Towards Highly-Available OLTP on GPUs - A Survey on upcoming Developments

Aditya
aditya@st.ovgu.de
Otto von Guericke University, Magdeburg, Germany

Gabriel Campero Durand
campero@ovgu.de
Otto von Guericke University, Magdeburg, Germany

ABSTRACT

The success of GPGPU (General-Purpose computation on Graphics Processors) in recent times has motivated various research areas to include GPU as a platform for their research technologies. Massive Parallelism, Hardware Support and SIMD (Single Instruction Multiple Data) executions provided by GPUs are major factors driving and shaping this trend. In this research paper, we survey developments for supporting Transaction Processing with GPUs.

OLTP (On-Line Transaction Processing) plays a significant role for everyday businesses. Due to the ever-increasing rate of users using online services and systems for their daily or business needs, the number of transactions in such online systems is also increasing and with this increases more demand for high throughput systems. In OLTP when it comes down to processing a transaction, various strategies are developed to avoid multi-user problems while handling, at the same time, as many transactions as possible to achieve high throughput by using the available system resources to their best possible extent.

We study and discuss in this research paper which major approaches are dealing with the OLTP and combining it with the GPU-technology for making the most of what the modern hardware has to offer. We also consider recent advances in Transaction Processing that might be relevant for improving OLTP on GPUs.

CCS CONCEPTS

• **Information systems → Data management systems**.

KEYWORDS

GPGPU, OLTP, Transaction Processing on GPU

1 INTRODUCTION

With the increase in the amount of internet services (like online banking services), the rate of growth for online Transaction Processing (OLTP) is also increasing. OLTP is to be understood as Transaction Processing among many multiple users accessing the transactional-system at the same time, not just for reading data but also for the vast amount of write operations. The major challenges such read and write operations among many concurrent running transactions lead to is the need to maintain consistency and system integrity, as a concurrency example is represented in Figure 1 .

Schedule a		Schedule b		Schedule c		Schedule d	
TNX_1	TNX_2	TNX_1	TNX_2	TNX_1	TNX_2	TNX_1	TNX_2
lock A		lock A		lock A		lock A	
unlock A			lock B		lock B	unlock A	
lock B		unlock A		unlock A			lock B
unlock B			unlock B	lock B			unlock B
	lock B	lock B			unlock B	lock B	
	unlock B		lock C		lock C	unlock B	
	lock C	unlock B		unlock B			lock A
	unlock C		unlock C		unlock C		unlock A

Figure 1: Multiple schedules applying concurrency among transactions [1]

This explains the existence of the vast number of strategies and techniques in Transaction Processing for avoiding multi-user problems, such as dirty read problem or lost update issue. In terms of hardware-platform usage, the typical platform for OLTP workloads is CPUs — the reason being that the OLTP workloads require huge amount of synchronization among many concurrent transactions within a transaction-schedule. For such massive synchronization CPUs seem to be a well fit for OLTP-Workloads. Unfortunately, this leads to the underutilization of heterogeneous processors available in modern times, like GPUs. With the emergence of General-Purpose-Computing on GPUs (GPGPU) heterogeneous processors seem to be well integrated for managing the various types of processing tasks. Limitations like Power-Wall and scalability thresholds of multi-core processors support the research and utilization of co-processors like GPUs further. Since GPUs are not the same as CPUs they have different strengths and weaknesses. For example, branching and control operations are a core feature of CPU architecture. Nevertheless, for the same type of branching and control tasks GPUs must use workaround-techniques which come with the cost of an overhead. On the other hand, massive data parallelism is the biggest strength of GPUs. Nevertheless, despite several weaknesses of GPUs, the types of workloads being optimized to utilize GPUs are increasing.

In our research paper, we follow a double agenda scheme by going through (1) state-of-the-art approaches for Transaction Processing on GPUs, and (2) state-of-the-art approaches for Transaction Processing on distributed systems, in general (i.e. Transaction-Workloads distributed among CPU and GPU). Through this double agenda approach, we cover the latest developments in the corresponding fields and we open a discussion about defining possible paths between developments in Transaction Processing and in GPUs. We will go through approaches which have already been tried on GPUs for OLTP, and observe if there is any big idea arising which could invigorate the research area. In the following we present our main contributions:

- We review the work that followed a seminal paper in the field (GPUTx) [2], i.e. reviewing the history of OLTP on GPUs after the emergence of GPUTx [2].
- We review the state-of-the-art in Transaction Processing with the focus on the distributed systems (i.e. Transaction-Workloads distributed among CPU and GPU) to filter out techniques, which could be applied for Transaction Processing on GPUs.
- We raise a hypothetical discussion on how the state-of-the-art in Transaction Processing can be extended for incorporating GPUs, hence filtering further possible paths of interests between GPUs and OLTP.

Coming next, we describe the precise problem statement with which paper is engaged.

