An Adaptive Heterogeneous Runtime for Irregular Applications in the Case of Ray-Tracing (Extended Abstract)

Chih-Chen Kao, Wei-Chung Hsu

An Adaptive Heterogeneous Runtime for Irregular Applications in the Case of Ray-Tracing (Extended Abstract)

Chih-Chen Kao and Wei-Chung Hsu

Department of Computer Science, National Taiwan University, Roosevelt Road, Taipei, 10617 Taiwan
hsuwc@csie.ntu.edu.tw

Heterogeneous architecture has been widely adopted in various computing systems, from mobile devices to servers. However, optimizing the performance for such platforms remains challenging in three aspects: the control flow divergence decreases the utilization of SIMD components, the significant memory copy overhead between computing devices consuming precious memory bandwidth and the load imbalance that degrades the overall performance. In this paper, we proposed three methodologies: Intermediate Feedback, Dynamic Task Partitioning and Heterogeneous Runtime that work collaboratively to overcome the aforesaid problems. We adopted and implemented these methodologies in a heterogeneous runtime library derived from Intel Embree[1] and compared the performance results of the two frameworks running Ray-Tracing[2] on various scenes. Experiment results have shown that the performance gain from the proposed methods is significant, especially in complex scenes with a large amount of objects or with large input data sizes the CPU cannot handle efficiently.

Due to the performance and power efficiency potentials of GPGPU and heterogeneous systems, a wide variety of applications, which include molecular simulation, fluid dynamics, biomedical image processing and computer vision, have been developed by leveraging the aforementioned programming models. However, for this type of heterogeneous configuration, many challenges remain before the performance potential can be fully unleashed. Despite significant advantages of GPU programming, writing high-performance heterogeneous programs still require programmers to be familiar with GPU architecture. The performance of a heterogeneous program is significantly influenced by how the computations are mapped into threads and how those threads are scheduled onto distinct cores, the usage of GPU registers and memory hierarchy, the synchronization among all threads and data access, the data transfer between host and GPU memories and the control flow branch divergence issue. In short, a program which benefits from GPGPU must contain explicit parallelism, high regularity and data reuse. The irregularities in an application may throttle the expected performance of the GPU by as much as an order of magnitude.

The terms regular and irregular are often used in compiler literature. For example, in regular code, control flow and data memory references are not data dependent. Dense matrix multiplication operations are good examples of regular code. On the other hand, in irregular code, both control flow and data memory references could be data dependent. For example, graph-based applications are

considered irregular, because the connectivity and values of the nodes in a graph are unknown before the input graph is available, and the connectivity and values of the nodes determine which graph elements are accessed[3]. Regular programs can often be efficiently mapped onto the GPGPU computing unit, and many of them have been ported to heterogeneous systems to benefit from increased performance and power efficiency. However, duplicating the success of heterogeneous computing from regular programs to irregular programs is a challenge. Irregular programs are often operated around pointer-based data structures such as graphs, trees or linked lists that are difficult to map to conventional regular GPU architecture or SIMD components. The execution paths of an irregular program are often unpredictable and could also vary dramatically during runtime[4]. Furthermore, in contrast to regular programs, the data dependencies in irregular programs can only be resolved dynamically at runtime, making it difficult to design a proper task scheduler[5][6].

The obstacles of efficiently adopting irregular programs to heterogeneous system can be categorized as follows: Firstly, the irregular memory access (e.g. indirect array references, sparse matrix references) could lead to low effective memory bandwidth, and there is no proper static solution to adaptively determine which type of memory should be used for irregular programs during execution. Secondly, the dynamically varying control flows create thread divergences, which reduces the level of parallelism and SIMD lane utilization in GPU. Thirdly, the chain of input dependencies often causes load imbalance among multiple cores, which degrades the overall throughput. Finally, memory-bound pointer chasing exhibits low data locality and exposes increased data access latency on GPGPU. Therefore, mapping irregular code efficiently onto a heterogeneous system remains difficult[7][8].

What type of heterogeneous computing model could handle irregular programs more efficiently is still debatable. For example, Ray-Tracing is an irregular program that is intensively used for global illumination in multimedia applications and has been adopted for implementation on different heterogeneous models. At the early stage of GPGPU computing, Ray-Tracing is designed to run on a GPGPU because a GPGPU is capable of handling a large number of rays in parallel. However, the potential irregularities of Ray-Tracing decrease the utilization of SIMD/SIMT components in GPU, and thereby offer little performance/power advantages[9].

Recently, Intel has announced a Ray-Tracing framework called Embree[1] with a different design philosophy that is optimized for traditional CPU architectures augmented with medium size (i.e. 512 bits) of SIMD capability. The Embree framework leverages CPU threads with wider SIMD units by using a compiler framework called Intel SPMD Program Compiler. (ISPC)[10]. The framework was originally designed for single ray traversal using SSE or AVX-enabled CPUs but has been extended to support Intel Xeon Phi architectures[11]. Embree features spatial acceleration structures and traversal algorithms and claim to support efficient Ray-Tracing with MIMD architectures and medium size SIMD capability. However, we believe that, with proper runtime and innovative method-

ologies, we could make traditional CPU/GPU type of heterogeneous system more competitive on irregular applications. In this work, we use Ray-Tracing as a case study to show the potential of our proposed approaches.

