Accurate Analytic Models to Estimate Execution Time on GPU Applications *

Pedro Velho[1], Daniel A. G. de Oliveira[1], Edson L. Padoin[1,2],
Philippe O. A. Navaux[1]
[1]Institute of Informatics
Federal University of Rio Grande do Sul (UFRGS) – Porto Alegre, RS – Brazil
[2]Department of Exact Sciences and Engineering
Regional University of Northwest of Rio Grande do Sul (UNIJUI) – Ijui, RS – Brazil
{pvelho, dagoliveira, elpadoin, navaux}@inf.ufrgs.br

Abstract

Today top ranked HPC systems feature several GPUs which present high processing speed at low power budget with various parallel applications. Many scientific applications still claim for even more computing speed than the available today. A general approach to provide more processing speed is to scale the system. However, aspects such as interference, the amount of resources, heterogeneity of resources and failure probabilities hinder the research towards the future HPC systems. For that reason, many research use models to understand and estimate the behavior of the systems. GPUs are present and future of HPC systems, however the community lack of models to fast estimate the behavior of several GPUs with the desired accuracy and scale. The goal of this paper is to characterize the GPU resources to infer models that can be used in large scale simulators. To create this models with the two main concerns in mind, speed and accuracy, we conduct linear regression to estimate the time. Our test show that when the workload is within the range of realistic parameters the error is less than 11 %.

1. Introduction

HPC harness the processing speed of several processing cores simultaneously to solve problems. The large scientific community requires increasing processing speed over the years [4]. A common approach to improve the processing speed of current HPC systems is to use GPUs. GPUs are graphical processing units that originally aimed the market of rendering 3D scenes. The distributed nature of parallel applications match the GPU feature of supporting thousands of simultaneous threads. Today, top ranked HPC systems feature several GPUs, and scaling the number of GPUs seems to be an alternative to achieve high processing speed at low power budgets. However, the huge scale of current HPC systems hinder the complexity to analyze or develop application targeting these platforms for at least three reasons: huge amount of resources, Heterogeneity of resources, and probability of failures.

The three reasons cited above are a small set of the issues to conduct experiments in HPC environments. HPC is a research field, and experimentation is mandatory to understand and produce new knowledge. Hindered by the issues of conducting experiments in such environments several research findings in HPC rely on simulation or analytical modeling. Modeling is an answer to cope with heterogeneity, volatility, and the scalability to conduct experimental research on HPC in feasible time. Using models the environment is controllable, hence enabling reproducibility. However, to properly scale models need to be fast. Moreover, models must cope with GPU that feature on current state-of-the-art HPC systems.

Speed is generally used to evaluate the performance of the fastest supercomputers, usually measured in floating points operations per second (Flops). This way of ranking supercomputers neglect some important factors on the design of HPC systems. Increase the performance means to increase the number of processing elements. Nowadays, with the huge size of those machines the biggest wall that needs to be teared down is energy consumption. The energy consumption of the fastest supercomputer is close to the entire energy provided by a nuclear reactor. If we continue growing like this the next supercomputers will need a nuclear power plant entirely just to power up [4].

The goal of this paper is to characterize the GPU re-

* This work was supported by CNPq, CAPES, FAPERGS and FINEP. This research has been partially supported by CAPES-BRAZIL under grants 3471-13-6. Work developed on the context of the associated international laboratory between UFRGS and *Université de Grenoble* - LICIA.

sources to infer models that can be used in large scale simulators. The model conception is based on three steps: sending data from the host CPU to the GPU, executing the data by the GPU, and finally getting back the results from the GPU. To create this models with the two main concerns in mind, speed and accuracy, we conduct linear regression. Therefore, the model is fast with a few algebraic operations. To evaluate accuracy, we split our sample in two working sets, one to infere the model and another to evaluate the error. Our test show that when the workload is within the range of realistic parameters the mean error is less than 1 %, and 11 % in the worst case.