2 PROBLEM STATEMENT

GPUs (Graphical processing units) are an optimal hardware platform for analytical processing, but unfortunately, they are underutilized in the case of Transaction Processing, the reason being that the massive amount of thread-parallelism on GPUs is optimal for the sake of parallelism itself, but that brings along with it the disadvantage that synchronization of concurrently running transactions among the threads becomes a challenge. The inability to use GPUs for Transaction Processing is of high concern, especially in systems which include a massive amount of OLTP, or might suddenly receive a high OLTP workload. To be able to use co-processors like GPUs, it makes sense that we find further strategies to involve GPUs for OLTP. Past research has already looked into this and there has been still no satisfactory solution found. Furthermore, the research to utilize GPUs for OLTP is still not clear about what developments follow next, hence we conclude that the ongoing research on OLTP-GPU deserves analysis to observe which novel research directions might be arising. There might be developments in the field which might not have been considered yet for the future developments on OLTP-GPU, hence we look into them and open discussions about, if these developments should be considered for future works.

After explaining the problem statement, on which our survey paper focuses — in the next section we mention necessary background information for the readers.

Processing Device	Parallelization Properties		Memory Scaling	
	Pipeline Parallelism	Data Parallelism	Memory Capacity	Memory Bandwidth
CPU	+/++	+	++	+
APU	+/++	+	++	+
MIC	+/++	+/++	+	++
GPU	-	+/++	+	++
FPGA	++	-/+	-/+	++

Legend: ++ = Excellent, + = Good, - = Poor

Figure 2: Hardware capabilities comparison overview [3]

3 BACKGROUND

In the Background section, first, we go through the necessary knowledge about data management in the context of databases and define important concepts about the GPUs, which we consider the reader to know, in order to fully understand our survey paper.

Afterward, we define the basics of transactions and Transaction Processing. For this purpose, we briefly define transaction theory and concurrency strategies in a typical Transaction Management system.

3.1 Data Management and GPUs

We begin with the context of Data Management and GPUs. The well-known problem of Power-Wall has led to the emergence of general purpose computing on co-processors. The Power-Wall issue refers to transistor switching rate limitations on singe-core CPUs. Furthermore, limited scalability-performance ratio of multi-core CPUs extends the Power-Wall issue, which leads to the development and popularization of co-processors like (1) GPUs (Graphical Processing Units), (2) MICs (Many Integrated Cores), (3) APUs (Accelerated Processing Units), and (4) FPGAs (Field-Programmable Gate Arrays) as processing accelerators. Keeping other types of aforementioned co-processors aside, we are focusing in our paper on GPUs. The main reason being the hardware capabilities, which GPUs have to offer. Figure 2 depicts a comparative overview of GPUs with other co-processor types available on the market. As can be derived from Figure 2, the combination of data parallelism, memory scaling, and low hardware costs which GPUs offer, make them of greater consideration on the co-processor market in comparison to the other co-processor types. The GPUs differ from the CPUs also in terms other than those presented in Figure 2. The main difference between GPUs and CPUs, on a functional level, lies within their architecture. The CPUs are designed for control-intensive operations, whereas the GPUs are more suitable for compute-intensive operations. The reason for this difference lies behind the limited capabilities of GPUs for branching operations. Branching operations are not optimal on GPUs because (1) concurrent writes might corrupt data, and (2) locking might serialize threads reducing performance. For such branching limitations of GPUs, there are workarounds available, for example, the concept of parallel-selections. Nevertheless, such workarounds still do not extend the capabilities of GPUs for branching operations to the extent to which CPUs manage originally. Furthermore, the execution of instructions starts on the GPU when data is available in GPU memory. For this, the GPUs work in combination with the CPUs as in a host-worker model, where the CPUs handle the I/O and perform other operations, while the GPU executes in parallel to the CPU. GPUs work with the concept of warps, where a warp is defined as a group of threads, which are lightweight and run the same instruction codes in parallel. Because of these warps being lightweight, the thread handling turns out to be not about reducing latency but instead hiding it — this becomes possible because of data-parallel execution and scheduling of batches on these warps. Similar to SIMD (Single Instruction Multiple Thread), the warps on the GPU follow the SIMT (Single Instruction Multiple Thread) fashion. Whereby SIMD can be defined as vectorization for leveraging

processing capabilities — it is also known as vectorized execution or vector unit. SIMD results in executing the same instruction on the whole set of data. We hope that through the aforementioned background information, the readers now are well equiped with the fundamentals for Data Management and GPUs and can better understand the following sections.

3.2 Transfer times CPUs and GPUs

After going through the background in context of Data Management and GPUs, this subsection deals with the context of the Transfer times between CPUs and GPUs. CPUs and GPUs work as in a worker-host processing model. Whereby the CPU handles the Input-Output part of the scenario and GPU starts with the execution of instructions as soon as the data provided by CPU reaches the GPU-Memory. The Connection between CPUs and GPUs exists using interconnects which have less data bandwidth as compared to the bandwidth in GPU-memory. Thus, such interconnects are considered to be the major bottleneck of the distributed transaction workloads between CPUs and GPUs. This issue contributes to one of the major researches in the research field of Distributed Transaction Processing.

