

Analysis of GPU Power Consumption Using Internal Sensors

Mariza Ferro¹, André Yokoyama¹, Vinicius Klôh¹, Gabrieli Silva¹, Rodrigo Gandra²,
Ricardo Bragança², Andre Bulcão², Bruno Schulze¹

¹Laboratório Nacional de Computação Científica (LNCC)
Getúlio Vargas, 333, Quitandinha – Petrópolis – Rio de Janeiro

{mariza, andremy, viniciusk, gabrieli, schulze}@lncc.br

²Centro de Pesquisa da Petrobrás - CENPES
Petróleo Brasileiro S.A. - PETROBRAS

{rodrigo.gandra, rbraganca, bulcao}@petrobras.com.br

Abstract. GPUs has been widely used in scientific computing, as by offering exceptional performance as by power-efficient hardware. Its position established in high-performance and scientific computing communities has increased the urgency of understanding the power cost of GPU usage in accurate measurements. For this, the use of internal sensors are extremely important. In this work, we employ the GPU sensors to obtain high-resolution power profiles of real and benchmark applications. We wrote our own tools to query the sensors of two NVIDIA GPUs from different generations and compare the accuracy of them. Also, we compare the power profile of GPU with CPU using IPMItool.

1. Introduction

Scientific computing generally requires huge processing power resources' to perform large scale experiments and simulations in reasonable time. These demands have been addressed by High Performance Computing (HPC), allowing many scientific domains to leverage progress. The design of high performance supercomputers is on the boundary of constant and significant changes. With the arrival of petascale computing in 2008, we see another potential paradigm shift in the construction of parallel computing and the use of hybrid designs employing heterogeneous computational accelerators. In these designs, specialized hardware devices such as graphics processing units (GPUs), dense multi-core processor, MIC and FPGAs have been widely used for HPC applications due to their extremely high computational powers [Mittal and Vetter 2014].

However, despite the impressive theoretical peak performance of petascale supercomputers, several areas require more computational power, like the exascale supercomputers expected for the coming decade. For example, in the energy industry simulations, for different energy sources like wind energy, efficient combustion systems for biomass and exploration geophysics in the oil and gas industry. For the latter, seismic applications, as used in this work, are targets of this type of processing [Menezes et al. 2012]. But, it is expected that to achieve exascale will require a nearly 30-fold increase in performance with only a 1.2-fold increase in power consumption [Adhinarayanan et al. 2016].

GPUs have become prevalent in HPC, as by offer exceptional performance, especially for scientific applications highly parallelizable, like seismic applications, as by power-efficient hardware. The position GPUs have established in high-performance and

scientific computing communities has increased the urgency of understanding the power cost of GPU usage [Bridges et al. 2016] [Adhinarayanan et al. 2016].

To attempt this urgency, the use of internal power sensors for more accuracy power and energy measuring is a current area of research. But, this is not trivial, since the documentation on these sensors is scarce. So, understand exactly how to obtain power, and other measures of performance, for the GPU board is unclear. In addition, as GPUs continue to evolve, understanding their power and performance profiles on real applications is increasingly difficult [Bridges et al. 2016].

In this work we performed experiments on two high-end NVIDIA GPUs from different generations and collect data from its internal sensors. We wrote our own tools to query the sensors that enable a high-resolution power profiles of GPU kernel functions and low overhead. We compare the accuracy of profiles via the NVIDIA Management Library (NVML) [NVIDIA 2012] or IPMItool. Also, we compare the power profile of GPU with CPU using IPMItool for a deeper understanding. The experiments were conducted through either a benchmark from Rodinia [Che et al. 2009] and a real application of seismic area, used by the Brazilian Oil Company (Petrobrás). Understand the behavior of this application and its relation with the energy consumption are among the objectives of the HPC4E project ¹, where this work is inserted.

2. Background and Related Work

In this section, we present some background about approaches for monitoring systems and power measurement in GPUs. The discussion of these concepts could be very broad, however, we will only discuss some concepts that are closely related to this research, as well, related works only focused on GPUs.

Power consumption of computer components can be obtained either for directly or indirectly methods. Directly measurements could be made via internal or external hardware sensors which periodically collect samples to estimate the power used during a time interval [Bridges et al. 2016]. An estimate of total energy is calculated as the integral of the power over the execution time [Burtscher et al. 2014]. External power meters are connected between the power supply unit and a component under investigation [Bridges et al. 2016]. Internal power meters are obtained directly from built-in sensors, allowing users convenient access to power data via profiling software by sampling.

According to [Bridges et al. 2016], directly measuring power via internal or external hardware sensors is considered the most accurate source of power consumption information. However, external power meters are generally not suitable for comprehensive power profiling, giving not accurate readings, especially in HPC settings. Internal sensors, when available, can be used for more accurate measurement. Besides accuracy, another advantage is the no need of any expensive extra hardware and allows component level profiling. However, not all hardware has these internal sensors and when there is, there is no standard feature nor consistency in accuracy across different hardwares. So, in some cases there is a need, especially in the HPC industry, for indirect methods [Bridges et al. 2016].

Indirect measurements cover modeling and simulation techniques for power and performance estimation. The Modeling approach estimates the power consumption using

¹<http://www.hpc4e.eu/>

a model that correlates power with hardware performance counters. A great number of works in the area of energy efficiency are focused on using models to estimate power consumption [Burtscher et al. 2014]. Some limitations to counter-based models are that the number and type of counters available being not uniform across hardwares. So, there is a great variety of ways for accessing and visualizing hardware counter data but, the models generally are hardware dependent. Example of hardware monitoring software is the suite of profiling tools offered by NVIDIA that enable capabilities for accessing, visualizing, optimizing, monitoring and profiling applications and GPU hardware [NVIDIA 2012].

