Compas'2019 : Parallélisme / Architecture / Système
LIUPPA - IUT de Bayonne, France, du 24 au 28 juin 2019

A macroscopic analysis of GPU power consumption

Dorra Boughzala[1][2], Laurent Lefevre[1], Anne-Cécile Orgerie[2]

[1]Univ. Lyon, Inria, CNRS, ENS de Lyon, Univ. Claude-Bernard Lyon 1, LIP
{dorra.boughzala,laurent.lefevre}@inria.fr
[2] Univ. Rennes, Inria, CNRS, IRISA, Rennes
anne-cecile.orgerie@irisa.fr

Abstract
For the last few years, High Performance Computing (HPC) systems have become highly heterogeneous through an unyielding integration of multi-core processors (CPUs) and accelerators. Among accelerators, Graphics Processing Unit (GPUs) have emerged as ideal co-processors for high-throughput applications, due to their enormous computational power and energy efficiency. However, these GPU-based systems still consume a lot of energy. In order to provide Exaflops, these extreme-scale systems must be optimized for "performance-per-watt". Similar to any optimization process, it is first required to understand how power is dissipated into general purpose processors. In this paper, we propose a macroscopic approach for analyzing the GPU power consumption through the benchmarking of a memory bound application kernel [1]. This approach offers great insights into inherent characteristics of GPU power behavior under different configurations.

Keywords : Exascale computing, GPUs, power consumption, energy efficiency, active SMs

1. Introduction

All technical reports on building an Exascale machine recognize the power consumption as the largest hardware challenge. To reach an Exascale performance, we should provide approximately 3-fold increase in energy efficiency. Therefore, energy efficiency has now become the major concern for the HPC community.
To address the power consumption challenge while continuing to provide tremendous performance, extreme-scale computing systems such as Top500 [2] and Green500 [3] lists, that rank the world's fastest and most energy-efficient supercomputers, have today multicore CPUs combined with low-power processors, or specialized accelerators (mix of GPUs, Intel Xeon Phis, or Field Programmable Gate Array (FPGAs)).
Indeed, heterogeneous computing has now become the most dominant paradigm in these systems. Today, GPUs are an integral part of most HPC systems and are used in conjunction with CPUs. The idea is that the CPU launches the application and handles as well serial operations, while the massively parallel parts are offloaded to the GPU. Thus, GPUs are now responsible

1. Experiments were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organisations (https ://www.grid5000.fr).
2. https ://top500.org
3. https ://green500.org

for a large fraction of energy used by these machines. For example, Summit [9], the number one system in the Top500 list (as of November 2018), powered by several CPUs and NVIDIA V100 GPUs, has a power consumption order of 9.7 MW.
For many years, GPUs have been dedicated graphics devices. Recently, they have evolved to support scientific computing across a wide range of massively parallel applications. Actually, general purpose GPGPUs architecture offers performance benefits over regular CPUs due to two main factors : higher number of parallel processors and higher memory bandwidth. Furthermore, as GPUs continue to develop, understanding their power and performance profiles on real applications is increasingly challenging [11].
In this article, we propose a macroscopic analysis to characterize the impact of applications on the GPU power consumption. By saying "macroscopic", we point to the fact that works in the literature rely on highly detailed-architecture approach. Our analysis provides information on the behavior of GPU under different configurations. Accurate understanding of power consumption of GPU is weakly addressed and only few works focus on the power profiling analysis. Regarding the position that GPUs hold in the HPC environment, it's mandatory to understand the power cost of GPU usage under different workloads and various configurations. This paper seeks to familiarize those interested in taking up GPU research with an overview of the GPU architecture and the CUDA execution model (Section 2), our proposed methodology to understand the GPU power cost (Section 3), then the experimental setup used for our study (Section 4) and results & analysis (Section 5) . Finally the state-of-art of GPU power profiling (Section 6) and our conclusion & future works are presented.

