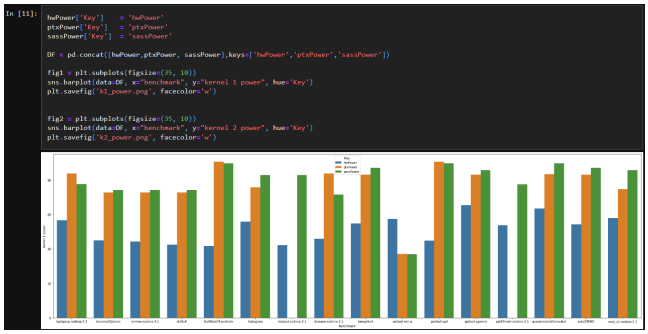


```
In [9]:
ptxDir = '/home/htran197/accel-sim-framework/accelwattch_power_reports/pascal_ptx_sim'
ptxCol = ['benchmark', 'kernel 1 power', 'kernel 2 power']
ptxPower = pd.DataFrame(columns=ptxCol)
for filename in os.listdir(ptxDir):
    f = os.path.join(ptxDir, filename)

    line5 = linecache.getline(f, 5)
    line299 = linecache.getline(f, 299)

    for t in line5.split():
        try:
            line5Power = float(t)
        except ValueError:
            pass
    for t in line299.split():
        try:
            line299Power = float(t)
        except ValueError:
            pass

    ptxPower = ptxPower.append({'benchmark': filename[0:-4], 'kernel 1 power': line5Power, 'kernel 2 power': line299Power}, ignore_index=True)

print("PTX")
print(ptxPower)
```

PTX

	kernel 1 power	benchmark	kernel 2 power
0	37.0237	parboil-mri-q	83.8025
1	83.8674	backprop-rodinia-3.1	82.5067
2	83.8674	tmvaux-rodinia-3.1	82.5067
3	90.6211	fastWalshTransform	91.1011
4	90.6211	parboil-sad	91.1011
5	83.5167	quasirandomGenerator	83.8654
6	83.1467	sobolQRNG	83.8654
7	83.1467	mergeSort	83.8654
8	83.1467	parboil-sgemm	83.8654
9	74.6558	srad_v1-rodinia-3.1	71.4038
10	72.7933	b+tree-rodinia-3.1	71.5451
11	72.7933	binomialOptions	71.5451
12	72.7933	dct8x8	71.5451
13	75.8586	histogram	67.3174

Figure 15. Collecting data



```
In [11]:
hwPower['Key']   = 'hwPower'
ptxPower['Key']  = 'ptxPower'
sassPower['Key'] = 'sassPower'

DF = pd.concat([hwPower,ptxPower, sassPower],keys=['hwPower','ptxPower','sassPower'])

fig1 = plt.subplots(figsize=(35, 10))
sns.barplot(data=DF, x="benchmark", y="kernel 1 power", hue='Key')
plt.savefig('k1_power.png', facecolor='w')

fig2 = plt.subplots(figsize=(35, 10))
sns.barplot(data=DF, x="benchmark", y="kernel 2 power", hue='Key')
plt.savefig('k2_power.png', facecolor='w')
```

Figure 16. Plotting

```
In [51]:
sassMAPEk1 = []
ptxMAPEk1 = []
for i, row in scatDFk1.iterrows():
    if math.isnan(scatDFk1['hwPower'][i]) != True and math.isnan(scatDFk1['sassPower'][i]) != True:
        sassMAPEk1.append(mape(scatDFk1['hwPower'][i],scatDFk1['sassPower'][i]))

for i, row in scatDFk1.iterrows():
    if math.isnan(scatDFk1['hwPower'][i]) != True and math.isnan(scatDFk1['ptxPower'][i]) != True:
        ptxMAPEk1.append(mape(scatDFk1['hwPower'][i],scatDFk1['ptxPower'][i]))

print("sassMAPE k1:", np.average(sassMAPEk1))
print("ptxMAPE k1:", np.average(ptxMAPEk1))

sassMAPEk2 = []
ptxMAPEk2 = []
for i, row in scatDFk2.iterrows():
    if math.isnan(scatDFk2['hwPower'][i]) != True and scatDFk2['hwPower'][i] > 0.0 and math.isnan(scatDFk2['sassPower'][i]) != True:
        sassMAPEk2.append(mape(scatDFk2['hwPower'][i],scatDFk2['sassPower'][i]))

for i, row in scatDFk2.iterrows():
    if math.isnan(scatDFk2['hwPower'][i]) != True and scatDFk2['hwPower'][i] > 0.0 and math.isnan(scatDFk2['ptxPower'][i]) != True:
        ptxMAPEk2.append(mape(scatDFk2['hwPower'][i],scatDFk2['ptxPower'][i]))

print("sassMAPE k2:", np.average(sassMAPEk2))
print("ptxMAPE k2:", np.average(ptxMAPEk2))
```

```
sassMAPE k1: 62.28068615938332
ptxMAPE k1: 62.94431677656719
sassMAPE k2: 60.14936245781224
ptxMAPE k2: 68.36075924570116
```