2.1. Related Works

Some researches in this area focuses on indirectly methods using models to estimate the power consumption. The work of [Adhinarayanan et al. 2016] proposes an application-dependent modeling technique, where the model is constructed online at runtime, based on the readings from the built-in GPU power meter. Wang analyzes the power consumption of the main functional units in GPU using a power simulator and shows that traditional power optimization methods for CPUs, such as dynamic frequency scaling and concurrency-throttling, could be effectively applied on GPU architectures for better power efficiency [Wang 2010]. The work of [Karami et al. 2013] uses the values of GPU performance counters and then applies a regression model and principle component analysis to build a power consumption and performance prediction model of OpenCL kernels.

There are many works that investigate the Dynamic Voltage and Frequency Scaling (DVFS), i.e., changing the processor voltage/frequency during task processing for saving energy. A number of works with this approach are analysed in the survey of [Mei et al. 2016], which also proposes a GPU DVFS runtime power modeling. The work of [Ge et al. 2013] studies the impacts of DVFS on application performance and energy efficiency for GPU computing and comparing them with those of DVFS for CPU computing. They conclude that the effects of DVFS on GPUs differ from on CPUs.

Another approach is modeling power and performance by using indirect methods and then evaluate the error of the model by means of the direct measurements from the internal sensors, which are more accurate. This kind of approach should be used when the access to internal sensors is not allowed [Kasichayanula et al. 2012].

By using external sensor, the work of [Lee and Kim 2015] made an empirical characterization of power efficiency. They measured the energy consumption using a digital multimeter, obtaining the power consumption of the entire system and providing the performance and power characterization of CPUs and GPUs.

Researches using internal sensor are similar to our study. In [Burtscher et al. 2014] they propose a methodology to compute the instant power and energy consumption of K20 GPUs. This work is similar to ours, in the sense that they developed its own application to query built-in sensors using NVIDIA NVML. But, the analyses is focused on the power (w) and energy for the tested kernels and for one generation of GPUs, while in ours the analyses included temperature and performance parameters. Also, we compare the behavior of two generations of GPUs with two different GPU's sensor and compare them with the CPU power profile. The work of [Coplin and Burtscher 2016] also uses the internal sensors (K20 GPU) for measurements, but through K20Power tool. They study the effects on energy consumption, power

draw and runtime when changing the core and memory clock frequencies, enabling or disabling ECC, using alternate implementations, and varying the program inputs.

In Section 3 we present some management and monitoring tools that use the method of direct measurements via internal hardware and how we use them to develop our way for power measurements.

3. Power Measuring Approach

For each GPU model the sensor is different and enables a kind of measurement, accuracy and requires an approach to collect the data from sensors. In this work, as mentioned, we employ the GPU power sensors to obtain the energy consumption of the tested applications. We wrote our own tools to query the sensors, via the IPMItool for Tesla M2050 and NVML interface for Tesla K40, which implement a direct method by sampling monitoring approach. This kind of method periodically collect samples of the parameters. The power is obtained directly from sensors, and the total energy is calculated as the integral of the power over the execution time.

The Intelligent Platform Management Interface (IPMI) is a standard interface for hardware management used by system administrators to control the devices and monitor the sensors. For these, it is necessary the IPMI Controller called Baseboard Management Controller (BMC) and a manager software (for example, IPMItool²). It provides an interface to manage IPMI functions in a local (in-band) or remote (out-of-band) system.

We are using the IPMI, in-band, to monitor the temperature and power consumption sensors by sampling methodology (both for CPU and GPU Tesla M2050). The IPMI Controller is used to identify how the application influences power consumption and increases the temperature. The driver included in our architecture is the OpenIPMI Linux Kernel Driver (`/dev/ipmi0`) and the manager software adopted was IPMItool³. To collect these parameters, we are using with a sudo user following command: `$ ipmitool dcmi power reading`

To use the IPMItool with the above command and to select the desired parameters (power and temperature), we developed a Shell-based tool to collect CPU data. This tool also collects the CPU and Memory utilization rates through the Linux's Top utility⁴. Since the tool runs in parallel with the monitors application, competing for the same CPU resources, there is an overhead in this usage. This overhead was measured to be about 2.5% of CPU utilization rate. To collect Tesla M2050 GPU data we developed only a Shell monitoring script that queries the sensor readings, using IPMItool commands.

The NVIDIA profiling tools and APIs can be used to monitor and manage states of the NVIDIA GPU devices. The NVIDIA Management Library (NVML) [NVIDIA 2012] is an API, based on C language, for monitoring and managing states of NVIDIA GPU devices, such as GPU utilization rate, running process, clock and performance state, temperature and fan speed, power consumption and power management. The `nvmlDeviceGetPowerUsage` function retrieves the power usage reading for the device, in milliwatts, with an error of ± 5 watts. This is the power draw for the entire board,

²<https://www.kernel.org/doc/Documentation/IPMI.txt>

³<https://docs.oracle.com/cd/E19464-01/820-6850-11/IPMItool.html>

⁴<https://linux.die.net/man/1/top>

including GPU, memory, among others [Kasichayanula et al. 2012].

