Using Integrated Processor Graphics to Accelerate Concurrent Data and Index Structures

Dissertation Summary

Joel Fuentes
University of California, Irvine
Irvine, USA
joel.fuentes@uci.edu

Isaac D. Scherson
University of California, Irvine
Irvine, USA
isaac@ics.uci.edu

ABSTRACT

With the advent of computing systems with on-die integrated processor graphics (iGPU), new programming challenges have emerged from these heterogeneous systems. We proposed different data and index structure algorithms that can benefit from the iGPU architecture and programming model. Experimental results show speedups of up to 4x on concurrent data structures and 11x on index structures, when comparing with state-of-the-art CPU implementations. Energy savings of up to 300% are also obtained when running these algorithms on iGPU.

KEYWORDS

integrated GPUs, concurrent data structures, index structures

1 INTRODUCTION

Graphics applications are used in every personal electronic device, such as laptops, tables, smartphones, etc. Most of these devices have, in a form of System on a Chip (SoC), an integrated GPU (iGPU) that allows processing graphics tasks. In recent years, the main chip manufactures have improved considerably their iGPUs, with compute capabilities that approach TeraFLOPS performance. Furthermore, iGPUs have been improved in terms of their instruction set as well as memory hierarchy and coherence. That is the case for AMD's APU and Intel's GenX architectures.

GPUs are well-known for having the potential to accelerate many kinds of computations, especially if these computations involve large number of data elements that can be executed in parallel. In this dissertation we study concurrent and index structures that can benefit from the GPU architecture to accelerate their processing. Our special focus are data and index structures that are commonly used by applications such as databases, search engines, file systems, and so on. We aim that if these data structures can be run on the iGPU instead of CPU cores, we can achieve important performance gains and energy savings.

1.1 Relevance to SC'18

Improving the performance of concurrent data and index structures through parallel processing is the main focus of this dissertation. Experimental results show that important speedup and energy savings can be obtained by using iGPUs. In addition, this research can serve as a starting point to applying similar concept to discrete GPUs and clusters.

2 INTEL INTEGRATED GPUS AND C FOR MEDIA

The Intel's on-die integrated processor graphics delivers a full complement of high-throughput floating-point and integer compute capabilities, a layered high bandwidth memory hierarchy, and deep integration with on-die CPUs. The last generations of Intel Core processors are complex SoCs integrating multiple CPU cores and iGPU (whose architecture is named GenX), all on a single shared silicon die. The communication between CPU cores, caches and the iGPU is done through a bi-directional ring interconnect that has a 32-byte wide bus. Some SoC configurations include a shared Last Level Cache (LLC) and embedded EDRAM exclusively for iGPU.

C for media (CM) is a high-level programming environment based on C/C++, which provides an efficient interface to utilize Intel's iGPU. The CM language was designed to efficiently leverage SIMD capability of the iGPU. The CM toolset consists of two primary components: 1) the compiler for the Cm language; and 2) the CM runtime, which provides an API to C/C++ programs, for setting up the GPU to transfer data and execute iGPU-targeted code. The CM project was recently open-sourced [1].

3 CONCURRENT DATA AND INDEX STRUCTURES ON IGPU

We propose a new lock-free Skiplist, which is a concurrent data structure that supports search and update operations. We also propose a general algorithm to construct and query compressed index structures based on *rank* and *select* operations. We tested our general algorithm with three well-known index structures: K^2-tree, Wavelet tree and Suffix tree.

3.1 Lock-free Skiplist

Skiplists are popular in concurrent algorithms, as they offer a probabilistic alternative to balanced search trees without costly balancing operations and still providing $O(logn)$ for search and update operations. We take advantage of SIMD16 capabilities of the Intel's GenX architecture to design a Skiplist based on chunked lists. So instead of having nodes and a list of individual updates when inserting or deleting keys, entire chunks are updated using SIMD16 atomic operations. Thus, the parallelism of this data structure is achieved at thread and instruction level (data-parallel through SIMD). A detailed proof of correctness using linearization points establishes that the proposed Skiplist guarantees the lock-freedom property for the insert and delete operations, and wait-free property for its search operation.