Figure 17. Calculating MAPE

3.1 Kernel 1 results

Figure 18 shows a bar plot of all results from running kernel 1 of each benchmark for real, PTX, and SASS power.

3.2 Kernel 2 results

Figure 19 shows a bar plot of all results from running kernel 2 of each benchmark for real, PTX, and SASS power.

3.3 Real Power vs Simulated Power Kernel 1

Figure 20 shows the correlation plot of real vs PTX and real vs SASS in kernel 1 of benchmarks. We got the average MAPE for PTX of 62.94% and the average MAPE for SASS of 62.28%.

3.4 Real Power vs Simulated Power Kernel 2

Figure 21 shows the correlation plot of real vs PTX and real vs SASS in kernel 2 of benchmarks. We got the average MAPE for PTX of 68.34% and the average MAPE for SASS of 60.15%.

4 Analysis

After getting all results together, we concluded that the model we created and tested using AccelWattch was inaccurate compared to what was advertised in the AccelWattch published paper. The average MAPE for both kernels on PTX and SASS for our experiment was 63.42%. In the paper, the MAPE of the Pascal TITAN X they got was 11±3.8%. We can clearly see that our model is far from accurate compared to their results. There are several possible causes for the inaccuracy.

- Since AccelWattch is driven by a performance model, our performance model was never validated at any stage of the procedure which by itself can be the only reason for this result. The performance model is the core part of AccelWattch which feeds execution stats to AccelWattch for estimating power consumption.
- There is a difference in binary code between running on real hardware vs simulation. Since PTX is a virtual ISA that will get translated to native SASS, there will be optimizations that will change the code. In addition, GPGPU-Sim does not simulate actual SASS instructions. It actually simulates PTXPlus[2] instructions which claimed to have a one-to-one mapping with SASS instructions. In general, the code simulated in GPGPU-Sim is not identical to the one running in real GPU.
- Another major cause is the inaccuracies within the GPGPU-Sim configuration file. Since internal microarchitecture information about the 1050 Ti is limited, we were not able to configure GPGPU-Sim to fit exactly our GPU configuration. Some of the configurations were not found so we used the provided configuration of the TITAN X.

5 Challenges

Overall, we were able to achieve the main goal of building the power model using AccelWattch and validating it against real power consumption readings. However, the process was not smooth sailing and we did encounter several challenges.

- Since AccelWattch was implemented by a small group of researchers, there were many small inconsistencies within the scripts. Source and destination directories are either hard-coded, inconsistent, or not updated which results in some directories not being found. Some Makefiles are also not fully working because of included flags that caused errors when compiling files. Since AccelWattch was specially designed for Volta architecture, there are TensorCore-related benchmarks