The NVIDIA System Management Interface (NVSMI) ⁵ is a command line program that allows the system administrators to query, with appropriate privileges, states of the GPU devices. Its functions are provided by the NVML library. By using the NVSMI it is possible to collect some of the same parameters as those obtained with our tool and with the sampling approach. However, when using NVSMI a much wider range of parameters is collected, which could result in overhead. In addition, since NVSMI is a high level utility, the rate of sampling power usage is very low (1 Hz). Such sampling rate might not be enough to notice the change in power, unless the kernel is running for a very long time.

For these reasons, we developed a CUDA C-based tool, using the NVML functions, to collect only the following parameters: Device name, Memory (Free, Used, Total), Utilization Rate (Memory, GPU), Power Consumption (W) and Temperature (C). In addition, this tool uses two threads: one thread responsible for launching, on GPU devices, the application to be monitored and another thread to collect by sampling, those parameters. So, an important feature of this tool is related to overhead, since the second thread only runs on CPUs, which does not impact the execution of the application on GPUs, the overhead is almost zero.

This NVML based monitoring application was developed with the intent of maximizing the sampling rate in order to increase the accuracy of the measurements, particularly for applications with small kernels. For this purpose the monitoring application only queries for the necessary parameters and the sampling rate is not fixed at a preset value. During the tests we observed a sampling rate of about 500 Hz, which provided some good measurements for the application used during the tests. This measurements, as well as the details of the methodology used to collect them, will be presented in the next section.

4. Experimental Evaluation

In this section we present the experiments with focus on the measurements obtained from GPU sensors and discuss some results about that.

4.1. Experimental Setup

Our hardware platforms include two high-end NVIDIA GPUs from different generations, described in Table 1 and one CPU x5650 Intel(R) Xeon with 23 GB of memory, 32 GB/s of memory bandwidth and 12 cores at 2.66 GHz. The Tesla K40 is used in Santos Dumont Supercomputer ⁶ (SDumont) and GPU Tesla M2050 and CPU Intel Xeon are used in ComCiDis Cluster.

The main application used in the experiments is an acoustic test implemented by Petrobrás and part of the HPC4E Project. This is a *3D Wave Propagation* which simulates the propagation of a single wavelet over time by solving the acoustic wave propagation equation. This equation is solved by using finite differences that have a high degree of parallelization, given the interdependence between the data [Menezes et al. 2012]. The program was written in standard C and CUDA and spreads computation over a single

⁵<https://developer.nvidia.com/nvidia-system-management-interface>.

⁶<http://sdumont.lncc.br/machine.php?pg=machine>

	Tesla M2050	Tesla K40
Processor	GF100	GK110B
Memory (ECC off) (GB)	3	12
Memory bandwidth (ECC off) (GB/s)	148	288
Clock (GHz)	1,546	3,0
Core	448	2880
GPU clock (MHz)	1150	745 (875MHz boosted)
TDP (Watts)	225	235
GFLOPS single precision	1030	4291-5040
GFLOPS double precision	515	1430-1680

Table 1. GPU models used in the experiments.

CPU and a single GPU. For the experiments with *3D Wave Propagation* were used different input sizes, described in Table 2. The Size ID corresponds to different sizes of the problem, which represents the discretization of the domain on a 3D grid. The X, Y and Z correspond to the size of the problem on the three axis and the MB line is the size of memory used for each group of the input size problem.

	Size ID											
	A	B	C	D	E	F	G	H	I	J	K	L
X	128	256	256	256	256	256	512	512	768	768	1024	512
Y	128	256	256	512	256	256	512	512	768	768	1024	512
Z	128	256	512	512	1024	2048	512	1024	768	1024	704	2832
MB	32	256	512	1024	1024	2048	2048	4096	6912	9216	11266	11328

Table 2. Input sizes used in experiments with 3D Wave Propagation.

In some experiments, particularly to compare CPU and GPU measurements with IPMItool, we used the LUD benchmark from Rodinia [Che et al. 2009]. We choose this kernel to perform a set of experiments with more controlled results and also that could be executed in both GPU (CUDA version) and CPU (OpenMP). LUD is an algorithm to calculate the solutions of a set of linear equations that decomposes a matrix as the product of a lower triangular matrix and an upper triangular matrix to achieve a triangular form that can be used to solve a system of linear equations easily. In this work, we presented results for matrix size of 16384.

4.2. Methodology

Since the IPMItool requires *sudo* permission it was used only on our own cluster (Com-CiDis). It was the only way to get power and temperature reading for the Tesla M2050, since it doesn't have the built-in sensors. The measurement data are collected using a shell script that repeatedly queries the sensor readings, using the IPMItools commands, until it is stopped by a "Ctrl+C" termination input. The output of the queries are written on an output file. To collect the data, the monitoring script is launched 30 seconds before the application to be monitored. This 'padding' allows us to establish the baseline of power drain when only the monitoring script is running, and by querying the sensors manually. We can get the idle power drain⁷ when no application is running. This information allows us to estimate the real energy consumed by the application by removing the overhead (in

⁷We named power drain the amount of power been [drawn/consumed] by the system.

power drain) caused by the monitoring script. The experiments performed on ComCidis with CPU followed the same methodology to collect the data.

For the experiments on SDumont, only NVML based monitoring application was used, since we don't have *sudo* permission to use IPMItool. K40 has built-in sensors, that no need *sudo* for temperature and power readings. These sensors have a higher precision than those read by the IPMItool and we can get a much more accurate amount of energy. Nevertheless, as we can't use the IPMItool, we are unable to get the total energy consumed by the entire computing node.

