Performance Prediction Model and Analysis for Compute-Intensive Tasks on GPUs

Khondker S. Hasan, Amlan Chatterjee, Sridhar Radhakrishnan, and John K. Antonio

School of Computer Science
University of Oklahoma
110 W. Boyd St., Norman
OK 73019, USA
{shajadul,amlan,sridhar,antonio}@ou.edu

Abstract. Using Graphics Processing Units (GPUs) to solve general purpose problems has received significant attention both in academia and industry. Harnessing the power of these devices however requires knowledge of the underlying architecture and the programming model. In this paper, we develop analytical models to predict the performance of GPUs for computationally intensive tasks. Our models are based on varying the relevant parameters - including total number of threads, number of blocks, and number of streaming multi-processors - and predicting the performance of a program for a specified instance of these parameters. The approach can be used in the context of heterogeneous environments where distinct types of GPU devices with different hardware configurations are employed.

Keywords: Compute-Intense Kernels, CUDA, GPU, Modeling and prediction.

1 Introduction

The availability of GPUs as commodity hardware and co-processors to CPUs, and the relative low cost-to-performance ratio has propelled these devices to the forefront or research in both academia and the industry. Compute Unified Device Architecture (CUDA), an extension to the C programming language, allows general-purpose problems to be solved using Nvidia GPUs. However, this paradigm shift has left developers striving for better performance from the available hardware. The general trend has been focusing on transferring compute intensive portions of tasks to the GPUs and exploiting parallelism in the existing code. Therefore, most of the research and studies relevant to GPUs focus on transferring the data from the CPU to the GPU and back, designing efficient data structures for the GPU, and utilizing available primitives to decrease memory latency [2, 4]. In reality, to exploit the full potential of these devices, understanding the underlying architecture and the basics of the programming model are required.

C.-H. Hsu et al. (Eds.): NPC 2014, LNCS 8707, pp. 612–617, 2014.

In this paper, we focus our research on optimally using the device at hand. We concentrate on deriving analytical models that can predict the *execution efficiency* of GPU programs i.e., GPU *kernels* based on certain parameters like the total number of resident blocks, number of threads in each block, the total number of blocks spawned in the device, the total number of streaming multiprocessors available in the device, and others. The importance of our models are in the fact that it will help developers to unleash the full potential of the available hardware by predicting execution efficiency of thread blocks before placing them in the GPU run queue and also by providing suggestions to improve the efficiency by making arrangements for optimal execution time. The benchmark programs used for empirical analysis of our analytical model are professionally developed programs from the Nvidia CUDA GPU Computing SDK [8] for demonstrating the reliability and accuracy of our proposed model. The benchmark programs are tested in both Tesla C1060 and Kepler 20 devices, respectively and achieved very low prediction error with an error bound of 0.13-5.69%.

2 Analytical Models

An analytical framework for estimating the overall execution efficiency for batches of thread blocks is derived for GPU systems. The introduced prediction model incorporates the following three major observed categories while GPU executes several blocks of threads:

- When the number of blocks to be executed is more than the aggregate number of resident blocks for the GPU.
- When the number of warps to be executed is more than the aggregate number of resident warps for the GPU.
- When the total number of threads to be executed is more than the aggregate number of resident threads for the GPU.

Table 1 contains the notation and definitions of required parameters of the model. The execution time is ideal, denoted by τ, when the number of blocks and threads in each block is less than the aggregate resident blocks and threads for the GPU (this number varies by GPU architecture). The following model incorporates the effect for the case when the number of thread blocks spawned for execution is more than the aggregate resident blocks for all multi-core processors of the GPU; for measuring the execution time (ξ_{rb}):

$$\xi_{rb} = \begin{cases} \tau & \text{if } \beta \leq (\rho_b \times \rho_{sm}), \\ \text{Max}\left(1, \left\lceil \frac{\beta}{\rho_b \times \rho_{sm}} \right\rceil\right) \times \tau & \text{else} \end{cases} \tag{1}$$

A higher occupancy reduces processor idle time (SM may stall due to unavailability of data or busy functional units) and improves overall performance [9], [10]. The estimated execution time model can be derived as:

$$\xi_{rw} = \left\{ \frac{\beta \times \vartheta}{(\rho_w \times \sigma_w \times \rho_{sm})} \right\} \times \tau \tag{2}$$

