GPU Approximation Acceleration For Scientific Applications

Ang Li, Shuaiwen Leon Song
HPC Group, Pacific Northwest National Laboratory (PNNL)
{Ang.Li, Shuaiwen.Song}@pnnl.gov

I. Introduction

Despite the conventional belief that being exact remains the default attribute for computing, for many promising applications, such as big data, machine learning and multimedia processing, extremely accurate compliance of the produced results is often not an essential requisite. This undoubtedly offers new opportunities for application speedup or the associated power reduction at the expense of modest precision loss [1]. Such precision loss is only acceptable when it is within the tolerance range of the user-defined quality-of-service (QoS) [2], which heavily depends on the specific application domain. Besides, many of these applications are data-parallelism intensive, making them well-suited candidates for the emerging general-purpose GPU computation (GPGPU) [3]. Concerning the above reasons, approximate computing has become an attractive research topic for GPUs [4], [5], [6], [7], [8], [9].

However, most existing GPU approximation designs are targeted for data-intensive applications [4], [5], [7], [9], which are comparatively more error-tolerable. Furthermore, they primarily rely on the spatial or temporal locality among the nearby-data or the consecutive functions so as to approximate the requested data/computation based on their neighboring[4], [5], [8], [9] or locally stored historical values [5], [6], [7], [9]. Such approaches, although quite efficient, may commit uneven errors across data elements or even catastrophic failures since the locality is not always held and the distortion to the final results could be considerable. Moreover, for the numerical-intensive scientific applications (e.g., various simulation and molecular dynamics) that are usually sensitive to accuracy loss, the current techniques are often not suitable. This is because even a relatively smaller error introduced in an intermediate result may potentially propagate and be significantly amplified when such applications are deployed in a supercomputer environment with thousands of working GPUs [10], [11]. Therefore, gaining performance while offering lower but still tractable assurance on accuracy loss becomes the major obstacle for applying approximation techniques to accuracy-sensitive applications on GPUs.

In fact, the concept of approximate computing is not new for GPUs the similar design paradigm has already been well utilized for graphic applications (e.g., geometric transformation). These applications often involves transcendental functions and interpolation calculations which demand substantial computation power. As ultra-high accuracy is mostly not a crucial demand for images and videos that display in a screen, modern GPUs largely rely on hardware-based accelerators which sacrifice the accuracy of last few digits, for the compensation of performance boosting in such calculations.

Although SFUs plays an indispensable role in the conventional graphic-pipeline processing, they are almost completely missing in state-of-the-art GPGPU applications. The major reasons are two folds: First, most application developers did not realize the outstanding capability of SFUs, nor did they offered with sufficient materials to learn. As far as we know, this is the first manuscript that exhaustively describes, models and explores the the characteristics and functionalities of SFUs in GPUs. To many developers, the only knowledge about SFU, however, is that you may gain performance speedup for single-precision floating-point computation if passing the - use fast math compiler option, which utilizes the SFUs when applicable. Second, when people tentatively set the fast-math option, they may surprisingly find that the accuracy of the outcome degrades precipitously, sometimes to an extent that cannot be accepted. Therefore, they roll back the code and give up the adoption of SFUs. For example, significant performance improvement has been observed after well-utilization of the SFUs (up to 5.8x) on all the three GPUs in different generations of NVIDIA products. However, using the original differences of GPU results with the CPU results as the reference, after applying SFUs, the accuracy of some applications drops by up to 90%. Therefore, a practical approach that can greatly harvest the SFU performance while at the time assures the final accuracy is in great desire.

To address this challenge, we explore a very important but often ignored approximation unit on GPUs the Special Functional Unit (SFU), and unveil its crucial role in performance acceleration for accuracy-sensitive scientific applications in the context of approximate computing. To better understand its approximation potentials, we first evaluate all the nine single precision and four double-precision numeric transcendental functions that could be accelerated by SFUs, in terms of performance, accuracy and power. Using the insights, we then leverage the GPU SIMT execution model to dynamically partition warps into executing two versions of the numerical computation: an accurate but slower version (i.e., SPU or DPU version) and a faster but approximate version (i.e., SFU version), and then tune this partition ratio to control the trade-offs between the performance and accuracy, or power and

accuracy. This software approach successfully introduces a relatively large, uniform and fine-grained tuning space. To accompany this design, we also propose an efficient heuristic searching method to quickly locate the optimal partition ratio that delivers the best performance under user-defined QoS. Finally, we compact the approach and its searching method into a transparent, tractable and portable SFU-centric approximate acceleration framework, which is then validated on multiple GPU architectures for its effectiveness. We have published the initial results for this line of research in [12]. To further understand the detailed design methodology of our framework, experimental results and analysis, please refer to that paper. Of course, the basic idea discussed in this poster can be applied to large-scale GPU-based HPC systems for accelerating future numerical-intensive scientific applications in exascale.