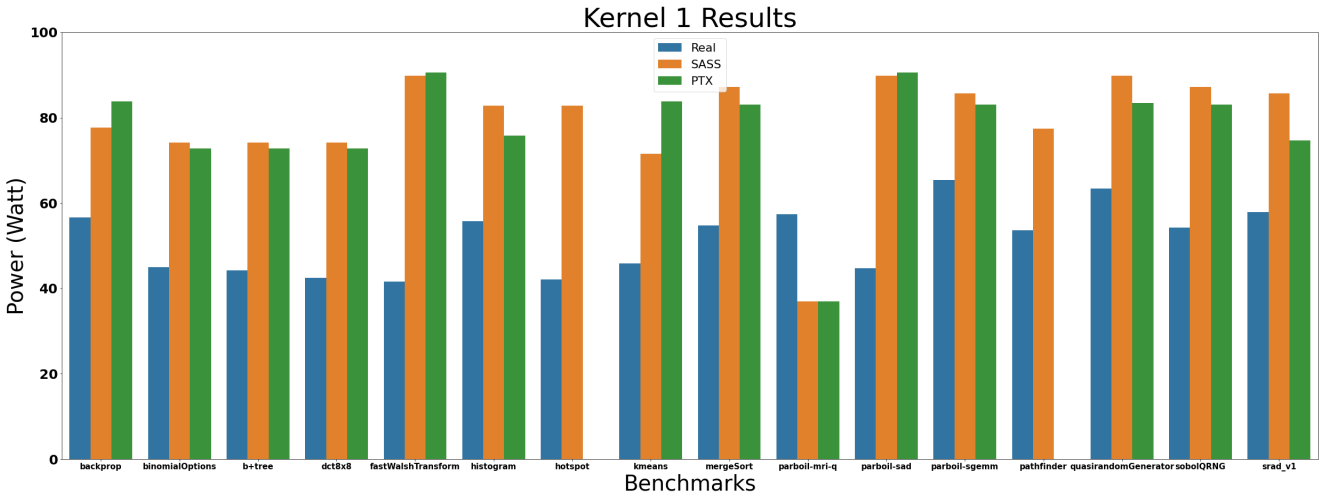


Figure 18. Real, PTX, SASS power results over kernel 1 of each benchmark

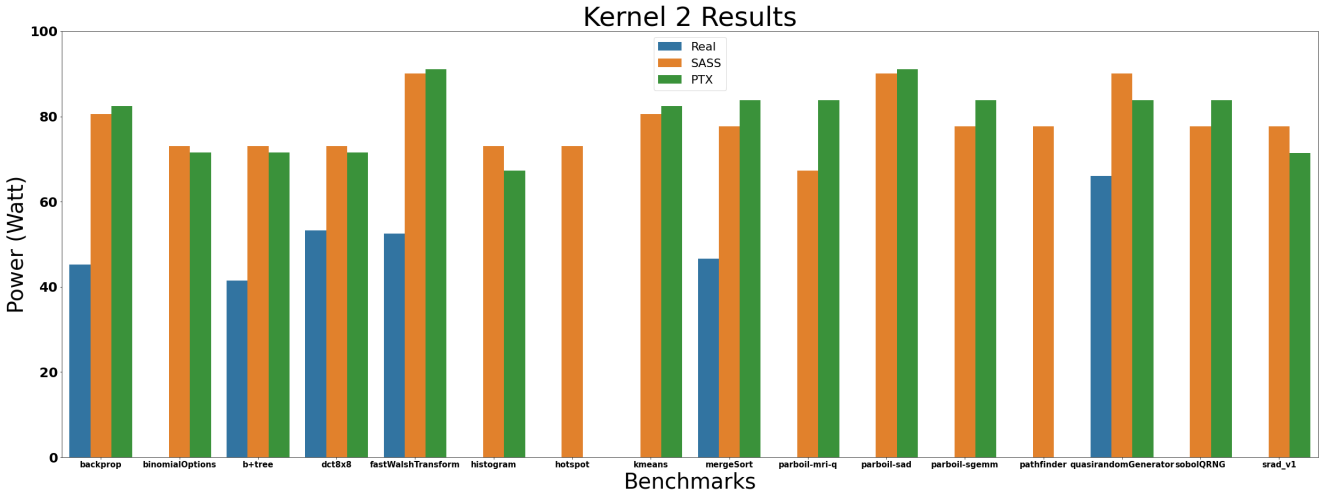


Figure 19. Real, PTX, SASS power results over kernel 2 of each benchmark

that were forced to compile or run which is not supported for Pascal since it doesn't have Tensor cores.

- We also faced problems with running some simulations of benchmarks that failed without much feedback from the runs. There are several causes that we thought prevented these benchmarks to run. According to AccelWattch's researchers, setup environment scripts were not fully successfully ran which causes CUDA's libraries not to be found by the benchmarks which we checked but didn't work. It might be our system is not fully set up correctly or maybe some programs are not compatible or outdated. There were instances where NVIDIA's software was not fully functioning which causes some CUDA-related errors.