Table 1. Terms and definitions of GPU efficiency prediction model parameters

Terms	Definition
ρ_{sm}	Number of Streaming Multi-core processors (SM) in the device.
ρ_w	Maximum number of resident warps in a streaming multi-core processor.
ρ_b	Maximum number of resident blocks in a streaming multi-core processor.
ρ_t	Maximum number of resident threads in a streaming multi-core processor.
σ_w	Size of warp. Size for both Tesla C1060 and Kepler 20 is 32.
σ_b	Maximum size (number of threads) of a block.
β	Number of blocks spawned in the GPU device.
ϑ	Number of threads per block (block size).

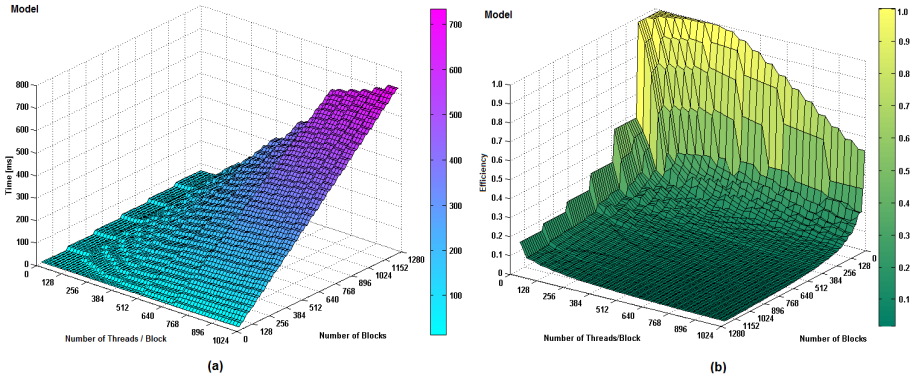

Fig. 1. Surface diagrams by deploying the introduced prediction model when ϑ is increased from 32 to 1024, β increased from 16 to 1248, and $\tau = 14.5$ (a) shows measured total execution time surface (b) shows measured efficiency surface diagram (associated with Eq. 3).

The composite prediction model which incorporates all major specified effects includes the input parameters: ideal thread execution time, execution efficiency when $\beta > (\rho_b \times \rho_{sm})$, and execution efficiency when threads in a SM $> (\rho_w \times \sigma_w)$. The model equation is derived from Eqs. 1 and 2 to reflect the observed effect in total thread execution time (κ).

$$\kappa = \begin{cases} \frac{\tau}{(\tau + \xi_{rw})}, & \text{when } (\beta \times \vartheta) \leq (\rho_w \times \sigma_w \times \rho_{sm}) \\ \frac{\tau}{(\xi_{rb} + \xi_{rw})}, & \text{when } (\beta \times \vartheta) > (\rho_w \times \sigma_w \times \rho_{sm}) \end{cases} \tag{3}$$

When the value of $(\beta \times \vartheta) \leq (\rho_w \times \sigma_w \times \rho_{sm})$, the τ is divided by an expression which incorporates the overhead of warp occupancy. Next, when $(\beta \times \vartheta) > (\rho_w \times \sigma_w \times \rho_{sm})$, the τ is divided by an expression which contains the overhead of both $\beta > \rho_b$ and $\vartheta > \rho_t$ expressed in Eqs. 1 and 2.

Figure 1 (a) shows the composite effect of increased blocks and threads in a block on total execution time of the GPU kernel. In this figure, two horizontal axes represents ϑ and β respectively and the vertical axis represents the total execution time in milliseconds. With careful observation, it can be seen that for each case when $\beta > (\rho_b \times \rho_{sm})$, the execution time jumps by the value of τ. Similarly, when $(\beta \times \vartheta) > (\rho_w \times \sigma_w \times \rho_{sm})$, for each $(\rho_w \times \sigma_w \times \rho_{sm})$, the execution time increases utilizing Eq. 2. The total execution time increases sharply for large number of blocks and its sizes. Figure 1 (b) depicts the execution efficiency surface measured by Eq. 3. The efficiency surface diagram clearly visualizes the composite effect of increased number of β and ϑ. The model depicts a sharp performance degradation as soon as the $\beta > \rho_b$.