As mentioned in Section 3, the NVML based monitoring application uses 2 threads, the first one queries the sensors of all GPUs installed in the computing node and write the readings in an output file, the second thread is responsible for launching the application that's been evaluated, it is also responsible for signaling the monitoring thread to stop collecting data. The monitoring thread starts collecting data as soon as it is launched and only stops when signaled by the second thread. This thread starts the application 30 seconds after the monitoring thread starts, and signal the stop 30 seconds after the end application. This 30 seconds padding allows us to establish a baseline consumption while all the GPUs in the node are in the idle state.

4.3. Results on GPUs

In this section we present and discuss results of the experiments highlighting some aspects of power profiles obtained from internal sensors. The results were obtained using the NVML based monitoring application for SDumont and IPMItool, while running the *3D Wave Propagation application*.

In the graphs of Figures 1 - 4 are presented the results for problem sizes A and G, performed both in SDumont as in ComCiDis. In Figure 5 are presented the results for problem size L, only in SDumont. It was too large to execute in ComCiDis, since this application requires the entire problem to be loaded to the GPU memory and problem size L uses about 11.2 GB, which is too large for the 2.65 GB of the ComCiDis' Tesla M2050. The largest size to fit in a single Tesla M2050 is the problem size G.

The parameters collected and presented in Figures 1, 3 and 5, are the memory and GPUs usage, power and temperature, for each device (device 0 and 1). In Figures 2 and 4, due the IPMItool limitation, there are only the temperatures of each CPU (CPU1 Temp, CPU2 Temp) and GPU (GPU1 Temp, GPU2 Temp) and Power.

The power levels collected by the monitoring application are the individual power levels for each GPU with a sampling rate around 500 Hz, while the IPMItool collects the total power used by the whole computing node with a rate between 1 and 3 Hz. These differences in how to collect the results by each tool can be compared by results presented in the pairs of Figures 1-2 and 3-4. It is possible to note how the difference in the sampling rate drastically affects the resolution and precision of the results.

Figures display the power profiles obtained when sampling the GPUs before, during and after the execution of the application. In Figure 1, T_0 indicates when the monitoring application starts to collect data. At this time, Slurm is still finishing the startup of the computing node, some activity can still be seen in device 1 (GPU Usage 1) and the power levels for both GPUs are still high, as they are in a ready state. In time T_1 ,

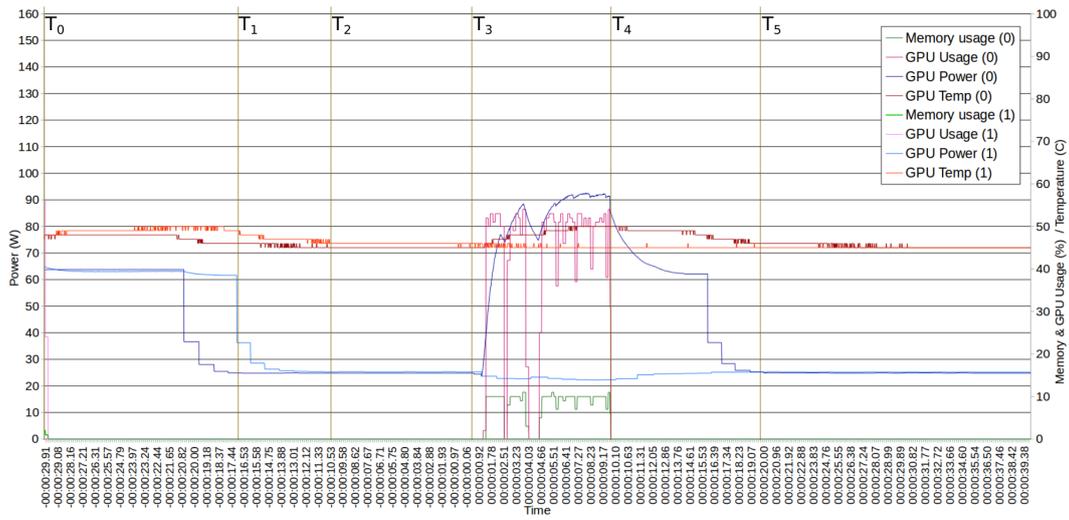


Figure 1. Results for problem size A on SDumont collected by the NVML based monitoring application.

both GPUs are now in idle state and the power levels drop. In T_2 the power levels and temperature stabilize with the GPU in idle state. In T_3 and T_4 the application starts and finishes, respectively. From T_4 , the power levels decrease slowly and stabilise at a ready state level. This happens because the GPUs are kept in the ready state for a few seconds after an application ends, so in case a new application is launched the overhead of starting the GPUs from the idle state is avoided. Finally, in T_5 the power levels stabilise at idle state.