3.3 Transaction Processing

After describing the transfer time issue between CPUs and GPUs, in this subsection we explain Transaction Processing in general — this is the second most important building block for the better understanding of our paper for readers. A transaction is formally defined as a sequence of operations which brings a database from one consistent state to another consistent state, applying ACID (Atomicity, Consistency, Isolation, and Durability) if possible. A transaction can be modeled to follow the read-write model, whereby according to the read-write model, a Transaction T is a finite sequence of operations p_i of the form $r(x_k)$ or $w(x_k)$: $T = p_1 p_2 p_3 \ldots p_n$ with p_i belongs to $r(x_k), w(x_k)$. A complete transaction T has as the last step either an abort a or a commit c : $T = p_1 \ldots p_n a$ or $T = p1 \ldots p_n c$.

Transaction Management in a database system is of priority because without such management the database system would lose its consistency and hence would lead to multi-user problems, such as non-repeatable read, dirty read, phantom problem, lost update, multi-user anomaly, or cursor reference problem. A vital part of Transaction Management in databases consists of the concept of serializability. According to which the transaction scheduler has the responsibility to decide and create serialized or serializable schedules of read-write operations. Whereas a serializable schedule is defined as an interleaved series of read/write operations between two or more transactions, whose end-result is the same as the result of serialized execution of those same transactions executed in the serialized style. Full serializability is not always needed and in fact database systems can be configured to run on different isolation levels, each enabling a different series of inconsistencies to potentially occur, and helping the system run faster. An interleaved execution is of great interest because it allows the usage of the system resources to its highest potential. In this context arises the term concurrency, which is to be distinguished from being parallel, because an interleaved execution of transactions within a schedule is not to be confused with the parallel execution, but with the concurrent one. In Figure3 the major components are described, which are involved in Transaction Management. Describing Transaction Processing further it must be mentioned that it uses various types of strategies for maintaining the transaction integrity, namely (1) semantic integrity and (2) runtime integrity. The variant strategies include strategies to verify view serializability, conflict serializability, and Transaction Processing schemes such as multi-version concurrency control (MVCC). Because of space constraints, we are only describing the MVCC in the following.

MVCC maintains different versions of data available for various transactions, whereby the implementation can be done using a ring buffer and the appropriate version of the data stays in the system until there are no further transactions alive that use the previous version.

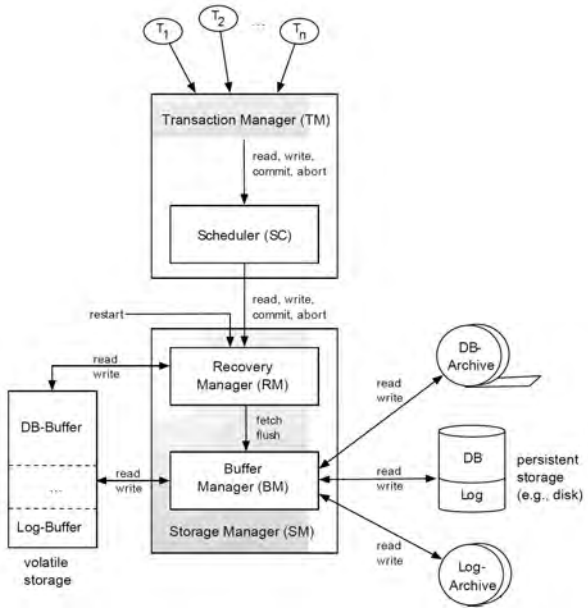

Figure 3: Major components included in Transaction Management [1]

After describing the transaction theory and typical concurrency strategies in Transaction Processing, it must be mentioned that there exist many research gaps in the area of Transaction Processing, for example, the implementation and optimization of prementioned MVCC on GPUs in a distributed environment (i.e. Transaction-Workloads distributed among CPU and GPU), where the distribution strategies work along with the concurrency techniques to manage load balancing.

Coming next, we discuss the methodology to which our survey's literature review is oriented.

4 LITERATURE REVIEW METHODOLOGY

In this section, we engage with the literature review methodology behind our survey paper. We begin this section with the clear definition of our research questions — because we believe that our

research methodology is concretely derived from our goals, which can be formulated as our core research questions.

4.1 Research Questions

This survey paper deals with the following research questions:

(1) What has followed the research on GPUTx[2] , i.e. what developments have been there in OLTP on GPUs after the emergence of GPUTx?
(2) What is the state-of-the-art in Transaction Processing with the focus on the distributed systems (i.e. Transaction-Workloads distributed among CPU and GPU)?
(3) How can the state-of-the-art in Transaction Processing be extended for incorporating GPUs, hence what further possible paths of interests between GPUs and OLTP can be established?

4.2 Methodology

After describing our research goals, in this subsection, we present a short description of the procedure of gathering research papers for our survey.

Since it is well known that GPUTx [2] is a good representative of OLTP on GPUs, we have considered the GPUTx-paper as our primary paper. Afterward, we have considered all papers that cite GPUTx, for the review process. In total, we have collected 103 papers out of which we focused only on those which propose measures related to Transaction Processing. In the end, we have selected 32 papers, which we summarize in the literature review section. Hence, as mentioned before, we crawled a series of papers which cite GPUTx. During the process, we skipped papers which are oriented to irrelevant topics, such as papers written in the research field of energy efficiency or the use of GPUs for processing medical images. We picked papers that were related to scheduling, locking, and synchronization as our core papers, as we consider them to be the most relevant for our survey. We also considered some papers related to Joins, OLAP, Concurrency, Survey, Graphs, and Key-Value-Stores. The research papers topics we explicitly excluded for being surveyed are topics on SQL, and query optimization as these topics do not meet the goal of our survey.