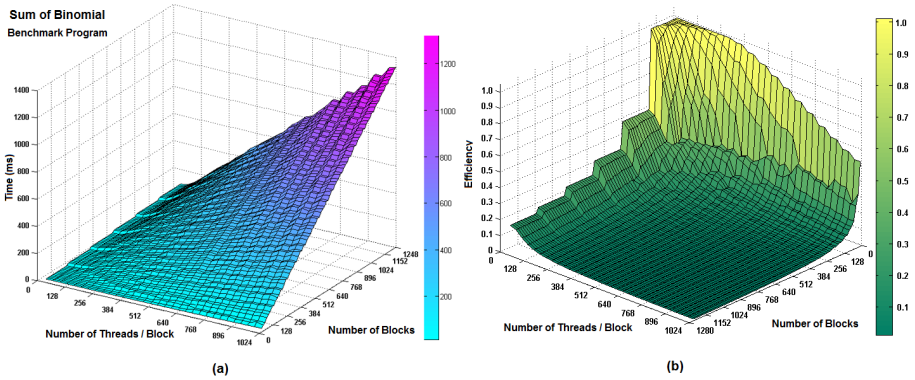

Fig. 2. Surface diagrams for the Sum of Binomial Series benchmark program (a) shows increasing run-time while ϑ increased from 32 to 1248 and β increased from 16 to 1024 (b) shows measured efficiency surface diagram with respect to τ

3 Empirical Studies

The purpose of this experimental study is to empirically measure the execution efficiency of kernels in GPUs as a function of aggregate blocks and threads in each block. The *Sum of binomial series* benchmark program is utilized from Nvidia CUDA SDK to conduct the empirical work to ensure real-world applications adaptability and accuracy. Two different GPU devices are used for evaluating the proposed efficiency prediction models for parallel thread execution. We have used Kepler 20 and Tesla C1060 GPU cards for empirical studies in heterogeneous environments.

It can be observed from Table 2 that for $\beta = 64$ and ϑ ranging from 32 to 320, as the $(\beta \times \vartheta) > \rho_t$, the execution time increases around $(2 \times \tau)$ depending on the number of warps. This behavior is modeled using Eq. 2. Next, when the total number of threads spawned $(\beta \times \vartheta)$ is more than the occupancy, $(\rho_w \times \sigma_w \times \rho_{sm} = 64 \times 32 \times 13 = 26,624$ for the Kepler 20 device), the specified series in Section 2

Table 2. Empirical results using **Kepler 20** GPU device for the Sum of Binomial Series benchmark program

β	Number of Threads per Block (ϑ)									
	32	**64**	**96**	**128**	**160**	**192**	**224**	**256**	**288**	**320**
16	14.61	14.61	14.62	14.63	14.62	14.62	14.62	14.62	14.63	14.64
64	14.61	14.60	14.63	14.62	15.42	15.91	16.90	17.73	21.34	22.57
112	14.60	14.53	15.20	16.69	21.38	24.35	27.93	42.31	42.74	41.72
160	14.54	15.19	17.84	22.47	40.29	40.58	44.35	45.28	52.16	65.71
208	14.55	15.83	20.80	27.64	43.00	45.12	52.69	55.23	70.21	77.90
224	29.02	30.36	35.30	42.16	42.97	46.65	56.08	69.69	75.50	78.63
272	29.05	30.37	35.47	43.40	50.60	66.41	70.39	79.28	84.12	95.45
320	29.07	30.54	36.56	45.43	67.23	71.02	80.39	98.46	102.9	117.3
368	29.08	31.60	39.95	51.77	69.03	78.60	96.55	103.4	123.5	127.1
416	29.02	31.63	41.58	55.38	75.08	93.54	102.2	110.4	129.7	147.5
432	43.45	46.13	56.03	69.73	78.60	94.69	105.3	124.9	134.9	147.9
480	43.50	46.15	56.19	70.99	93.14	99.24	124.6	134.3	153.7	170.8
528	43.51	46.32	58.16	74.42	94.91	117.8	132.6	152.3	164.8	178.4
576	43.50	46.75	60.69	79.36	101.6	120.7	139.1	161.9	181.4	199.4
624	43.51	47.41	62.39	83.10	105.8	130.2	152.4	165.5	192.9	209.2
640	57.87	61.91	76.75	97.30	118.9	131.1	156.4	180.1	203.4	222.6

can be observed in the Table 2. For $\beta = 208$ and $\vartheta = 128$, the number of threads spawned in the device is $208 \times 128 = 26,624$ (i.e., maximum occupancy reached)[1]. As soon as the ϑ or β increases, the execution time increases depending on the number of execution cycle. This observed behavior is modeled using Eq. 3.