2. GPU & CUDA

In this section, we first discuss the evolution of GPU architecture and then we present the Compute Unified Device Architecture (CUDA) execution model that offers flexibility and efficiency for programmers to accelerate their applications. In this article, we focus on NVIDIA GPUs in CUDA environment.

2.1. NVIDIA GPU Architecture

Since the GPU architecture design is being advanced very fast, we have today several generations of GPUs. The G80 architecture was the initial model proposed by NVIDIA for GPU computing. Then, the Fermi architecture was proposed as the most significant leap forward the GPU computing. Furthermore, we describe in the following the Fermi architecture.
Each GPU has several streaming multiprocessors (SMs). Hundred to thousands of Stream Processors (SPs), known as CUDA cores, are organized into an SM, along with a few special function units (SFUs) and load/store units. CUDA cores execute the same instruction on different data, that's why we call them SIMD cores. From the memory hardware perspective, each SM includes register files and (read/write) shared/L1 cache memory. GPUs also contain an L2 memory, which is accessible by all SMs. On a GPU card, we have the global memory which is accessible by all threads. The access to this memory is very expensive, this explains why we have this memory hierarchy.

2.2. CUDA Execution Model

CUDA is both the platform and the programming model built by NVIDIA for developing applications on NVIDIA GPUs cards. The CUDA programming model is an extension to the C programming language, which makes it easy for programmers to port their applications to GPUs.

The heart of CUDA performance and scalability lies in its execution model. CUDA exposes an abstract view of the GPU parallel architecture in order to simplify the mapping of the parallelism with an application to hardware. Therefore, CUDA proposes three key programming abstractions : threads, blocks and grids. Actually, when a kernel is launched, a grid of thread blocks is generated and threads are grouped into blocks.
A thread block is a set of threads that can cooperate among themselves through barrier synchronization and shared memory. Each thread block has a block ID within its grid. A thread within a thread block executes an instance of the kernel, and has a thread ID that can be calculated uniquely from the indexes of the block and the grid it belongs to. We note that the CUDA terminology includes an important notion of "warp". A warp (a block of 32 consecutive threads) is the basic unit for scheduling work inside the SM.
CUDA provides two-level of scheduling : global and local scheduling. The global scheduling assured by the GigaThread Scheduler, which assigns one or more thread blocks to each SM. The local scheduling is assured by the SM warp schedulers, which select every clock cycle active warps and dispatch them to execution units.

3. Our Macroscopic Approach

We seek in our approach to explore input sensitiveness (data size, number of threads per Block) and the GPU design space (number of blocks/number of active SMs) in order to study their impact on the performance and power consumption.
For that, we did real measurements of the execution time and the power consumption of almost operations in our vector addition code. First, we run our application to study the data size effect on the execution time and the power consumption. Then, we run our kernel while varying the number of threads per block and we study the impact on the execution time and power consumption. Last, We run our kernel while varying the number of active multiprocessors from 1 to the maximum, as well as the number of blocks. The measures we have are averages of repeated experiments. Here is a brief description of our kernel outline : it starts by allocating arrays in the CPU memory, then filling them with numbers. Besides, it allocates arrays in the GPU memory and copy those vectors from the CPU memory to the GPU memory. Once the data is ready in the GPU memory, the kernel is launched by the CPU and executed in the GPU. Afterwards, it copies the result from the GPU memory to the CPU memory. Finally, it frees arrays from the GPU and CPU memory.

4. Experimental Setup

In this work, we rely on the the Grid'500 infrastructure [4], in particular on the orion cluster, due to the availability of a GPU card and accurate hardware wattmeters. We access the fine-grained measurements of "Omegawatt" wattmeters through the Grid'5000 API. Each node (CPU-GPU) is monitored individually by wattmeters which provide the electrical power consumed each 20 milli-seconds, with a precision of 0.1 watt. The orion cluster is composed of 4 homogeneous nodes, each consisting of 2 Intel Xeon E5-2630 with 6 physical cores per CPU, 32 GiB of RAM and an NVIDIA Tesla M2075 GPU card (installed in 2012). Our GPU is based on the Fermi 2.0 architecture, consisting of 14 SMs (448 CUDA cores), built on the 40 mm process. The Tesla M2075 is connected to the rest of the system using a PCI-Express 2.0 x16 interface. A brief description of resource characteristics of Tesla M2075 is presented in Table 1. The GPU is operating at a frequency of 574 MHz, memory is running at 783 MHz. Its power draw is defined at 225