After defining the research methodology in this subsection, we move forward to the next section, which engages itself with the schenario of OLTP on GPUs.

5 OLTP ON GPUS

This section is divided into two parts, namely (1) GPUTx (2) Building on GPUTx. In the first subsection ***GPUTx*** we define the core approach for OLTP on GPUs, in the second subsection ***Building on GPUTx***, we describe which recent developments have arisen above the main core approach.

5.1 GPUTx

This subsection begins with the description of the major approach for OLPT on GPUs, i.e GPUTx. Bingsheng He has brought at the international conference on Very Large Databases (VLDB) the bulk execution model known as GPUTx [2]. It is based on a strategy to execute transactions on GPUs whose prototypic implementation is done in CUDA C. The core idea behind GPUTx is to bundle a set of transactions and execute them as one single kernel on the GPU. One of its limitations is, that it needs pre-defined transaction types as stored procedures. GPUTx is built on the following features (1) Transaction ordering by the timestamp of transactions at the time of their submission (2) Transaction pools and bulk transaction generators (3) T-Dependency graph for data dependency modeling and defining correctness.

The T-Dependency graph is defined as a data structure made of directed acyclic graph (DAG) as Figure 4 illustrates, which leads to the advantage that no deadlocks may arise. The T-Dependency graph has two formal properties, namely (1) if the transactions belong to the same k-set (set of vertices with depth k) then they must be conflict-free, and their execution in parallel must preserve correctness.

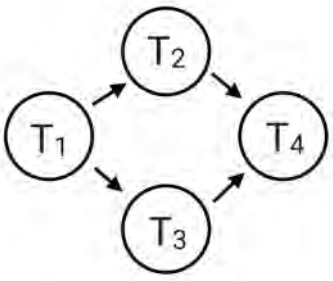

Figure 4: T-Dependency graph as datastructure for Transaction Management as proposed by Bingsheng He et al. [2]

(2) if there are transactions not already committed and processed in lower k-sets, then all transactions present in the whole k-set can not be executed as bulk-processing.

Unfortunately, the T-Dependency graph construction is bound with high complexity, hence in case the transactions become more advanced in terms of their read-write operations then T-Dependency graph would become the bottleneck of the GPUTx core strategy of bulk-processing. As a workaround for this, Bingsheng He et al. also proposed a data-oriented algorithm.

After completing the subsection on GPUTx, we move next to the approaches which have been developed in advancement of GPUTx.

5.2 Building on GPUTx

This subsection engages itself with the advancements of GPUTx, many research papers have cited GPUTx and our study of them can be concluded as the following. The work based on GPUTx is very broad, it includes papers which look at joins, OLAP, Concurrency, Graphs, Key-Value-Stores, and Query Optimization. However, we focused on a subset of papers which look at Scheduling, albeit it is with OLAP and Locking. We divide the surveyed research papers into different categories, depending on how relevant their content is to the aim of our survey.

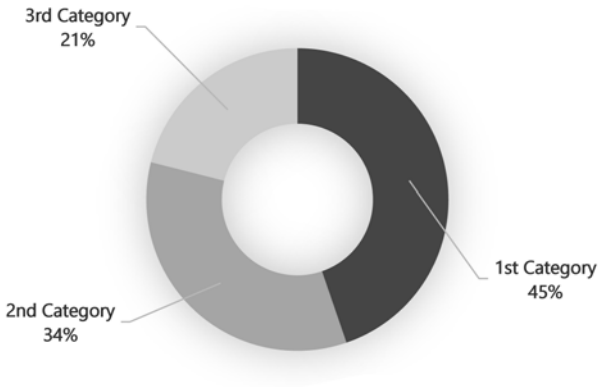

Figure 5: Literature Review-Methodology

For example, works we considered to be in one category include works on surveys aiming at GPU optimization for OLTP [4][5], a survey focused on memory [6], storage layouts [7] [8], bit-oriented decomposed storage for GPUs[9], OLAP in terms of scheduling or concurrency[10][11], scheduling for OLTP[12], concurrency in general [13][14], locking in regards to comparing concurrency models under pessimistic Transaction Processing [15][16][17][18][19], and an optimistic variant of it, albeit it is for CPUs [20].

In another category we consider works related to Joins [21], OLAP[22][23][24][25][26][27][28], Concurrency [29][30], Surveys which are not completely relevant for us but still of interest[31][32] and future work[33].

Under the last category we consider papers related to Graphs[34][35][36], Key-Value-Storage [37], Query Optimization and Index Structures for GPUs[38], XML document storage[39] and future directions[40].