Experimental tests were carried out in order to measure the performance of combined concurrent operations using the proposed Skiplist. Additionally, similar operations were performed on state-of-the-art concurrent Skiplist for CPU [2, 4–6]. Figure 1 shows the number of operations per second (in millions) that are achieved by different Skiplist on CPU and our proposal on iGPU. The x axis shows the ratio of [search, insert, delete] operations, i.e. the first data item corresponds to insert-only operations, while the last data item corresponds to search-only operations. The proposed Skiplist

Figure 1: Throughput of Skiplists on CPU and iGPU

for iGPU (CMSL) achieves higher number of operations for all the scenarios, being its best performance when all the operations are search, with 4.3x speedup. When all the operations are updates, it still presents 1.5-2x speedup. We also measured the energy savings that are achieved by CMSL on iGPU, showing up to 300% of energy savings over the same experiments on CPU cores.

3.2 Index Structures

Index structures are auxiliary structures designed to speed up the search for records in different applications, e.g. databases use different types of index structures to answer quickly different types of queries. While the search in these structures is fast, constructing the index structures is a time-consuming task. We propose a general algorithm (named CMIB) to construct compressed index structures as well as algorithms to perform the actual queries on iGPU.

Most of the index structures are in a tree-like form. As such, there exist dependencies between levels that prevent parallelism through levels. So they are built level by level in top-down or bottom-up manner. CMIB considers this restriction but takes advantage of large levels that are parallelized by threads, where every thread also performs data-parallel operations through SIMD instructions.

CMIB is an iterative algorithm that can dispatch several GPU kernels depending on the input size. It begins by analyzing the input size, if it is big enough a kernel is set up and dispatched. Once the kernel concludes its execution, a CPU post-processing step gathers the resulting data. If there are pending levels to process, the algorithm starts over dispatching a new kernel until all the levels are processed.

Three well-known index structures were selected to test the CMIB algorithm: K^2-tree which represents webgraphs in compressed space, Wavelet tree which is used to index text/string and Suffix tree that is also used for text/strings. We measured and compared the performance of CMIB with sequential and multi-threaded implementations for CPU [3]. Figure 2 shows the time spent on constructing K^2-tree of sizes from 16,000 to 60 millions edges.

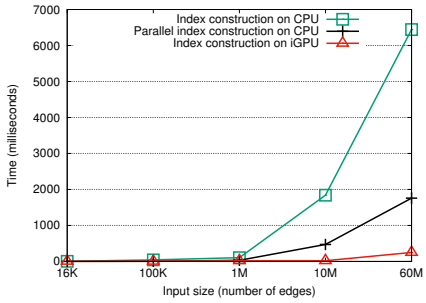

Figure 2: Index construction time on CPU and iGPU

As it can be seen in Figure 2, CMIB delivers less construction time for all the scenarios where the number of edges is greater than 100 thousand. This validates the general logic behind of CMIB, which will only dispatch iGPU kernels if the input size is big enough. Otherwise, the work can be more efficiently performed on the CPU. In general, the greater the input size is, greater is the difference of construction time between iGPU and CPU. In the current experiment, CMIB achieved 7x speedup when constructing the K^2-tree of size 60 millions edges. In terms of concurrent queries, it achieves 11x speedup over the CPU with same structure size.

4 CONCLUSIONS AND FUTURE WORK

In this dissertation, we have explored the use of iGPU for running workloads on structures typically designed for CPU. To the best of our knowledge, this is the first attempt to implement lock-free data structures and compressed index structures on iGPUs. Experimental results show that important speedup can be achieved over state-of-the-art algorithms for CPU. In addition to the performance gains, energy savings are achieved as well.

For future work, we plan to study additional data and index structures. Moreover, a characterization of CMSL and CMIB on different kinds of iGPUs would give us perspective of their performance based on hardware configurations.

ACKNOWLEDGMENTS

We are grateful to Intel Corporation and the CM Compiler team for providing us the computing resources needed for this research.

REFERENCES

[1] Intel Corporation. 2018. C-for-Media Compiler. https://github.com/intel/cm-compiler.
[2] Keir Fraser. 2004. *Practical lock-freedom.* Technical Report. University of Cambridge, Computer Laboratory.
[3] Simon Gog, Timo Beller, Alistair Moffat, and Matthias Petri. 2014. From Theory to Practice: Plug and Play with Succinct Data Structures. In *13th International Symposium on Experimental Algorithms, (SEA 2014)*. 326–337.
[4] Rachid Guerraoui and Vasileios Trigonakis. 2016. Optimistic concurrency with OPTIK. In *ACM SIGPLAN Notices*, Vol. 51. ACM, 18.
[5] William Pugh. 1990. Skip lists: a probabilistic alternative to balanced trees. *Commun. ACM* 33, 6 (1990), 668–676.
[6] Nir N Shavit, Yosef Lev, and Maurice P Herlihy. 2011. Concurrent lock-free skiplist with wait-free contains operator. US Patent 7,937,378.