In this work, we address the forenamed problems of irregular programs and proposed a feedback tuning mechanism that can be used to model a specific category of heterogeneous program where the input data is recursively modified and added back to the commonly shared database for the next computation. Previous research applied a statistic approach and heuristic. Our method is based on analyzing and monitoring the communication protocol and behavior among all modules in a program. A new methodology is introduced in this research that encodes the representative feature of a heterogeneous program gathered at runtime and sends them to adjacent modules for adjusting iterative computation to fit the given platform configuration in order to gradually fine tune the system performance. We proposed a dynamic task partitioning mechanism and heterogeneous thread pool in order to resolve issues related to branch divergence and load imbalance. To demonstrate the effectiveness of our proposed schemes, we evaluate our performance gain by implementing the methodologies into a runtime library which is derived from the Intel Embree Ray-Tracing framework and compare the execution time with the original version.

In Conclusion, we explore the performance potential of mapping irregular programs onto heterogeneous systems by taking a Ray-Tracing algorithm as a case study. Three methodologies, Intermediate Feedback, Dynamic Task Partitioning and Heterogeneous Thread Pool, are introduced in this framework. The experiment results shows that our proposed methods could benefit heterogeneous computing resources and thereby increase the system performance, especially for handling complex scenes and for large input data sizes. We believe that the proposed methods could be applied to other heterogeneous frameworks to address the challenges of branch divergences, memory copy overhead and load imbalance when mapping irregular applications to GPGPU.

The benefit of the intermediate feedback is significant. Without this mechanism, the program must be built by pure heuristic approaches, which have limited success and are often effective for only certain specific configurations. The optimized execution setup will be lost if the system setting is changed. For instance, in the case of Embree, the hybrid packet/single-ray tracing algorithm is implemented by utilizing a specific type of BVH tree. However, determining the appropriate tracing method in the algorithm is based on heuristic. The process starts with a 16-wide packet traversal which performs 16-wide box tests. At any point in time, the bit in an active mask will be counted to indicate how many of the packet's rays are still active for a subtree. If this number falls below a given threshold, which is set to 7, the process leaves the packet traversal mode and sequentially traces all active rays in the single-ray mode[12]. The drawback of this method is that the threshold may need to change and the program will require recompilation if the system is moved to a machine with a shorter SIMD lane. Also, this method does not prevent any possible divergent execution that lowers the effectiveness of the SIMD engine. Our intermediate feedback differs

from the above method by setting the correlation based on the actual activities and data analysis. It is effective for any kind of system configuration since the runtime would automatically adjust itself. If this method were moved to another platform, the program which utilized our runtime library could adapt and gradually arrives at an optimized setting.

References

1. S. Woop, L. Feng, I. Wald, and C. Benthin, "Embree ray tracing kernels for cpus and the xeon phi architecture," in *ACM SIGGRAPH 2013 Talks*. ACM, 2013, p. 44.
2. T. J. Purcell, I. Buck, W. R. Mark, and P. Hanrahan, "Ray tracing on programmable graphics hardware," in *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '02. New York, NY, USA: ACM, 2002, pp. 703–712. [Online]. Available: http://doi.acm.org/10.1145/566570.566640
3. M. Burtscher, R. Nasre, and K. Pingali, "A quantitative study of irregular programs on gpus," in *Workload Characterization (IISWC), 2012 IEEE International Symposium on*. IEEE, 2012, pp. 141–151.
4. E. Z. Zhang, Y. Jiang, Z. Guo, K. Tian, and X. Shen, "On-the-fly elimination of dynamic irregularities for gpu computing," in *ACM SIGARCH Computer Architecture News*, vol. 39, no. 1. ACM, 2011, pp. 369–380.
5. R. Nasre, M. Burtscher, and K. Pingali, "Data-driven versus topology-driven irregular computations on gpus," in *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*. IEEE, 2013, pp. 463–474.
6. P. Monteiro and M. P. Monteiro, "A pattern language for parallelizing irregular algorithms," in *Proceedings of the 2010 Workshop on Parallel Programming Patterns*, ser. ParaPLoP '10. New York, NY, USA: ACM, 2010, pp. 13:1–13:14. [Online]. Available: http://doi.acm.org/10.1145/1953611.1953624
7. K. Pingali, D. Nguyen, M. Kulkarni, M. Burtscher, M. A. Hassaan, R. Kaleem, T.-H. Lee, A. Lenharth, R. Manevich, M. Méndez-Lojo *et al.*, "The tao of parallelism in algorithms," *ACM SIGPLAN Notices*, vol. 46, no. 6, pp. 12–25, 2011.
8. M. Kulkarni, M. Burtscher, R. Inkulu, K. Pingali, and C. Casçaval, "How much parallelism is there in irregular applications?" *SIGPLAN Not.*, vol. 44, no. 4, pp. 3–14, Feb. 2009. [Online]. Available: http://doi.acm.org/10.1145/1594835.1504181
9. T. Aila and S. Laine, "Understanding the efficiency of ray traversal on gpus," in *Proceedings of the Conference on High Performance Graphics 2009*. ACM, 2009, pp. 145–149.
10. M. Pharr and W. R. Mark, "ispc: A spmd compiler for high-performance cpu programming," in *Innovative Parallel Computing (InPar), 2012*. IEEE, 2012, pp. 1–13.
11. G. Chrysos and S. P. Engineer, "Intel® xeon phi coprocessor (codename knights corner)," 2012.
12. C. Benthin, I. Wald, S. Woop, M. Ernst, and W. R. Mark, "Combining single and packet-ray tracing for arbitrary ray distributions on the intel mic architecture," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 18, no. 9, pp. 1438–1448, 2012.