The remainder of this paper is structured as follows. In Section 2 we present related work on GPU simulation and characterization. Section 3 describes GPUs and how can we model it. Section 4 we present the methodology using during tests. After, in Section 5 we analyse the behaviour of the benchmarks used and verified the error in using linear regression models. Finally, in Section 6, we outline our conclusions, contributions and future work perspectives.

2. Related Work

Bakhoda *et al.* [2] present GPGPU-Sim that is a cycle-accurate simulator of GPUs. The simulator support CUDA Parallel Thread Execution (PTX) instruction set, it can run application without source code modifications, but requires access to the source code. Although it can give good time prediction, this approach requires unfeasible execution time to simulate one simple HPC application. Therefore, this simulator is very accurate but unsuitable for large scale research. Collange *et al.* [3] developed a modular functional GPU simulator based on the framewok UNISIM, it is faster than a cycle-accurate simulator but also takes too much time, orders of magnitude more time than than the application on a real hardware.

Kerr *et al.* [6] present an empirical evaluation of 25 applications on GPUs. Using a combination of instrumentation and statistical analysis, they attempt to predict the performance of similar classes of applications on different processors.

Zhang *et al.* [9] developed a throughput model considering three components of GPU applications, the instruction pipeline, shared memory access and global memory access. Their model is based on the GPU's native instruction set, and they reported results of a prediction with a 5-15% error.

Jiao *et al.* [5] used FFT and BLAS operators to analyze the power and energy efficiency of GPU and CPU when dynamically scaling voltage and clock frequency. Wang *et al.* [8] use a similar approach. They indicate that the use of GPUs is a straight forward path to achieve green computing.

Ren and Suda [7] present an approach to automatically allocate workload on CPU+GPU architectures aiming at improving energy efficiency. To allocate the workload, they use a model based on the power consumption of distinct modules.

In general, related papers focus on simulating one single GPU. This fine grain approach goal is to understand architectural choices to improve the application running on a single GPU or debugging purposes. In the other hand we aim at providing models to simulate several GPUs coping with each other in HPC environment. Therefore, our focus is to provide models less accurate but that can fast estimate the computing time of several GPU applications.

3. GPU Modeling

GPU is seen as a coprocessor by the CPU featuring an independent internal memory. Therefore, to run an application on GPU we first need to send the data and programs. After, the application start executing on the GPU. When the applications finishes the result need to be copied from the internal GPU memory to the main memory. Therefore, running a GPU application consist of three steps: **dispatch** the data to GPU memory, **execution** of the application, and **collect** the results from the GPU memory to the main memory

Our model to estimate GPU computing time follow this approach of three steps. The total computation time is hence the sum of dispatch, execution, and collect times. While the execution time is estimated by the GPU speed, the time for dispatch and collect is estimated by the bandwidth of the PCIe bus.

To create the model we assume that the time in each one of the steps is given by a linear function of workload. Therefore, the model is based on the hypotheses below:

(a) Dispatch time is a linear function of the input data ($W_{dispatch}$), i.e., the amount of data to copy from CPU memory to GPU, divided by the bandwidth of the dispatch. $\beta_{dispatch}$ is the error term of the linear regression;

$$T_{dispatch}(W_{dispatch}) = \frac{W_{dispatch}}{dispatchBandwidth} + \beta_{dispatch} \quad (1)$$

(b) Execution time is a linear function of workload (W_{exec}) with coefficient $\frac{1}{GPUSpeed}$;

$$T_{exec}(W_{exec}) = \frac{W_{exec}}{GPUspeed} + \beta_{exec} \quad (2)$$

(c) Collect time is a linear function of the output data ($W_{collect}$), i.e., the amount of memory to copy from GPU to CPU when the application have finished, divided by the bandwidth of the collect.

$$T_{collect}(W_{collect}) = \frac{W_{collect}}{collectBandwidth} + \beta_{collect} \quad (3)$$

(d) Total time of computing the GPU application can be obtained by the sum of the time estimated on the three steps .

$$T_{GPU} = T_{dispatch} + T_{exec} + T_{collect} \quad (4)$$

4. Methodology

In this section we detail our execution environment used to run the experiments. Next, we present three real applications used to characterize and model the GPU computing time. In the end, we show the choices of metric to measure accuracy.

