CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Comparative Performance and Scalability Analysis of GPU-accelerated Database Operations.

Harnessing Hardware for Database Efficiency:
A Comprehensive Benchmarking Study of V-Search, Fuzzy Search, and Join Operations on Central and Graphics Processing Units.

Master's thesis in Computer science and engineering

Carl Andersson, Jonathan Nilsson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

Comparative Performance and Scalability Analysis of GPU-accelerated Database Operations.

Harnessing Hardware for Database Efficiency:
A Comprehensive Benchmarking Study of V-Search, Fuzzy Search, and Join Operations on Central and Graphics Processing Units.

Carl Andersson, Jonathan Nilsson

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Comparative Performance and Scalability Analysis of GPU-accelerated Database Operations.
Harnessing Hardware for Database Efficiency: A Comprehensive Benchmarking Study of V-Search, Fuzzy Search, and Join Operations on Central and Graphics Processing Units.
Carl Andersson, Jonathan Nilsson

Supervisor: Pedro Petersen Moura Trancoso, CSE
Advisor: Örjan Vestgöte, Vesiro AB
Examiner: Pedro Petersen Moura Trancoso, CSE

Master's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Description of the picture on the cover page (if applicable)

Typeset in LaTeX
Gothenburg, Sweden 2023

Comparative Performance and Scalability Analysis of GPU-accelerated Database Operations.
Harnessing Hardware for Database Efficiency: A Comprehensive Benchmarking Study of V-Search, Fuzzy Search, and Join Operations on Central and Graphics Processing Units.
Carl Andersson, Jonathan Nilsson
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

This Master's thesis investigates the performance dynamics of database operations - V-Search, Fuzzy Search, and Join - implemented on both Central Processing Units (CPU) and Graphics Processing Units (GPU). With the ever-increasing demand for efficient data processing, it has become crucial to understand and optimize the use of different hardware platforms for executing diverse database tasks. As such, this research sheds light on the performance of each type of processing unit when running the said operations. The study first details the design and implementation of each database operation on both CPU and GPU, taking into account the different architectural characteristics and processing capabilities of each unit. The specific operations were chosen due to their wide use in the field of data management and their different processing requirements, which allows for a comprehensive performance analysis. Next, a series of benchmark tests is conducted to evaluate the relative performance of the CPU and GPU implementations. Factors such as data size, data type, and transfer time, among others are taken into account. The results show a detailed comparison of execution times between the two implementations, offering insights into the potential advantages and limitations of each. This work contributes to a better understanding of the trade-offs involved when choosing between CPU and GPU for database operations. We hope that our findings will inform future work on hardware-specific optimization for database systems, leading to more efficient and effective solutions for large-scale data processing tasks.

Keywords: Database, GPU, CPU, Performance, V-Search, Fuzzy, Join, CUDA.0

Acknowledgements

We would like to thank our supervisor Pedro Petersen Moura Trancoso for the support given to us throughout this project. We would also like to thank our Advisor Örjan Vestgöte, as well as Oskar Hagman and Oscar Widén at Vesiro for their continued guidance and input that has made this thesis possible.

Carl Andersson, Jonathan Nilsson, Gothenburg, 2023-07-04

Contents

List of Figures

List of Tables

1

Introduction

In the swiftly changing digital environment of today's world, the amount of data being generated and consumed continues to grow exponentially [1]. The spread of connected devices and the rise of Internet of Things (IoT) have contributed significantly to this data explosion, making it crucial for organizations and industries to manage, process, and analyze this vast amount of information efficiently. Concurrently, the performance of Graphics Processing Units (GPUs) has experienced an accelerated growth rate, at the same time as the effects of Moore's law is coming to an end.

This surge in data generation, coupled with the impressive advancements in GPU technology, has led to a growing interest in harnessing the potential of GPU-accelerated computing for database operations. By optimizing database performance and efficiency, not only can the effectiveness of businesses and organizations be increased but also contribute positively to global sustainability efforts. Efficient data processing and management can lead to reduced energy consumption, decreased resource usage, and lower operational costs, ultimately benefiting the world at large.

In this master's thesis, we focus on implementing three GPU-accelerated database operations and evaluating their performance in comparison to traditional CPU-based approaches, as well as investigating the factors that impact the performance of their execution. The specific database operations chosen for this project are string search using the Vesiro algorithm (V-Search), Levenshtein distance-based fuzzy search, and join operation using the hash join algorithm. By comparing and contrasting the efficiency of CPUs and GPUs in conducting these specific database operations, we aim to provide a more nuanced understanding of the potential benefits and drawbacks of each processing unit type. The outcome of this work aspire to provide valuable insights that can guide database administrators and developers in choosing the right hardware for specific database tasks, ultimately leading to more efficient and optimized data processing systems.