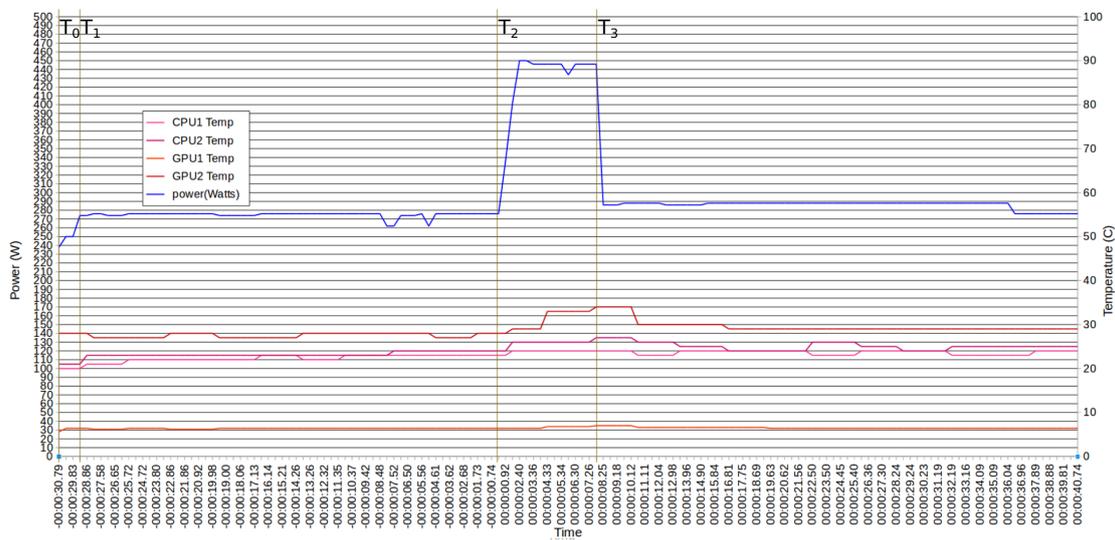


Figure 2. Results for the problem size A on ComCiDis Cluster collected by the IPMITool application.

In Figure 2, the monitoring script starts on T_0 . Due to the low refresh rate of the sensors, the first reading shows the power levels from before the script start, when the computing node was idle. In T_1 the power readings stabilise reflecting the power drain of the monitoring script. In T_2 and T_3 the application starts and finishes, respectively.

Figures 1 and 2 display the results with the problem size A for SDumont and

ComCiDis. The average time to complete the process in SDumont was 9.5 seconds while for the ComCiDis it was 7.9 seconds. The average power drained by both GPUs was about 290 W and the energy-to-solution was 2,291 J. The power drain by both GPUs in SDumont was less than 115 W, with an energy-to-solution of 1,092 J.

Problem size A was a very small problem and doesn't provide us enough data for a more detailed analysis of the power profile of the applications. But, as we move to the G size, it gives us a clearer view of the behavior of the application as well as the impact of the sampling rate of the two methods used in the data collection.

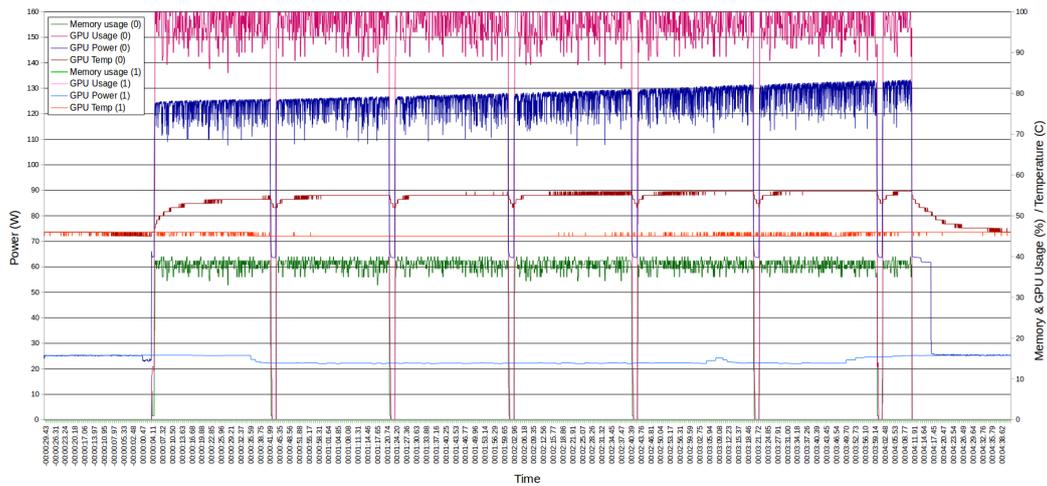


Figure 3. Results for problem size G on SDumont collected by the NVML based monitoring application.

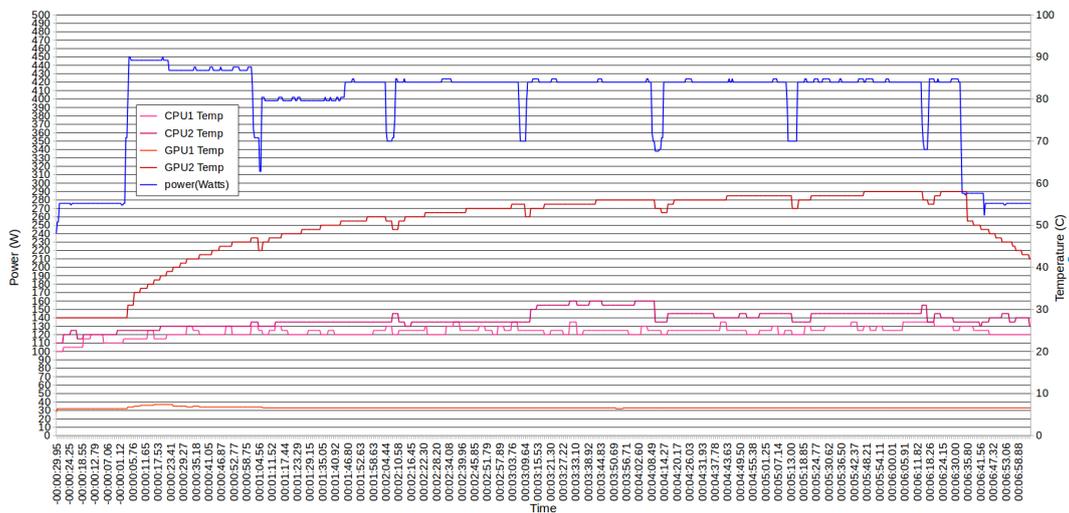


Figure 4. Results for problem size G on the ComCiDis collected by the IPMITool application.