Figure 2 (a) and (b) shows the complete measured *run time surface* and *efficiency surface* (using Eq. 3) diagram for the sum of binomial series benchmark program. Both surfaces depict the complete 2496 independent test run results to capture all possible scenarios by varying the values of β and ϑ. It can be observed from Figure 2 (a) that each time when $\beta \times \vartheta$ crosses $(\rho_w \times \sigma_w \times \rho_{sm})$, the execution time increases for new scheduling cycles. Similarly, when β crosses $(\rho_b \times \rho_{sm})$, the execution time increases by τ. It can be observed from Figure 2 (b)that the efficiency value decreases significantly when β and ϑ increases beyond the capacity of scheduling cycle. Both surface diagrams clearly depicts same behavior and shape as compared to the models' run-time and efficiency surfaces in Figure 1 respectively.

4 Conclusion

This paper has introduced prediction models to forecast the execution efficiency of GPUs for computationally intensive kernels. The key challenge was to determine the arrangement of blocks and threads in a block prior to placement of

[1] Maximum number of resident threads in a Kepler 20 SM is 2048 though a thread block can contain a maximum of 1024 threads.

threads into the run queue. The model has been validated with Nvidia CUDA GPU SDK benchmark program for accuracy. The provided surface diagrams depict clear visualization of measured efficiency based on variable number of blocks and size of blocks. The empirical studies performed on the prediction model show that the model surface follows the same shape and pattern of the real-world benchmark programs with only $0.13 - 5.69\%$ prediction error. The conducted study is highly useful for understanding and optimizing performance on GPUs and useful in the context of heterogeneous environment to choose the device with a better performance potential.

References

1. Sim, J., Dasgupta, A., Kim, H., Vuduc, R.: A Performance Analysis Framework for Identifying Potential Benefits in GPGPU Applications. In: 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP 2012), New Orleans, Louisiana, pp. 11–22 (2012)
2. Khondker, S., Hasan, J.K.: Antonio, and Sridhar Radhakrishnan, "A New Composite CPU/Memory Model for Predicting Efficiency of Multi-core Processing". In: The 20th IEEE International Symposium on High Performance Computer Architecture (HPCA 2014) Workshop, Orlando, FL, February 15-19 (2014)
3. Holmen, J.K., Foster, D.L.: Accelerating Single Iteration Performance of CUDA-Based 3D Reaction–Diffusion Simulations. International Journal of Parallel Programming 42(2), 343–363 (2014), doi:10.1007/s10766-013-0251-z
4. Chatterjee, A., Radhakrishnan, S., Antonio, J.K.: Data Structures and Algorithms for Counting Problems on Graphs using GPU. International Journal of Networking and Computing 3(2), 264–288 (2013)
5. Hasan, K.S., Radhakrishnan, S., Antonio, J.K.: Composite Prediction Model and Task Distribution on a Cloud of Multi-core Processors. In: IEEE International Conference on High Performance Computing (HiPC 2014) Workshop, Bangalore, India (December 2013)
6. Kepler Compute Architecture Technology in a Brief (2012), `http://www.nvidia.com/content/PDF/kepler/NV_DS_Tesla_KCompute_Arch_May_2012_LR.pdf`
7. Zhang, Y., Owens, J.D.: A quantitative performance analysis model for GPU architectures. In: 2011 IEEE 17th International Symposium on High Performance Computer Architecture (HPCA), pp. 382–393 (2011)
8. Nvidia CUDA GPU SDK, Sample CUDA Toolkits, `http://docs.nvidia.com/cuda/cuda-samples/`
9. CUDA Warps and Occupancy. GPU Computing Webinar (July 2011), `http://on-demand.gputechconf.com/gtc-express/2011/presentations/cuda_webinars_WarpsAndOccupancy.pdf`
10. Lam, S.K.: CUDA Performance: Maximizing Instruction-Level Parallelism (September 2013), `http://continuum.io/blog/cudapy_ilp_opt`