1.1 GPU-accelerated databases and their applications

With the introduction of NVIDIA CUDA in 2006, enabling general-purpose computing on GPUs, general-purpose computing on GPUs became increasingly adopted.

This eliminated the need for programmers to convert data into a graphical form to leverage the processing power of GPUs [2]. Taking advantage of this technology, GPU databases have emerged as a powerful solution for accelerating data processing and query performance.

In 2017 HEAVY.AI (formerly OmniSci) lanuched the world's first open source GPU database and SQL engine, enabling faster query execution and real-time data exploration [3]. SQreamDB, another notable GPU-accelerated database system, offers exceptional performance for data analytics and warehousing [4]. Additionally, there are implementations that extend existing databases, such as BrytlytDB and PG-Strom, both based on PostgreSQL, leveraging GPU processing for enhanced query execution and data analysis [5][6].

Although GPUs can achieve significant speedups for tasks that can be highly parallelized, such as data analytics and machine learning, it is important to acknowledge the limitations of GPUs. Operations that have low parallelism or rely on sequential processing are not well-suited for GPU acceleration. Additional limitations for GPU databases are small queries or operations that require frequent data transfers between the CPU and GPU. Despite achieving faster execution times on the GPU, if the data transfer time exceed the speedup, the overall performance may suffer. As a result, GPUs are most effective and yield the greatest benefits in big data applications, where their parallel processing capabilities and high-memory bandwidth can be fully leveraged.

1.2 Previous studies

Numerous past studies have underlined the remarkable performance benefits that GPUs, with their high parallel processing capabilities, can offer in database operations. One such study focused on executing aggregation functions, namely MIN, MAX, and SUM, within a CUDA kernel. In fact, when compared to CPU execution using MySQL, an average speedup of 97 times was observed [7]. This staggering performance difference attests to the significant potential of GPU-based computation for these types of functions.

In another study, GPU acceleration was incorporated into SQLite. The results, documented in [8], showcased an average speedup of 35 times for numerical select queries. Again, the use of a GPU proved to be a game-changer, demonstrating impressive acceleration of the query processing. The use of GPUs has also been found to outperform both MySQL and Apache Spark for large data queries [9]. However, when it comes to smaller data queries, CPUs still tend to be more efficient, indicating that each processing unit might have superior performance depending on the context.

String matching, exact string matching as well as approximate string matching, has also been studied and found to benefit from the parallel processing capabilities of GPUs [10][11][12].

Building on these explorations, research has ventured into additional optimization

strategies to further amplify the power and efficiency of GPU-accelerated databases. One such approach is the concurrent processing of queries, an advancement over the traditional sequential execution model. This method leverages the inherent parallel processing capabilities of GPUs to simultaneously process multiple queries, dramatically reducing overall execution time and boosting system throughput[13].

Moreover, the concept of in-memory GPU databases has sparked considerable interest. These databases store data directly on the GPU's memory, significantly reducing the overhead associated with data transfer between the CPU and GPU. This approach is particularly advantageous when working with large datasets and complex queries, where the transfer costs can otherwise significantly impact performance for execution on GPU [14].

1.3 Aim

The objective of this master thesis is to implement and evaluate the performance and scalability of three GPU-accelerated database operations in relation to their CPU counterparts, while also examining the different factors that impact the performance of each operation and device with the overall goal to evaluate the potential of GPU-accelerated computing in database operations.

1.4 Problem definition

Outlined below are key issues and questions that this study aims to address and resolve in order to achieve the stated aim.

1. **Choice of Algorithm:** The choice of algorithm for the implementation of the operations will significantly affect the efficiency and performance of these operations. An important aspect to consider when selecting algorithm is whether it is fair to use the same algorithm on both the GPU and CPU. This involves investigating the compatibility of the chosen algorithm with both types of processors, as well as the trade-offs associated with potential variations in the algorithmic design for each.

2. **Test Data Selection:** Selecting appropriate test data is key to correctly evaluating the performance of operations on both the CPU and GPU. It's essential to carefully consider which test data to use in these performance evaluations, ensuring that it not only represents real-world application usage but also enables others to make an accurate comparison of the result.