Figures 3 and 4 display the results for the problem size G on SDumont and ComCiDis. The average time for this size in SDumont was 4 minutes and 12 seconds, consuming 36802 J (30,520 J of the computing GPU and 6,281 J of the idle GPU). For the ComCiDis it was 6 minutes and 32 seconds, more than 1.5 times longer and consuming

101703 J (2.76 times more than SDumont’s K40). The peak of the power drain observed in the ComCiDis by GPUs was of 290 W, but it stays at 250 W most of the time. For SDumont the peak was about 160 W for the two GPUs together (134 W for de GPU when it was processing and 26 W for the GPU idle).

In Figure 3 we can observe a high level of fluctuation on the power readings (GPU Power 0). In this graph it is possible to see, particularly looking at the GPU and memory usage for the running device (Memory usage 0 and GPU Usage 0), that the pattern of increase and decrease in usage matches that of the power readings, with only the power reading showing a steady increase over time. This could be related to the GPU’s temperature and this behavior has been observed also in [Burtscher et al. 2014].

Looking at the behavior of the application for the size G (Figures 3 and 4), there is a significant drop in the power readings in regular intervals. This happens because the application has pauses in the processing, in regular intervals, to take ‘snapshots’ (copying the data in the GPU memory to the CPU memory). During this time the GPU goes into a idle state, but ready to run a new kernel. In this state, the power drain is higher than a completely idle GPU. What we can see in the graphs is that during this pauses in the processing, the temperature drops as expected. But, when the application resumes the processing, the temperature rises faster than in the previous sequence o kernels. Also, the temperature gets higher, as can be seen in Figure 3, than the maximum temperature observed in the last sequence, being about 2 degrees higher than in the first sequence. In addition to that, it also gets to that temperature faster. This increase in the temperature could be contributing to the increase in power drain, but it could also be caused by it. We need to perform more tests in order to properly verify this behavior, including the execution of tests after pre-heating the GPU, or super-cooling it.

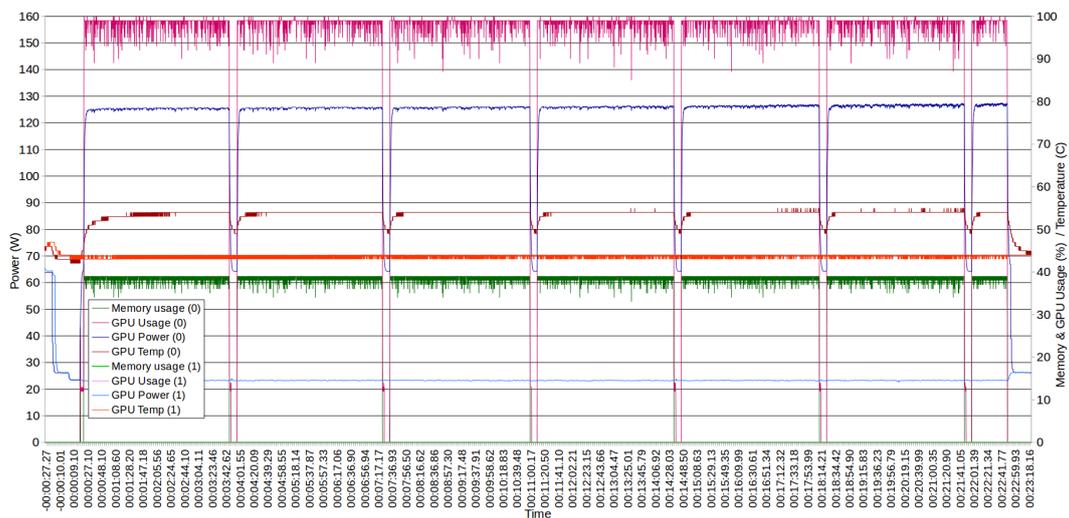


Figure 5. Results for the problem size L, executed on SDumont and collected by the NVML based monitoring application.

Analyzing the results for the largest execution size (size L) on SDumont (Figure 5), it is possible to see some behavior seen also in the graphs for the G size. For example, the steady increase in the power readings over time, as well for the temperature, and the ‘stop-and-go’ behavior, similar for the G size. Although there is also a high level

of fluctuation on the usage of the GPU cores and memory, the range of the fluctuation is shorter than that seen in the problem size G. In this case, the readings have a higher concentration, at 99% usage for the GPU and 39% for the memory. We also can see the fluctuation on the power readings, but the range is very narrow, almost negligible. The L size took 22 minutes and 50 seconds to complete, consuming 200,740 J (166,496 J of the computing GPU and 34,244 J of the idle GPU).

Table 3 summarizes the results for all experiments. Time to solution for small problem size (A) has less execution time in ComCiDis, which has lower computational capacity. However, the energy to solution was considerably larger. This behavior change from problem size G, where time and energy to solution are lower for SDumont, since the power consumption by this GPU generation is smaller and also the execution time is shorter. In general what could be observed, for this set of experiments, is that as the size of the problem increases, the execution time increases. However, the energy consumption is not proportional to the increase in execution time.

Size	Cluster	Time to solution	Energy to solution
A	SDumont	9,5 sec	1,092 J
A	ComCiDis	7,9 sec	2,291 J
G	SDumont	4 min 12 sec	36,802 J
G	ComCiDis	6 min 32 sec	101,703 J
L	SDumont	22 min 50 sec	200,740 J

Table 3. Time and Energy to Solution on GPUs.