The Literature Review-Methodology categorization is concluded in Figure 5. The developments after the emergence of GPUTx lead us further to take a deep look into the state-of-the-art of Transaction Processing also on CPUs, Databases in general and then we are trying to compare it with Transaction Processing on GPUs and hence connect the relations between the mentioned areas, even when it is very hypothetical. We focus on these hypothetical connections because we advocate that it is not that OLTP on GPUs is not a well-suited task, but it might be the case that we do not yet have the breakthrough approach one needs for it to work ideally. Keeping this goal for our survey paper, we plan to discuss, if there is any breakthrough that helps the OLTP-GPU scenario to develop further.

Pinnecke et al. [4] surveyed in the year 2017 to find out if modern systems are good enough for HTAP management in cooperation with general- and special-purpose processors. Their survey ends with the conclusion that modern systems are not yet ready for such types of workloads. for this, they compared state-of-the-art engines supporting HTAP workload on CPUs and GPUs.

Jason V Ma from Ryerson University [5] reviews various approaches to accelerate the transaction queries on GPUs. The approaches Ma summarizes are (1) creating GPU accelerated primitives, (2) using an opcode execution model, and (3) using primitives with GPU kernels for reducing memory-usage and increasing compiler-optimization.

Arefyeva et al. [6] survey major techniques for GPU-Memory-Management in the context of cross-device query processing. The techniques through which their survey went were (1) Divide-and-conquer approach, (2) pinned CPU memory usage, (3) Unified Virtual Addressing, and (4) Unified Memory. In Figure 6 the comparison results of these four major approaches are described.

After covering the details of OLTP on GPUs in this section, in the next section we look forward to the details in context of the Distributed Transaction Processing.

	Divide-and-conquer	Mapped memory	Unified Virtual Addressing	Unified Memory
Data location	Main memory	Main memory	Both	Migrating
Transfers and executions are overlapped	Yes	Yes	Yes	No
Requires explicit allocations and transfers	Yes	No	No	No
Requires synchronization	Yes	No	No	No
Requires coherence maintenance	No	Yes	Yes	No
Avoids unnecessary data transfers	No	Yes	Yes	Yes
Unified address space	No	No	Yes	Yes
Speed of memory accesses	Fast	Slow	Depends on data location	Fast
Allows for memory oversubscription	No	Yes	No	From CUDA 8.0

Figure 6: Comparison of the approaches for cross-device query processing as presented by Arefyeva et al. [6]

6 DISTRIBUTED TRANSACTION PROCESSING

In this section, we present a small review on Transaction Processing in distributed environments (i.e. Transaction-Workloads distributed among CPU and GPU), to filter out what is compelling in the developments.

In terms of OLTP and concurrency Kaibo Wang et al. have engaged with the underutilization GPU-resource issue, and since sharing GPUs for concurrent transactions is not heavily supported, they observed the underutilization to be 25%. Sharing of transactions causes Resource contention and hence they consider the most vital part to be the software support for contention's administration. They propose techniques for query scheduling and swapping pages in memory as their contribution regarding the problem.

Regarding scheduling for OLTP, one of the core research papers which our survey has gone through [12] evaluates and proposes a series of approaches for analysis mostly with orientation for CPUs, like Data-Oriented Architecture and delegation. It could be seen as one of the potential future works for doing the same for OLTP on GPUs.

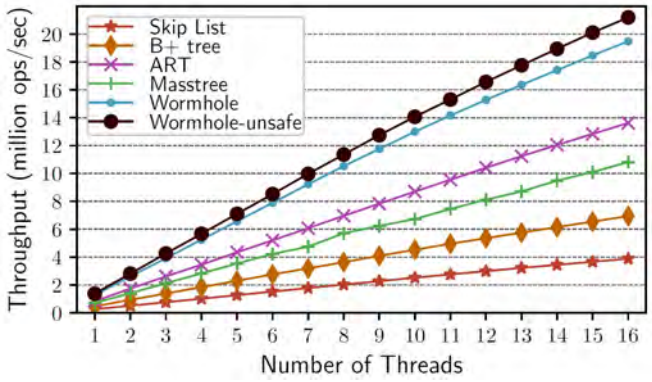

Figure 7: Lookup throughput with different number of threads — Xingbo Wu et al. proposed Wormhole-datastructure, which uses a transaction locking model internally [15]

The aspect of locking in OLTP is of two types, pessimistic and optimistic. Under pessimistic locking, we have gone through the work of Xingbo Wu et al.[15]. They originally introduce a data structure with the name of Wormhole, which they claim to be leveraging the strengths of three data structures altogether, namely (1) hash tables, (2) prefix tree, and (3) B+ tree. The combination of mentioned data structures does not come at the expense of extra space requirements for index-maintenance, which makes Wormhole a quite attractive approach. In the context of locking, Wormhole divides the database operations into three groups, namely (1) point and range lookups, (2) insertions and deletions — where modifications are contained in terms of the number of leafs on which the modification occurs, and (3) insertions and deletions — which causes split and merge of leaf nodes. Through the division of database operations into three different groups, they try to minimize the locking impact on transactions running concurrently. The reduction of impact occurs through minimizing the number of locks applied in total. Hence the aim, which the proposed solution follows is to minimize the locking overhead for insertions and deletions, especially for lookup operations. Figure 7 depicts their experiment results. Furthermore, along with their locking strategy they were able to deliver the worst-case time of O(log L) where L represents the length of the key being searched for. This runtime can be compared with the runtime of B+ trees, which is O(log N), where N stands for the number of keys in an index.

