LITERATURE REVIEW: GPU-based dynamic hash tables

John Shortt
School of Computer Science
Carleton University
Ottawa, Canada K1S 5B6
johnshortt@carleton.ca

October 6, 2020

1 Introduction

Parallel computing is the study of computer and network architectures to determine how to efficiently solve a problem by using large numbers of interconnected processors to each individually solve part of the problem. Technologies have evolved to allow this to become more and more feasable and the limits of single processor computing power in the light of the thermal barrier have made these architectures more and more important. Many model taxonomies for describing the types of architecture have been described in the literature. The purpose of this paper is to explore the use of a particular model and technology (GPU) to solve a specific problem (dynamic hash tables).

If we consider a model taxonomy that's based on program control and memory access (Floyd's Taxonomy) then one model that we come to is that where multiple, identical processors each have their own memory and execute a shared instruction stream. This is the SIMD model. It's particular well suited to problems where data can be partitioned and assigned to each processor and each processor can compute it's part of the solution independently of the others. This kind of problem occurs frequently in image processing problems (both 2D and 3D). Examples are image clipping, linear image transformations, rendering and shading. Because of this the graphics processor unit (GPU) has been a highly successful implementation of the SIMD model. Contrast this to the MISD model where we have multiple processors sharing a common memory with each processor executing a separate instruction stream. This model is that of the multi-core processor.

Researchers have turned to the question of leveraging GPU technology beyond graphics algorithm processing to more general purpose computing problems (GPGPU). One such problem that's received attention recently is that of constructing a large hash table that supports search, insert, update and delete operations. This is a basic capability that is useful in a broad range of application areas (such as bioinformatics, computational geometry, deep learning, visualization and data analysis). Results to date have shown that with state of the art GPU hardware rates of operations numbering in the hundreds of millions or billions per second can be achieved. These results appear to be an order of magnitude higher rates than can be achieved with a MISD architecture.

The purpose of this project is to study recent implementations of GPU-based dynamic hash tables to determine their performance characteristics, strengths and weaknesses and to explore ways to improve upon these implementations. TBD: the question of suitability of GPU-based hash tables in specific application domains.

2 Literature Review

A review of the literature finds no fewer than seven implementations of GPU-based hash tables that fall into three broad categories:

- Static hash table implementations for which there are papers available. Since the focus of this project is on dynamic hash tables this category is included for completeness but no study of these implementations is done. These are HashFight[5] and HashGraph[3].
- Hash tables that are implemented in broadly used GPU libraries. There's limited technical data available about them but they are used as baseline comparison implementations in the papers that fall into the other categories. These are cuDPP[6] and **cuDF**[7]. cuDPP is a static hash table implementation and so is not explored further.
- The category of interest is implementations of dynamic hash table for which there's significant relevant technical literature. This is **GelHash**[2], **SlabHash**[1] and **WarpCore**[4]

Table 1 summarizes some characteristics found in the referenced literature for these implementations. The performance data is unconfirmed and is likely based on different hardware.

Name	**Code**	**Insert**	**Search**	**Static**	**Features**
HashFight	github	700 M/s	2.5 B/s	Yes	platform neutral; based on data parallel primitives
HashGraph	github	2.5 B/s	2.0 B/s	Yes	Compressed Sparse Row graph data structure
cuDPP				Yes	
cuDF				No	Python DataFrame implementation
GelHash	no	800 M/s	2.5 B/s	No	group atomicity
SlabHash	github	512 M/s	937 M/s	No	slab-based memory allocation
WarpCore	github	1.6 B/s	4.3 B/s	No	batch and individual updates, multi-GPU

Table 1: Summary of GPU-based hash table implementations

The thrust of this project will be to perform an in-depth study of four dynamic hash-table implementations namely GelHash, SlabHash, WarpCore and cuDF. The remainder of this literature review is a summary of the papers that describe the implementations of dynamic GPU-based hash tables.

SlabHash
SlabHash takes a distinctive approach to the difficult problem of memory allocation that a hash table implementation must solve. It uses the slab-based allocation scheme which has been used successfully in UNIX and Linux for the allocation of kernel objects. It's known to be fast and to not cause memory fragmentation. The hash table implementation is based on a novel data structure termed the "slab list" whose structure is aligned with the underlying GPU thread, warp and memory structure in a way that minimizes the need for cooperation and shared memory between threads and warps. The authors refer to this as a "warp-cooperative" approach. No specific applications domains were cited as being best or illustrative of SlabHash performance.

GelHash
This paper claims improvements over the approaches taken in SlabHash and cites online transaction processing, OLTP, as an application area that demonstrates these improvements. As with SlabHash, an approach to memory allocation is defined and a data structure and locking strategy that benefits from the memory allocation approach is employed. To execute the supported hash table operations GelHash uses an adaptive execution model that chooses between thread-level and warp-level execution. To facilitate this it's necessary to benchmark an application's typical operation patterns to arrive at a model that optimizes between thread-level and warp-level execution. It's not clear that this will be practical for some applications. A challenge in verifying the performance of GelHash will be that there seems to be no available reference implementation.

WarpCore.
This paper is an update to the authors' earlier work on WarpDrive which was restricted to 32 bit single-value hash tables. It claims the highest performance numbers of any of the gpu-based gash table implementations reviewed (both static and dynamic) and also claims that this represents a two orders of magnitude increase over MISD based solutions. It is also scalable across multi-GPU architectures using NVIDIA's proprietary NVLink technology. An interesting aspect of this implementation is a set of host interfaces that simultaneously and efficiently allow both batch and individual updates. This means that, for example, different host programs performing different task that have different update patterns will work efficiently. The authors describe how WarpCore can be used to improve the performance of a bioinformations application, metagenomic classification.

cuDF
cuDF stands for CUDA Data Frame. It's part of the Python CUDA RAPIDs Open GPU Data Science project. A data frame is a popular Python data structure (typically from the PANDAS library) that implements a data table whose rows and columns can be accessed directly by their key. Hash tables are the way this is implemented. This project is well-supported and is likely one of the first places that data scientists consult when looking for GPU libraries to support their projects. We intend to evaluate the performance of cuDF so that it can be consistently and accurately compared to the 3 other dynamic hash table implementations.

References

[1] S. Ashkiani, M. Farach-Colton, and J. D. Owens. A dynamic hash table for the gpu. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 419–429, 2018.

[2] L. Gao, Y. Xu, C. Xu, R. Wang, H. Yang, Z. Luan, and D. Qian. Towards a general and efficient linked-list hash table on gpus. In *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 1452–1460, 2019.

[3] Oded Green. HashGraph – Scalable Hash Tables Using A Sparse Graph Data Structure. *arXiv e-prints*, page arXiv:1907.02900, July 2019.

[4] Daniel Jünger, Robin Kobus, André Müller, Christian Hundt, Kai Xu, Weiguo Liu, and Bertil Schmidt. WarpCore: A Library for fast Hash Tables on GPUs. *arXiv e-prints*, page arXiv:2009.07914, September 2020.

[5] Brenton Lessley, Shaomeng Li, and Hank Childs. Hashfight: A platform-portable hash table for multi-core and many-core architectures. *Electronic Imaging*, 2020(1):376–1–376–13, 2020.

[6] CUDPP 2.0 CUDA Data-Parallel Primitives Library. *Overview of CUDPP hash tables*, 2010. Available at http://cudpp.github.io/cudpp/2.0/hash_overview.html.

[7] RAPIDS Open GPU Data Science. *Welcome to cuDF's documentation!*, 2020. Available at https://docs.rapids.ai/api/cudf/stable/.