4.1. Environment

The test platform was a computer with a quad-core Intel Core i7 930 operating at 2.80 GHz, the system features a x16 PCIe bus version 2.0 with total capacity of 8 GB/s. The GPU card is an NVIDIA GTX 480 with 1536 MB of global memory and 480 cores, each core operate at 1.40GHz.

The operating system running was ubuntu 11.04. the compiler was gcc in the version 4.4 and the CUDA was in the 4.0 version.

4.2. Applications

Several applications of different areas share similar problems, they have some recurrent routines, called kernels. A considerable portion of execution time is spent on these kernels. In the scientific report from the university of Berkeley, Asanovic *et al.* [1] pointed that only 13 dwarfs can characterize almost all scientific applications. Dwarfs are kernels, algorithm pieces, that characterize data access and computation patterns. Therefore, we evaluate GPU applications observing several application kernels. Precisely, we analyze three different algorithms, described below.

- **Matrix Multiplication**: this kernel is commonly found on solvers for linear equation systems and algebraic applications.
- **Fast Fourrier Transform**: also known as FFT, this kernel is useful for signal processing.
- **Needleman-Wunsch** : A dynamic programming kernel for sequence comparison commonly found in bioinformatics.

4.3. Error Metric

In our experiments we use the error metric described on the Equation 5. We rather use this metric because it respects the properties of symmetry and triangular inequality. This metric can also be easily transform into some discrepancy percentage using the Equation 6.

$$\epsilon = \|ln(a) - ln(b)\| \quad (5)$$

$$\text{Discrepancy percentage} = \epsilon^{\text{Error}} - 1 \quad (6)$$

To conceive the model we use 30 samples for random sizes of workload. From these samples one third, 10 samples, was used to calibrate the model and infer the transmission rate of the PCIe bus and the speed rate of the GPU. The others 20 samples were used to evaluate the accuracy of the model.

5. Results

All the models are linear. Therefore, the models are fast taking a constant time regardless of workload size. We evaluate next the quality of the models to verify in which conditions the model is accurate.

Equations 7, 8, and 9 show the final model for all three of them. Figure 1 shows the linear regression model for the matrix multiplication, the other two applications have similar graphs. Here the input parameters of workload correctly specify a different workload depending on the application kernel.

- **Matrix Multiplication**

$$T^{MM}(W^{MM}_{dispatch}, W^{MM}_{exec}, W^{MM}_{collect}) =$$

$$\frac{W^{MM}_{dispatch}}{5.68GB/s} + \frac{W^{MM}_{exec}}{241.10GFlops} + \frac{W^{MM}_{collect}}{2.13GB/s} + 0.001928 \quad (7)$$

- **FFT**

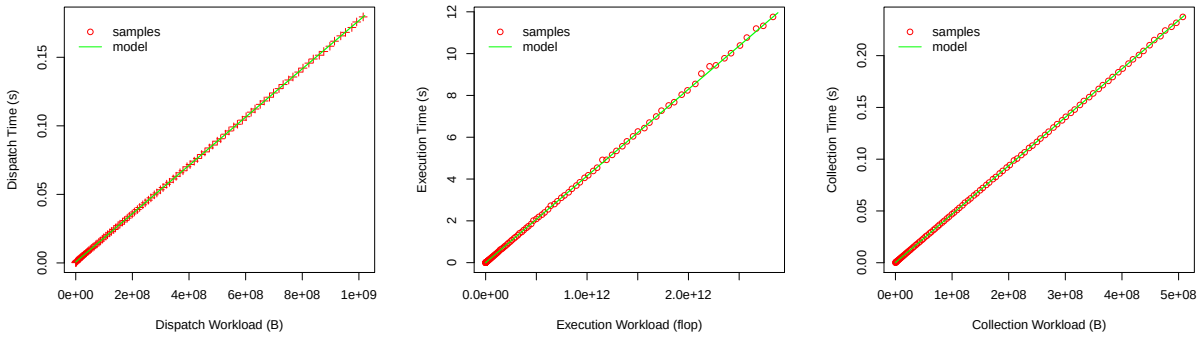
$$T^{FFT}(W^{FFT}_{dispatch}, W^{FFT}_{exec}, W^{FFT}_{collect}) =$$