In the context of OLTP and transactional synchronization, Yunlong Xu et al. [18] proposed in the year 2016 another locking model for GPUs. The proposed locking model is based on the following developments (1) stealing locks within individual warps, where warps are to be defined as a group of threads on a GPU, and (2) lock virtualization. Their proposed scheme claims to be delivering 1.22x speedup in combination with 94% memory cost reduction. Figure 8 provides further comparison attributes between their locking scheme and related works. The motivation for their locking-scheme proposal is that the GPU locking, when implemented during concurrency, leads to (1) concurrency bugs, or (2) Single Instruction Multiple Threads (SIMT) model of GPUs causing low hardware utilization.

7 DISCUSSION

In the last section we covered the details about Transaction processing in a distributed environment, in this section we move on the discussions about the developments. We begin with the discussion subsection on GPUTx, afterwards we go through the subsection for the discussion on the Transaction Processing.

7.1 GPUTx

As part of our discussion on GPUTx, we take a broad look at what advances and limitations are to be observed after GPUTx [2]. GPUTx has the following limitations (1) Support for pre-defined stored procedures only, (2) T-Dependency graph construction, (3) Sequential workload, (4) Database fitting into the GPU memory, (5) GPUs without atomic operation support, and (6) Portability to other many-core architectures.

Because of the limited hold for pre-defined stored methods, users of the GPUTx engine are not able to generate transactions through commands, this concludes that GPUtx is only ideal for implementations including transactions, which need to be concluding the static-oriented methods and procedures.

The T-Dependency graph construction limitation defines the scope for the GPUTx functional competence. As the construction of the T-Dependency graph is necessary for GPUTx bulk execution strategy, the construction overhead remains fine but becomes the bottleneck of the whole system if the transaction-set becomes complicated — leading to more resourceful construction of the T-Dependency graph. The authors of GPUTx suggest the there might be better algorithms designed for the aforementioned construction difficulties.

Furthermore, the limitation related to the portability is based on the fact, that the GPUTx implementation is designed and based on CUDA. whereby other many-core architectures such as AMD GPUs and Fusion remain untouched.

GPUTx brings along many advances in the field, two of them are as follows (1) Bulk Execution Model for OLTP transactions, (2) T-Dependency graph as data structure under system design. After the discussion-subsection on GPUTx, we move next to the context of the discussion on the Transaction Processing.

7.2 Transaction Processing

The discussion on Transaction Processing in a distributed environment, (i.e. Transaction-Workloads distributed among CPU and GPU) follows what is happening currently in the field.

Holger Pirk [9] gives attention in his paper to the problem of not using the strengths of CPU and GPU altogether in a system for query processing. As a load distribution strategy, Pirk introduces the idea of decomposing relational data into individual bits and using these bits as partitions to be processed by CPU or GPU. Pirk claims it to be an ideal distribution strategy for real-life use cases of query processing. In Figure 9 the proposed strategy is compared with the typical isolation processing approach.

Approach	Application scope	Hardware modification	Performance overhead	Complexity
Atomic operations [3]	small	no	medium	low
Spinlocks, try-locks [10][17]	small	no	medium	medium
Spinlocks with serialization within Warp [18]	large	no	high	medium
Hardware-supported locks [26]	large	yes	low	medium
Hardware transactional memory [10][11][22]	large	yes	low	low
Software transactional memory [8][13][24]	large	no	high	low
Algorithmic optimizations [12][15][19]	single program	no	program relevant	high
Proposed locking scheme	**large**	**no**	**medium**	**medium**

Figure 8: Summary of synchronization approaches for thread contention on GPUs along with the proposed locking-scheme from Yunlong Xu et al. [18]

According to the author, the handling of the query-streams in continuous flow is implemented in a way that the evaluation phases of multiple queries can be — for the purpose of keeping all devices busy — can be interleaved. The two such evaluation phases are (1) GPU preselection and (2) CPU refinement. Whereby the GPU preselection phase concludes with the prefiltration of the dataset, in the second phase of CPU refinement, the CPU duplicates the incomplete results from the device memory of the GPU and hence does a join with the main memory list of remaining components of the query. Further information about this approach and the other related approaches can be derived from the citations provided in our survey research paper. Now as we are at the end of the discussion section, we move next to the conclusion, which we emphasize bounded in terms of the potential future work.

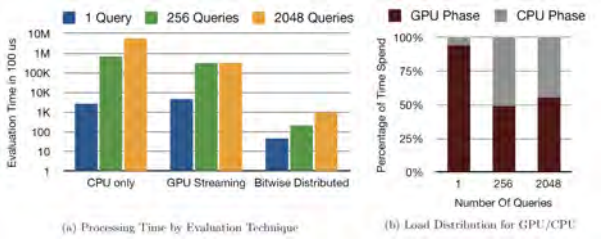

Figure 9: Query-Evaluation-Performance comparison along with proposed Bitwise Distribution strategy from Holger Pirk [9]

8 CONCLUSION AND FUTURE WORK

In this section, we summarize what we have found through our survey and subsequently suggest future directions of the work in the context of distributed Transaction Processing with GPUs. As we have so far observed, OLTP on GPUs require extensions and the performance it reaches is still not good as OLTP systems on CPUs.