II. Conclusion

In this position paper, we focus on a crucial GPU component which however, has long been ignored — the Special Function Units (SFUs), and show its outstanding role in performance acceleration and approximate computing for GPU applications. Based on a thorough hardware performance and energy evaluation, we proposed a transparent, tractable and portable design framework for SFU-driven approximate acceleration on commercial GPUs, requiring no hardware or application modification.

References

[1] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate Computing and the Quest for Computing Efficiency," in *Proceedings of the 52nd Annual Design Automation Conference*, ser. DAC'15. ACM, 2015, pp. 120:1–120:6. [Online]. Available: http://doi.acm.org/10.1145/2744769.2751163

[2] S. Misailovic, S. Sidiroglou, H. Hoffmann, and M. Rinard, "Quality of Service Profiling," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE'10. ACM, 2010, pp. 25–34. [Online]. Available: http://doi.acm.org/10.1145/1806799.1806808

[3] H. Wen-Mei, *GPU Computing Gems Emerald Edition*. Elsevier, 2011.

[4] M. Samadi, J. Lee, D. A. Jamshidi, A. Hormati, and S. Mahlke, "Sage: Self-tuning approximation for graphics engines," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. ACM, 2013, pp. 13–24.

[5] M. Samadi, D. A. Jamshidi, J. Lee, and S. Mahlke, "Paraprox: Pattern-based Approximation for Data Parallel Applications," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS'14. ACM, 2014, pp. 35–50. [Online]. Available: http://doi.acm.org/10.1145/2541940.2541948

[6] J. Sartori and R. Kumar, "Branch and data herding: Reducing control and memory divergence for error-tolerant GPU applications," *IEEE Transactions on Multimedia*, vol. 15, no. 2, pp. 279–290, 2013.

[7] J.-M. Arnau, J.-M. Parcerisa, and P. Xekalakis, "Eliminating redundant fragment shader executions on a mobile GPU via hardware memoization," in *Proceedings of the 41st ACM/IEEE International Symposium on Computer Architecture (ISCA)*. IEEE, 2014, pp. 529–540.

[8] A. Yazdanbakhsh, G. Pekhimenko, B. Thwaites, H. Esmaeilzadeh, T. Kim, O. Mutlu, and T. C. Mowry, "RFVP: Rollback-Free Value Prediction with Safe-to-Approximate Loads," in *Proceedings of the 11th International Conference on High Performance and Embedded Architectures and Compilers (HiPEAC)*. ACM, 2016.

[9] M. Sutherland, J. San Miguel, and N. E. Jerger, "Texture Cache Approximation on GPUs," 2015.

[10] D. Fiala, F. Mueller, C. Engelmann, R. Riesen, K. Ferreira, and R. Brightwell, "Detection and Correction of Silent Data Corruption for Large-scale High-performance Computing," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC'12. IEEE Computer Society Press, 2012, pp. 78:1–78:12. [Online]. Available: http://dl.acm.org/citation.cfm?id=2388996.2389102

[11] R. A. Ashraf, R. Gioiosa, G. Kestor, R. F. DeMara, C.-Y. Cher, and P. Bose, "Understanding the Propagation of Transient Errors in HPC Applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC'15. ACM, 2015, pp. 72:1–72:12. [Online]. Available: http://doi.acm.org/10.1145/2807591.2807670

[12] A. Li, S. L. Song, M. Wijtvliet, A. Kumar, and H. Corporaal, "Sfu-driven transparent approximation acceleration on gpus," in *Proceedings of the 2016 International Conference on Supercomputing*, ser. ICS '16. New York, NY, USA: ACM, 2016, pp. 15:1–15:14. [Online]. Available: http://doi.acm.org/10.1145/2925426.2926255