4.4. Results - CPU x GPU

Here, we will present the results obtained in ComCiDis cluster using both the IPMItool based monitoring application for CPU and the script for GPU (the same used in the experiments presented in Section 4.3). In this experiments we used the application LUD for problem size of 16384, with OpenMP and CUDA parallel models.

Figure 6 displays the results using the OpenMP parallel model in CPU. The average execution time was about 2 minutes and 54 seconds with the sampling rate between 1 and 3 Hz. There are two intervals of 30 seconds, one before T1 and another after T3. In these intervals ('padding'), only the monitoring tool was running to establish the baseline of power drain. The average of the power drain was of 170 W with an energy consumption of 29,240 J and the average of the power consumption measured with the monitoring tool running was of 182 W. So, the monitoring tool increased the power consumption by 12 W, with an energy consumption of 2,064 J. The energy consumption was calculated only when the application was running.

The LUD application started in T1 and finished in T3. While the application was running, it is possible to note two steps: *i*) between T1 and T2 the application was loading the matrix (serial region of code) and the average of power consumption was of 218 W - power consumption's increase of 36 W - with an energy consumption of 2,916 J, and *ii*) between T2 and T3, the application was solving the problem (OpenMP region of code), the average of power consumption was of 298 W - increase of 116 W - with an energy consumption of 10,556 J. The rise in temperature in the CPU (0), when the application was loading the matrix, indicates that only that CPU was executing the process. Between

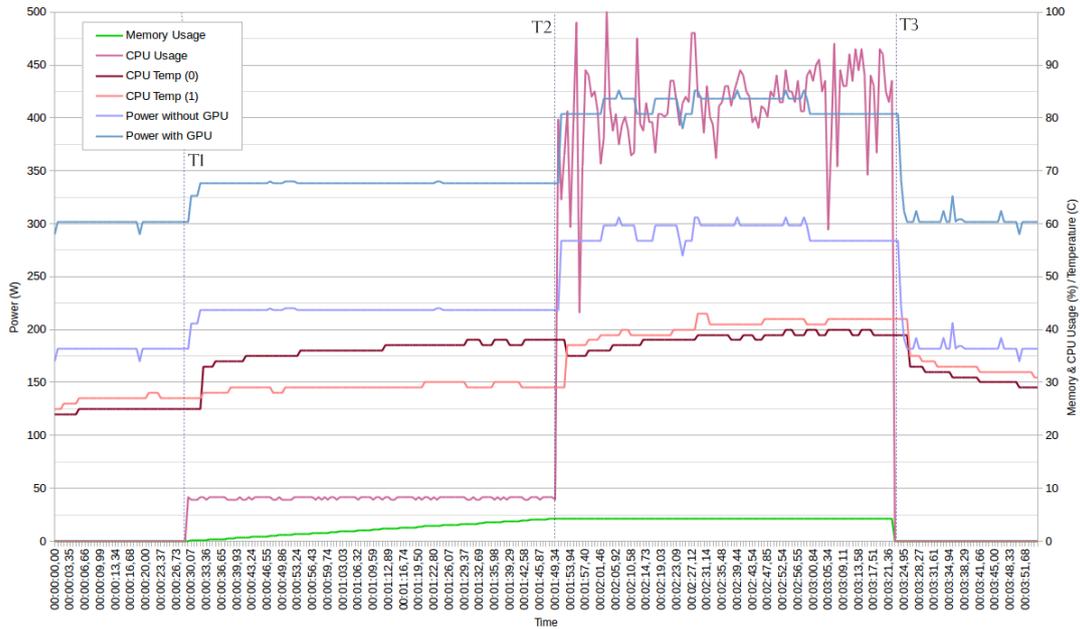


Figure 6. Results for the ComCiDis CPU collected by the IPMItool based monitoring application.

the interval T2 and T3 the parallel region of the code is started and both CPUs were executing the process. So, it is possible to observe the rise of the temperature in both CPUs and the peak of the power consumption and CPU utilization rate.

		CPU			GPU		
		System	Monitor	App	System	Monitor	App
Energy (J)	Serial Region	13,770	972	2,916	17,989	811	1,758
	Parallel region	15,470	1,092	10,556	5,721	258	4,990
Power (W)	Serial region	170	12	36	266	12	26
	Parallel region	170	12	116	266	12	232

Table 4. Results for the ComCiDis's CPU and GPU.

Table 4 compares the results for the energy and power consumption by system, monitor and application, while the application was running on serial and parallel region of the code. In the column *System*, can be observed that the power consumption by the GPU system was greater than by the CPU system, increased by GPU devices. For the application (*App*), it's possible to note that power consumption on serial region was much smaller than on parallel region for either CPU and GPU. Even the power consumption on parallel region in the GPU was greater than in the CPU. Although the power consumption was greater, given the execution time of the application, the energy-to-solution in the parallel region in CPU was smaller than for GPU. The power consumptions by the monitors (*Monitor*) were the same for both CPU and GPU, but the energy consumptions were different because the execution time on GPU was less than on CPU.

5. Final Considerations

In this work were analysed two directly measuring methods via internal sensor for two generations of NVIDIA GPUs. Also, we compare the power profile of GPU with CPU

using IPMItool for a deeper understanding of this method. For this we developed two monitoring tools, based on NVML and IPMItool.