The paper from Iya Arefyeva et al. [8] raises the question of OLTP on GPUs might be a dream. They hold the following behind this question, (1) if transactions arrive on GPUs for processing in small batches, then GPUs can not be used to their full extent, and (2) if transaction operations are processed on GPUs in bulk, the bulk size might be too big for being collected and sent to GPUs in time, hence providing reply to the users at the wished intervals might become a challenge.

9 ACKNOWLEDGMENTS

With that sharp warning Athena seated headlong Ares on his throne. But Queen Hera summoned Apollo from the halls and Iris too, the messenger of the immortals, and gave them both their winged marching orders: "Zeus directs you to Ida with all good speed! But once you arrive and meet great Zeus's glance, do whatever the Father drives you on to do." — Muses, who hold the halls of Zeus

REFERENCES

[1] T. Leich and G. Saake, "Lecture slides in transaction processing," November 2020, Otto-von-Guericke University of Magdeburg.
[2] B. He and J. X. Yu, "High-throughput transaction executions on graphics processors," *arXiv preprint arXiv:1103.3105*, 2011.
[3] B. Gurumurthy, "Lecture slides in advanced topics in databases," May 2019, Otto-von-Guericke University of Magdeburg.
[4] M. Pinnecke, D. Broneske, G. C. Durand, and G. Saake, "Are databases fit for hybrid workloads on gpus? a storage engine's perspective," in *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pp. 1599–1606, IEEE, 2017.
[5] J. V. Ma, "Accelerating rdbms operations using gpus," 2013.
[6] I. Arefyeva, D. Broneske, G. Campero, M. Pinnecke, and G. Saake, "Memory management strategies in cpu/gpu database systems: a survey," in *International Conference: Beyond Databases, Architectures and Structures*, pp. 128–142, Springer, 2018.
[7] I. Arefyeva, D. Broneske, M. Pinnecke, M. Bhatnagar, and G. Saake, "Column vs. row stores for data manipulation in hardware oblivious cpu/gpu database systems.," in *Grundlagen von Datenbanken*, pp. 24–29, 2017.
[8] I. Arefyeva, G. C. Durand, M. Pinnecke, D. Broneske, and G. Saake, "Low-latency transaction execution on graphics processors: Dream or reality?," in *ADMS@ VLDB*, pp. 16–21, 2018.
[9] H. Pirk, "Efficient cross-device query processing," in *The VLDB PhD Workshop. VLDB Endowment*, 2012.
[10] K. Wang, K. Zhang, Y. Yuan, S. Ma, R. Lee, X. Ding, and X. Zhang, "Concurrent analytical query processing with gpus," *Proceedings of the VLDB Endowment*, vol. 7, no. 11, pp. 1011–1022, 2014.
[11] K. Zhang, F. Chen, X. Ding, Y. Huai, R. Lee, T. Luo, K. Wang, Y. Yuan, and X. Zhang, "Hetero-db: next generation high-performance database systems by best utilizing heterogeneous computing and storage resources," *Journal of Computer Science and Technology*, vol. 30, no. 4, pp. 657–678, 2015.
[12] R. Appuswamy, A. C. Anadiotis, D. Porobic, M. K. Iman, and A. Ailamaki, "Analyzing the impact of system architecture on the scalability of oltp engines for high-contention workloads," *Proceedings of the VLDB Endowment*, vol. 11, no. 2, pp. 121–134, 2017.
[13] L. Gao, Y. Xu, R. Wang, H. Yang, Z. Luan, and D. Qian, "Accelerating in-memory transaction processing using general purpose graphics processing units," *Future Generation Computer Systems*, vol. 97, pp. 836–848, 2019.
[14] L. Gao, Y. Xu, R. Wang, Z. Luan, Z. Yu, and D. Qian, "Thread-level locking for simt architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 5, pp. 1121–1136, 2019.

[15] X. Wu, F. Ni, and S. Jiang, "Wormhole: A fast ordered index for in-memory data management," in *Proceedings of the Fourteenth EuroSys Conference 2019*, pp. 1–16, 2019.

[16] M. Moazeni and M. Sarrafzadeh, "Lock-free hash table on graphics processors," in *2012 Symposium on Application Accelerators in High Performance Computing*, pp. 133–136, IEEE, 2012.

[17] Y. Xu, R. Wang, N. Goswami, T. Li, L. Gao, and D. Qian, "Software transactional memory for gpu architectures," in *Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization*, pp. 1–10, 2014.

[18] Y. Xu, L. Gao, R. Wang, Z. Luan, W. Wu, and D. Qian, "Lock-based synchronization for gpu architectures," in *Proceedings of the ACM International Conference on Computing Frontiers*, pp. 205–213, 2016.