References

[1] Ali Bakhoda, George L. Yuan, Wilson W. L. Fung, Henry Wong, and Tor M. Aamodt. 2009. Analyzing CUDA workloads using a detailed GPU simulator. In *2009 IEEE International Symposium on Performance Analysis of Systems and Software*. 163–174. https://doi.org/10.1109/ISPASS.2009.4919648

[2] GPGPU-Sim. 2009. PTXPlus. http://gpgpu-sim.org/manual/index.php/Main_Page#PTX_and_SASS

[3] Jupyter. 2015. Jupyter Energy Monitor Bricklet. https://jupyter.org/

[4] Vijay Kandiah, Scott Peverelle, Mahmoud Khairy, Junrui Pan, Amogh Manjunath, Timothy G. Rogers, Tor M. Aamodt, and Nikos Hardavellas. 2021. AccelWattch: A Power Modeling Framework for Modern GPUs. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture* (Virtual Event, Greece) *(MICRO '21)*. Association for Computing Machinery, New York, NY, USA, 738–753. https://doi.org/10.1145/3466752.3480063

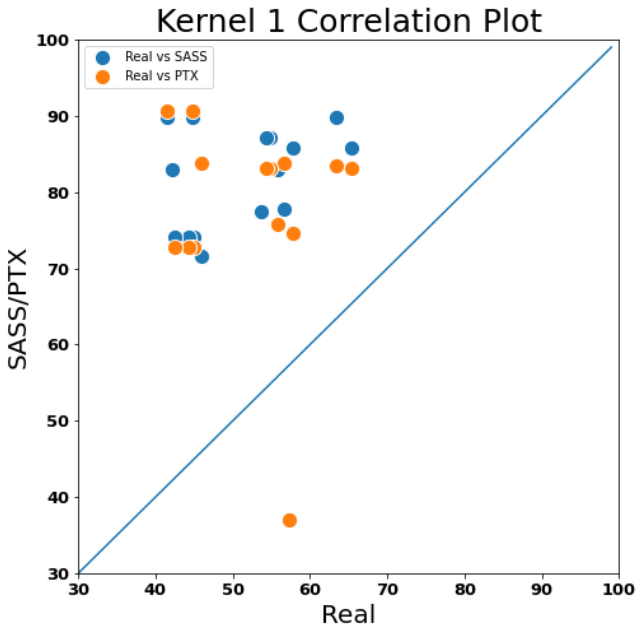


Figure 20. Correlation plot of kernel 1

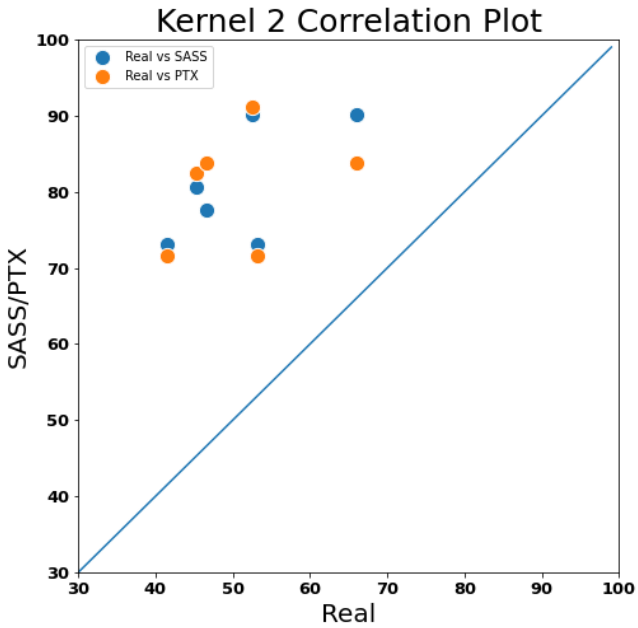


Figure 21. Correlation plot of kernel 1

[5] Mahmoud Khairy, Zhesheng Shen, Tor M. Aamodt, and Timothy G. Rogers. 2020. Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 473–486. https://doi.org/10.1109/ISCA45697.2020.00047

[6] NVIDIA. 2007. Parallel Thread Execution ISA Version 8.1. https://docs.nvidia.com/cuda/parallel-thread-execution/index.html

[7] NVIDIA. 2007. PTX and SASS Assembly Debugging. https://docs.nvidia.com/gameworks/content/developertools/desktop/ptx_sass_assembly_debugging.htm

[8] SchedMD. 2003. Slurm, Workload Manager. https://slurm.schedmd.com/

[9] TinkerForge. 2011. TinkerForge Energy Monitor Bricklet. https://www.tinkerforge.com/en/shop/energy-monitor-bricklet.html

[10] Michael Waskom. 2012. seaborn: statistical data visualization. https://seaborn.pydata.org/