4. https ://www.grid5000.fr

W. In addition, we use CUDA driver version 8.0 on a custom Debian GNU/Linux image.

Resources	Constraints
Number of SMs	14
Global memory size	5301 Mbytes
Number of cores/SM	32
Shared memory/Block	49152 bytes
Number of cores	448
Registers/Block	32768
Max Number of threads/SM	1536
Memory bus width	384-bit
Max number of threads/block capability	1024 2.0
Warp size	32

TABLE 1 – Tesla M 2075 resources description

5. Results & Analysis

Since we measure the power of the entire node (CPU+GPU), we subtract its idle power 156W from the total system power in order to obtain the idle GPU power. The idle power of the targeted GPU is thus 57W.
For the first case study, we use 1024 threads/block so we can explore larger data size. Indeed, the number of blocks is calculated based on the data size and the number of threads/block. Thus, if there are too many threads in a block to fit inside the resources of an SM, the kernel launch fails. Similarly different blocks may execute in parallel, depending on the number of SMs, the amount of shared memory, and the number of registers per SM. For the second case study, we choose a constant data size N=5000000, we just vary the number of threads per block from 128 to 1024. For the third case study, we run our kernel while varying the number of active multiprocessors from 1 to the maximum (for Tesla M2075, 14 SMs). Indeed, we vary the number of blocks such that only one block can be executed in each SM. More precisely, we vary the data size. All measurements in this paper are averages of samples repeated. We highlight here that for getting longer execution times and being able to observe the power consumption profile, we added a loop for the kernel AddVect(G). It's good to know as well that the copy from CPU to GPU includes copying 2 vectors of size N, while for the copy from GPU to CPU includes copying only one vector of size N.

5.1. Case study 1 : Data size impact

As we can see in Table 2, data size has not a huge effect on allocation of arrays either for CPU or GPU. However, the operation of copying from the CPU to the GPU and vice versa is almost linearly increasing when we increase the data size. Unsurprisingly, for the Addition operation, the execution time increases linearly to the data size. For characterizing the dynamic power consumption, we were able to observe only measurements for GPU malloc and GPU vector addition. We can say that the data size does not have a huge impact on the power consumption of the GPU. But, certainly an impact on the energy consumption, so optimizations should be done to reduce the execution time.

	Malloc(C)	Malloc(G)	CopyC2G	AddVec(G)	CopyG2C	Free(G)	Free(C)
N=500000	0.01	2218.67	1.66	23880.14	1.73	0.28	0.38
N=5000000	0.01	2216.51	12.76	235425.28	14.85	0.34	2.59
N=50000000	0.01	2216.95	123.75	2368287.0	144.21	0.61	22.92

TABLE 2 – Execution Time characterization in milliseconds.

	Malloc(C)	Malloc(G)	CopyC2G	AddVec(G)	CopyG2C	Free(G)	Free(C)
N=500000	-	37.9	-	146.9	-	-	-
N=5000000	-	37.2	-	146.5	-	-	-
N=50000000	-	41.2	-	146,7	-	-	-

TABLE 3 – Dynamic Power Consumption characterization in Average Watt.

5.2. Case study 2 : Thread Per Block impact

As we can see in Table 4 and 5, the execution time and the dynamic power consumption of the vectorAdd operation are slightly affected while varying the number of thread per block. This variation may be explained by the fact that the size N=5000000 is not a multiple of 32. So, at the execution, we don't have all blocks with the same number of threads per block. For example, we would have 9765 blocks with T/B=512 and a block with 320 threads, while we would have 19 531 blocks with T/B =256 and a block with 64 threads. We investigate to understand why with the configuration of T/B =256 we have less execution time of vectorAdd and more dynamic power consumption compared to the configuration with T/B= 1024. And the explanation that we found is that near the end of the execution, with the 256 configuration we have 2 active SMs (mean the SMs that are executing the program), while for the 1024 configuration, we have 11 active SMs. Therefore, we can say that idle SMs (mean the SMs that are on but not executing anything) still consume power.