3. **Testing Procedure:** A well-defined testing procedure is necessary to ensure a fair comparison of performance between the CPU and GPU. This raises the question of how the operations should be tested on each processor, considering factors like workload distribution and redundancy. Given that data transfer times to the GPU can significantly influence overall performance, it is important to consider how tests should be conducted to account for this factor as

well as to establish a method for when to include data transfer times in the test results.

4. **Configuration impact:** Variations in hardware configurations can have great impact on performance. It is important to understand the impact that specific hardware properties can have on performance. Similarly, database configurations can significantly affect the efficiency of operations, and it is therefore essential to analyze which properties have the most impact on performance.

5. **CPU vs. GPU Usage:** Finally, the most critical aspects of this thesis is to determine if, and in that case when, it becomes more beneficial to use the GPU for the operations as opposed to the CPU. This involves understanding the factors that drive this decision and developing an execution plan that can guide this choice in real-world applications.

1.5 Scope and Limitations

This master's thesis focuses primarily on the implementation and performance evaluation of three specific GPU-accelerated database operations. It's important to note that, while there could be numerous ways to implement these operations, this work only takes into account a single algorithm approach for each operation. The objective is to understand the practical applicability and performance implications of GPU acceleration for these database operations and not to compare different algorithms for the same operation.

It is also worth mentioning that this study does not provide any specific hardware recommendations based on the results obtained. This is primarily because the focus of the research is not to suggest specific hardware configurations, but to evaluate the potential of GPU-accelerated computing in database operations.

When considering performance, this thesis primarily focuses on the speed of operations, rather than energy performance. Although it is acknowledged that these two aspects often overlap, the specific energy efficiency of the operations is outside the scope of this study.

In terms of hardware, this study exclusively considers NVIDIA GPUs, with CUDA serving as the development tool. Other GPUs and development tools are available but none with the same widespread adoption and development support.

Lastly, it should be noted that the database operations considered in this thesis are not limited to any specific type of database. The operations themselves are generic and could potentially be adapted for different types of databases and indicate performance benefits for other operations. Therefore, while the focus is on three specific operations, the findings of this work could have broader implications for the field of database management and GPU computing.

1.6 Thesis Organization

The subsequent chapter provide background information required to easily follow the rest of the thesis, including CUDA, OpenMP, AWS and a description of the selected database operations. Following that, chapter 3 gives a brief description of the implementation of the three database operations, the hardware and data sets used for the performance testing, and the test queries used with these datasets. Chapter 4 then presents the result from the performance tests for both CPU and GPU executions for multiple impacting configurations including database sizes and data sets. Building upon the results presented in Chapter 4, Chapter 5 then presents a comprehensive analysis of the performance tests conducted, accompanied by the corresponding outcomes and findings. Lastly, Chapter 6 concludes the thesis by summarizing the key findings and contributions.

2

Background

The following background section offers an overview of the relevant aspects associated with our investigation into comparing the performances of CPUs and GPUs in executing selected database operations. In the first subsection we delve into the intricacies of the architecture of NVIDIA's GPUs and the CUDA parallel computing platform. We explore how NVIDIA's architectural choices facilitate high-performance computing and how CUDA enables software developers to use a CUDA-enabled GPU for general purpose processing. In the subsequent subsection, we provide an overview of OpenMP, detailing its functionalities and how we have leveraged it in our research. Following this, we present an overview of AWS (Amazon Web Services) and expand on its role in our project. Finally, in the fourth subsection, "Selected Database Operations", we present a detailed description of the three chosen operations for our study, V-Search, fuzzy Search, and join. This section explains the rationale behind their selection and their importance in the realm of database management.

2.1 NVIDIA architecture and CUDA

The architecture used in the NVIDIA GPU's used in this thesis is the NVIDIA Ampere architecture [15]. The architecture is divided into streaming multiprocessors (SMs), where each SM consists of CUDA cores, Tensor Cores, Texture units, one Ray tracing core and a L1 cache/Shared memory. Each SM uses a Single Instruction Multiple Thread (SIMT) architecture and handles threads in groups of 32 parallel threads called warps [16]. Each thread in a warp start at the same program address, but since they all have their own register state and instruction address counter they are free to operate separately. The warp scheduler in each SM is allowed to switch executing between warps for free since each warps execution context is maintained on-chip during the whole lifetime of the warp.