[19] A. Yilmazer and D. Kaeli, "Hql: A scalable synchronization mechanism for gpus," in *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, pp. 475–486, IEEE, 2013.

[20] M. Smolinski, "Coordination of parallel tasks in access to resource groups by adaptive conflictless scheduling," in *Beyond Databases, Architectures and Structures. Advanced Technologies for Data Mining and Knowledge Discovery*, pp. 272–282, Springer, 2015.

[21] J. He, M. Lu, and B. He, "Revisiting co-processing for hash joins on the coupled cpu-gpu architecture," *arXiv preprint arXiv:1307.1955*, 2013.

[22] Y. Yuan, R. Lee, and X. Zhang, "The yin and yang of processing data warehousing queries on gpu devices," *Proceedings of the VLDB Endowment*, vol. 6, no. 10, pp. 817–828, 2013.

[23] J. He, S. Zhang, and B. He, "In-cache query co-processing on coupled cpu-gpu architectures," *Proceedings of the VLDB Endowment*, vol. 8, no. 4, pp. 329–340, 2014.

[24] S. Breß and G. Saake, "Why it is time for a hype: A hybrid query processing engine for efficient gpu coprocessing in dbms," *Proceedings of the VLDB Endowment*, vol. 6, no. 12, pp. 1398–1403, 2013.

[25] S. Breß, F. Beier, H. Rauhe, K.-U. Sattler, E. Schallehn, and G. Saake, "Efficient co-processor utilization in database query processing," *Information Systems*, vol. 38, no. 8, pp. 1084–1096, 2013.

[26] J. Li, H.-W. Tseng, C. Lin, Y. Papakonstantinou, and S. Swanson, "Hippogriffdb: Balancing i/o and gpu bandwidth in big data analytics," *Proceedings of the VLDB Endowment*, vol. 9, no. 14, pp. 1647–1658, 2016.

[27] H. Rauhe, J. Dees, K.-U. Sattler, and F. Faerber, "Multi-level parallel query execution framework for cpu and gpu," in *East European Conference on Advances in Databases and Information Systems*, pp. 330–343, Springer, 2013.

[28] S. Breß, I. Geist, E. Schallehn, M. Mory, and G. Saake, "A framework for cost based optimization of hybrid cpu/gpu query plans in database systems," *Control and Cybernetics*, vol. 41, 2012.

[29] J. Alglave, M. Batty, A. F. Donaldson, G. Gopalakrishnan, J. Ketema, D. Poetzl, T. Sorensen, and J. Wickerson, "Gpu concurrency: Weak behaviours and programming assumptions," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 1, pp. 577–591, 2015.

[30] T. Sorensen and A. F. Donaldson, "Exposing errors related to weak memory in gpu applications," *ACM SIGPLAN Notices*, vol. 51, no. 6, pp. 100–113, 2016.

[31] S. Breß, M. Heimel, N. Siegmund, L. Bellatreche, and G. Saake, "Gpu-accelerated database systems: Survey and open challenges," in *Transactions on Large-Scale Data-and Knowledge-Centered Systems XV*, pp. 1–35, Springer, 2014.

[32] A. Raza, P. Chrysogelos, A. C. Anadiotis, and A. Ailamaki, "Adaptive htap through elastic resource scheduling," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pp. 2043–2054, 2020.

[33] M. Pinnecke, G. C. Durand, D. Broneske, R. Zoun, and G. Saake, "Gridtables: A one-size-fits-most h 2 tap data store," *Datenbank-Spektrum*, vol. 20, no. 1, pp. 43–56, 2020.

[34] J. Zhong and B. He, "Medusa: Simplified graph processing on gpus," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 6, pp. 1543–1552, 2013.

[35] B. Zhao, J. Zhong, B. He, Q. Luo, W. Fang, and N. K. Govindaraju, "Gpu-accelerated cloud computing for data-intensive applications," in *Cloud Computing for Data-Intensive Applications*, pp. 105–129, Springer, 2014.

[36] C. Lin, B. Mandel, Y. Papakonstantinou, and M. Springer, "Fast in-memory sql analytics on graphs," *arXiv preprint arXiv:1602.00033*, 2016.

[37] K. Zhang, K. Wang, Y. Yuan, L. Guo, R. Lee, and X. Zhang, "Mega-kv: A case for gpus to maximize the throughput of in-memory key-value stores," *Proceedings of the VLDB Endowment*, vol. 8, no. 11, pp. 1226–1237, 2015.

[38] T. Yamamuro, M. Onizuka, T. Hitaka, and M. Yamamuro, "Vast-tree: A vector-advanced and compressed structure for massive data tree traversal," in *Proceedings of the 15th International Conference on Extending Database Technology*, pp. 396–407, 2012.

[39] X. Si, A. Yin, X. Huang, X. Yuan, X. Liu, and G. Wang, "Parallel optimization of queries in xml dataset using gpu," in *2011 Fourth International Symposium on Parallel Architectures, Algorithms and Programming*, pp. 190–194, IEEE, 2011.

[40] S. Chen, L. Liu, W. Zhang, and L. Peng, "Architectural support for nvram persistence in gpus," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 5, pp. 1107–1120, 2019.