	Malloc(C)	Malloc(G)	CopyC2G	AddVec(G)	CopyG2C	Free(G)	Free(C)
T/B= 128	0.01	12216	12.804	234380.4	14.60	0.35	2.43
T/B= 256	0.01	12221.8	12.76	233362.03	14.82	0.36	2.76
T/B= 512	0.01	12216.2	12.77	242002.4	14.64	0.32	2.53
T/B= 1024	0.01	12223.3	12.92	235320.53	14.28	0.34	2.59

TABLE 4 – Execution Time characterization in milliseconds.

	Malloc(C)	Malloc(G)	CopyC2G	AddVec(G)	CopyG2C	Free(G)	Free(C)
T/B= 128	-	44.3	-	141.6	-	-	
T/B= 256	-	31.5	-	150	-	-	-
T/B= 512	-	39.8	-	153.3	-	-	-
T/B= 1024	-	39.4	-	142	-	-	-

TABLE 5 – Dynamic Power consumption characterization in Average Watt.

5.3. Case study 3 : Active SMs & Number of blocks impact

For the GPU dynamic power consumption vs. active SMs, we can observe that we have an increase in dynamic power consumption as we increase the number of active SMs (Figure 1). The maximum power delta between using only one SM versus using all SMs is 61W . To check the linearity, we applied a linear regression fitting, which results in a standard error of 0.2. Thus, we can say that the power consumption increases linearly when we increase the number

of active SMs. For the GPU dynamic power consumption vs. number of blocks, we can observe that we have a log-based model (Figure 2) instead of a linear curve. This can be explained by the fact that the Thermal Design Power (TDP), which is the maximum amount of heat dissipated by the GPU, for the Tesla M2075 is 225W and we are already close with 210W. Running more blocks than SMs does not increase power consumption further.