$$\frac{W^{FFT}_{dispatch}}{5.44GB/s} + \frac{W^{FFT}_{exec}}{0.0004GFlops} + \frac{W^{FFT}_{collect}}{2.14GB/s} + 0.050523 \quad (8)$$

- **NeedleMan-Wunsch**

$$T^{NW}(W^{NW}_{dispatch}, W^{NW}_{exec}, W^{NW}_{collect}) =$$

$$\frac{W^{NW}_{dispatch}}{5.68GB/s} + \frac{W^{NW}_{exec}}{1.15GFlops} + \frac{W^{NW}_{collect}}{2.14GB/s} + 0.053851 \quad (9)$$



(a) Dispatch (b) Execution (c) Collect

Figure 1. Linear regression for each step for the matrix multiplication.

Evaluating the final model hypothesis, which is the sum of the previous hypotheses, we have a very good model accuracy. The maximum error was 11 % with the Matrix Multiplication kernel, using only realistic workloads where the GPU platform can be fully occupied. For FFT and NeedleMan-Wunsch, the maximum error for the sum was less than 8 %. Therefore, the final model is very fast and has outstanding accuracy.

6. Conclusion

This paper presented the characterization of GPU applications through three steps, dispatch, execution, and collect. The model depends on applications kernel, which are the basic building block to complex scientific HPC applications. More precisely, we tested three GPU application kernels: Matrix Multiplication, Fast Fourier Transform, and NeedleMan-Wunsch. This characterization enables to estimate the computing time of GPU applications fast with good accuracy. The worst error found using realistic input values was less than 11 %. The model is very fast with constant complexity $O(1)$, just a few algebraic operations. Therefore, these models are ideal to HPC simulators aiming exascale studies with several GPU capable hosts.

For future work, we will extend the GPU model for more commonly found kernels of scientific applications, and conceive a model to estimate the energy spent by the kernel.

References

[1] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick. The landscape of parallel computing research: A view from berkeley. Technical report, Electrical Engineering and Computer Sciences University of California at Berkeley, December 2006.

[2] A. Bakhoda, G. Yuan, W. Fung, H. Wong, and T. Aamodt. Analyzing cuda workloads using a detailed gpu simulator. In *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, pages 163 – 174, april 2009.

[3] S. Collange, M. Daumas, D. Defour, and D. Parello. Barra: A parallel functional simulator for gpgpu. In *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*, pages 351 –360, aug. 2010.

[4] C.-H. Hsu, W.-c. Feng, and J. S. Archuleta. Towards efficient supercomputing: A quest for the right metric. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 11 - Volume 12*, IPDPS '05, pages 230.1–, Washington, DC, USA, 2005. IEEE Computer Society.

[5] Y. Jiao, H. Lin, P. Balaji, and W. Feng. Power and performance characterization of computational kernels on the gpu. In *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on & Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*, pages 221–228. IEEE.

[6] A. Kerr, G. Diamos, and S. Yalamanchili. Modeling gpu-cpu workloads and systems. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, pages 31–42. ACM, 2010.

[7] D. Q. Ren and R. Suda. Investigation on the power efficiency of multi-core and gpu processing element in large scale simd computation with cuda. In *International Conference on Green Computing*, pages 309–316. IEEE, 2010.

[8] G. Wang and X. Ren. Power-efficient work distribution method for cpu-gpu heterogeneous system. In *International Symposium on Parallel and Distributed Processing with Applications*, pages 122–129. IEEE, 2010.

[9] Y. Zhang and J. Owens. A quantitative performance analysis model for gpu architectures. In *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, pages 382–393. IEEE, 2011.