Analyzing the results obtained with both monitoring approaches, as well as for different GPUs and problem sizes, it is possible to make some observations. With regard to different generations of sensors, and consequently the available sensors to measure temperature and energy, the precision of K40 in relation to M2050 is very high. This can be clearly observed for the smallest problem size A, where the profiling power is much coarse-grained for M2050. Also, the refreshing rate for K40 is very faster and its idle power drain is about 25 W against 60 W for M2050, indicating the improvement in the energy saving of the newer generations.

Regarding the monitoring applications, the sampling rate for NVML based monitoring application is much higher, reaching 500 Hz versus only 3 Hz for IPMItool. Thus, the accuracy of the power profile obtained with the NVML based application is much higher with incurred overhead of almost zero for the execution time and the power measured for GPU. Another advantage is that this method does not need *sudo* user to query the sensors, in order to enable the monitoring of environments that have high restrictions to access, like SDumond. Using the IPMItool based monitoring it was possible to compare the power consumption and estimate the energy consumption for GPU and CPU. The accuracy is the same for both architectures, since IPMItool query the same kind of sensors. The power profile is not so accurate, as demonstrated in results and there is an overhead about 2.5% of CPU utilization rate. Despite these limitations, in some cases its use is still a nice option, like in generations of GPUs that do not have built-in sensors. Besides that, this monitoring method allows to analyse the power profile, for the same application, in different architectures and parallel models. The major limitation is the need for *sudo* user.

In general, both monitoring tools were feasible for our studies on energy consumption of the applications. However, we conclude that the best option could be a monitoring tool that combines the two monitoring tools, since the NVML measures only the individual power levels for each GPU and IPMItool measures only the total power used by the whole computing node. Thus, it would be possible to develop a more accurate monitoring.

As future works we plan to develop this hybrid monitoring tool, to execute tests with a multi-GPU version of the real application and also to perform multiple experiments, similar to those presented using IPMItool, however, using different times of application launches in relation to the launch time of the monitor.

Acknowledgments

This work has received partial funding from the European Union's Horizon 2020 Programme (2014-2020) and from Brazilian Ministry of Science, Technology, Innovation and Communication through the Rede Nacional de Pesquisa (RNP) under the HPC4E Project (www.hpc4e.eu), grant agreement n 689772 and from the National Council for Technological and Scientific Research (CNPq). The authors would also like to thank CENPES/Petrobras.

References

Adhinarayanan, V., Subramaniam, B., and chun Feng, W. (2016). Online power estimation of graphics processing units. In *16th IEEE/ACM International Symposium on*

- Cluster, Cloud and Grid Computing, CCGrid 2016, Cartagena, Colombia, May 16-19, 2016*, pages 245–254. IEEE Computer Society.
- Bridges, R. A., Imam, N., and Mintz, T. M. (2016). Understanding gpu power: A survey of profiling, modeling, and simulation methods. *ACM Comput. Surv.*, 49(3):41:1–41:27.
- Burtscher, M., Zecena, I., and Zong, Z. (2014). Measuring gpu power with the k20 built-in sensor. In *Proceedings of Workshop on General Purpose Processing Using GPUs, GPGPU-7*, pages 28:28–28:36, New York, NY, USA. ACM.
- Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J. W., Lee, S.-H., and Skadron, K. (2009). Rodinia: A benchmark suite for heterogeneous computing. In *IISWC*, pages 44–54. IEEE.
- Coplin, J. and Burtscher, M. (2016). Energy, power, and performance characterization of GPGPU benchmark programs. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPS Workshops 2016, Chicago, IL, USA, May 23-27, 2016*, pages 1190–1199. IEEE Computer Society.
- Ge, R., Vogt, R., Majumder, J., Alam, A., Burtscher, M., and Zong, Z. (2013). Effects of dynamic voltage and frequency scaling on a k20 gpu. In *2013 42nd International Conference on Parallel Processing*, pages 826–833.
- Karami, A., Mirsoleimani, S. A., and Khunjush, F. (2013). A statistical performance prediction model for opencl kernels on nvidia gpus. In *The 17th CSI International Symposium on Computer Architecture Digital Systems (CADS 2013)*, pages 15–22.
- Kasichayanula, K., Terpstra, D., Luszczek, P., Tomov, S., Moore, S., and Peterson, G. D. (2012). Power aware computing on gpus. *Application Accelerators in High-Performance Computing, Symposium on*, 00:64–73.
- Lee, Y. and Kim, S. (2015). Empirical characterization of power efficiency for large scale data processing. In *2015 17th International Conference on Advanced Communication Technology (ICACT)*, pages 787–790.
- Mei, X., Wang, Q., and Chu, X. (2016). A survey and measurement study of GPU DVFS on energy conservation. *Digital Communications and Networks*.
- Menezes, G. S., Silva-Filho, A. G., Souza, V. L., Medeiros, V. W. C., Lima, M. E., Gandra, R., and Braganca, R. (2012). Energy estimation tool fpga-based approach for petroleum industry. In *Proceedings of the 2012 41st International Conference on Parallel Processing Workshops, ICCPPW '12*, pages 600–601, Washington, DC, USA. IEEE Computer Society.
- Mittal, S. and Vetter, J. S. (2014). A survey of methods for analyzing and improving gpu energy efficiency. *ACM Comput. Surv.*, 47(2):19:1–19:23.
- NVIDIA (2012). *NVML API Reference Manual*. <http://developer.download.nvidia.com/assets/cuda/files/CUDADownloads/NVML/nvml.pdf>.
- Wang, G. (2010). Power analysis and optimizations for gpu architecture using a power simulator. In *2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE)*, volume 1, pages V1–619–V1–623.