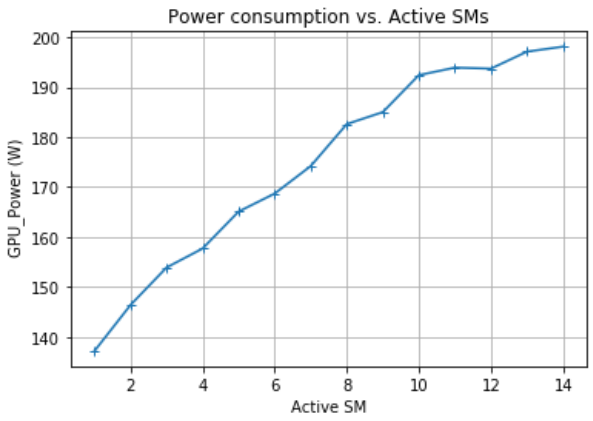

FIGURE 1 – Active SMs impact

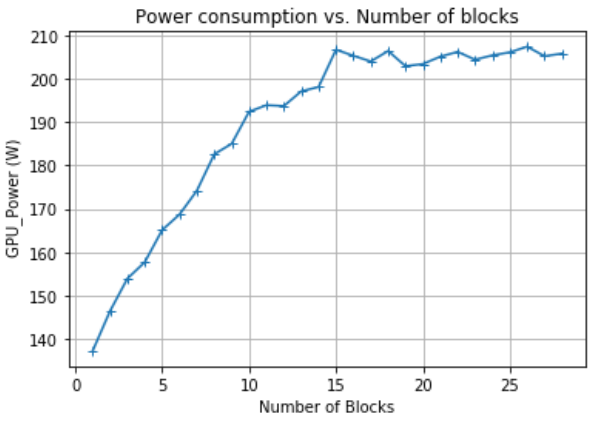

FIGURE 2 – Number of blocks impact

6. Related Works

There have been several efforts to model GPU power consumption in the literature. Today we have different types of models for predicting the power consumption, for example counter-based models such as [4], [8], [11] and simulators-based models such as [6], [7]. Few works focus on analyzing the power profiling and sharing its details in their power modeling methodology. In [3], S. Collange et al. investigate using measurements, how and where modern GPUs are using energy during various computations in a CUDA environment. The authors use various GPU kernels on different NVIDIA GPUs (G80, G92 and GT200) to stress GPU sub-components, in order to characterize power consumption of various GPU functional blocks with real measurements. Rofouei et al. [10], present an experimental investigation on the power and performance cost of GPU operations comparison versus a CPU-only system. S. Huang et al. [5] propose an empirical study of the performance, the power and energy characteristics of GPUs for scientific computing. Authors present also a comparison of the GEM GPU implementation versus a multi-threaded CPU version and the original serial CPU version. J. M.Cebrian et al. [2] analyze kernels taken from the CUDA SDK in order to discover resource underutilization. M.Burtscher et al. discuss in [1] unexpected behaviors when measuring GPU's power consumption, when working with k20 power samples.

7. Conclusion & Future works

Although the power profiling is part of the power modeling (measurements are used to calibrate models), it is not well-documented and studied in the literature. Actually, previous works take a detailed-architecture approach to study the power consumption. And this explains why

models present in the literature lack flexibility and simplicity. However, we believe that a detailed analysis of the GPU power profile is necessary to build an accurate and strong model. In this paper, we propose to take a macroscopic approach for studying the power consumption of a GPU. For that, we studied the execution time and power consumption while varying the data size, the number of Threads/Blocks and the number of active SMs & number of blocks. This approach helps us to characterize for almost operations, their power and time cost under different configurations. What we propose helps the reader to gain insight on some factors that impact our two metrics. As future work we plan to repeat the study on more recent GPU to highlight more characteristics. While studying the impact of varying the number of SMs on power consumption, we observed an irregular behavior in the power profile. As an initial observation, we refer this irregularity to the NVIVIDA scheduler. A more profound study is under exploration.

Bibliography

1. Burtscher (M.) et al. – Measuring gpu power with the k20 built-in sensor. – In *Proceedings of Workshop on General Purpose Processing Using GPUs, GPGPU-7*, New York, USA, 2014. ACM.
2. Cebri'n (J. M.) et al. – Energy efficiency analysis of gpus. – In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum*, May 2012.
3. Collange (S.) et al. – Power Consumption of GPUs from a Software Perspective. – In *9th International Conference on Computational Science*, Baton Rouge, Louisiana, USA, May 2009.
4. Hong (S.) et Kim (H.). – An integrated gpu power and performance model. *SIGARCH Comput. Archit. News*, vol. 38, n3, juin 2010, pp. 280–289.
5. Huang (S.) et al. – On the energy efficiency of graphics processing units for scientific computing. – In *2009 IEEE International Symposium on Parallel Distributed Processing*, May 2009.
6. Leng (J.) et al. – Gpuwattch : Enabling energy optimizations in gpgpus. *SIGARCH Comput. Archit. News*, vol. 41, n3, juin 2013, pp. 487–498.
7. Lucas (J.) et al. – How a single chip causes massive power bills gpusimpow : A gpgpu power simulator. – In *ISPASS 2013*, pp. 97–106, April 2013.
8. Nagasaka (H.) et al. – Statistical power modeling of gpu kernels using performance counters. – In *International Conference on Green Computing*, pp. 115–122, Aug 2010.
9. OakRidgeNationalLab. – Online- summit. – https ://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/.
10. Rofouei (M.) et al. – Energy-aware high performance computing with graphic processing units. – In *HotPower'08*, pp. 11–11, Berkeley, CA, USA, 2008. USENIX Association.
11. Song (S.) et al. – A simplified and accurate model of power-performance efficiency on emergent gpu architectures. – In *IEEE IPDPS 2013*, May 2013.