CUDA, a general purpose computing platform and programming model, allows developers to use NVIDIA GPUs to solve problems in a more effecient way by parallelizing them on the many core architectures offered by GPUs. CUDA C++ extends C++ so developers can create C++ functions, called kernels, that allows them to parallelize there code into block and threads [16]. These blocks of threads are divided to the available SMs and then partitioned into warps by the warp scheduler. The maximum number of warps per SM depends on the graphics cards compute capability,

however the number of active warps on a SM varies over time and is determined by four factors; Warps per SM, Blocks per SM, Registers per SM and Shared Memory per SM. Together these four determine the maximum number of active warps each SM can have and thus also the occupancy per SM, which is defined as the ratio between number of active warps and maximum number of active warps.

2.2 OpenMP

Open Multi-Processing (OpenMP) is a widely used, industry-standard API that supports shared-memory parallel programming in C, C++, and Fortran [17]. It is designed for multi-processor, multi-core, and multi-threading computing systems and provides a simple and flexible interface for developing parallel applications. Unlike other parallel computing methods, OpenMP uses a directive-based approach to parallel programming. These directives allow developers to mark areas of code to be executed in parallel, forcing the compiler to handle the creation, synchronization, and termination of threads.

In the context of our study, we use OpenMP to fully utilize the CPU's capabilities by distributing the processing load across its multiple cores. This helps us make a fair comparison with GPU execution and provides a benchmark for assessing the performance improvements achieved through GPU-accelerated database operations.

2.3 Amazon Web Services (AWS)

AWS is a comprehensive cloud service platform that offers a wide range of cloud based services, including Elastic Compute Cloud (EC2) [18]. EC2 is a service that offers virtual computers, called instances. Instances can be launched with different operating systems and a large variety of hardware combinations of CPU, GPU, network capacity, memory, and storage.

In this project EC2 instances with different CPU and GPU configurations have been used for the performance tests. The usage of EC2 for this research has enabled a more diverse set of hardware that would have otherwise been possible.

2.4 Selected database operations

2.4.1 V-search

The first implemented and tested operation focused on locating a string in a sorted list. A variety of methods can be applied to achieve this, including sequential and binary search techniques. Our approach utilizes the Vesiro algorithm, which has been specifically optimized to better utilize core performance during the search process.

2.4.2 Fuzzy search

The second operation implemented is fuzzy search or approximate string search. Fuzzy search works by creating a measurement of the similarity between the search term and the terms being searched, the terms with a similarity measurement under a certain value is then returned as the result. Multiple similarity measurement methods exist like Hamming Distance that measure the number of differing characters in string of the same length, n-grams that divides the strings into sub-strings of length n and measures their similarities and Levenshtein distance that is a measurement that return the number of edits required to transform one string into another using the edit operations insert, delete and replace [19]. Levenshtein distance is the method used in this thesis.

Previous studies have investigated the performance of calculating the Levenshtein distance on GPU's but have mainly focused on comparing the performance of the actual computation as opposed to the performance in the context of a database query [20][21][22]. Little attention has been given to the impact on performance of the word length and at what size of the search space is needed for each device to be superior.

2.4.3 Join

The last operation implemented is the join operation that combines values from two tables into one table using common values from each table. Multiple methods can be used to achieve this operation including Nested Loop Join that for each record compares the join key in every record in the other table and Sort-Merge Join that first sort all records by the join key and then iterates through the sorted tables combining matching records. The algorithm chosen for this thesis is the Hash Join algorithm that works by creating a hash table from one of the tables using the join key as the hash key. The other table is then iterated through and its join keys is used as hash keys to find matching records in the first table.

Moreover, evidence from literature indicates the superiority of the Hash Join algorithm for join operations, especially when executed on GPUs[23]. It consistently exhibits substantial speed enhancements compared to other algorithms. This is further corroborated by another study, revealing that the implementation of GPUs for join operations can catalyze an impressive speedup, potentially up to 20 times faster than the execution on CPUs[24]. Such findings underline the significant advantages that can be gained by harnessing the power of GPUs in database operations.

3

Methodology

This chapter describe the methodology used in this research, providing a comprehensive overview of the process undertaken to evaluate and compare the performance of the CPU and the GPU in executing the specific database operations. The approach consists of selecting appropriate datasets and algorithms, conducting performance tests, and analyzing the results in a comparative manner. The methodology aims to strike a balance between experimental control and practical relevance, ensuring that the results obtained are both scientifically valid and applicable to real-world scenarios.

The chapter is organized as follows: firstly, an explanation of how the chosen operations have been implemented is given. Subsequently, an overview of the computing hardware and services used to execute the performance tests is provided, detailing the specific system configurations and cloud-based platforms utilized. Thereafter follows a comprehensive overview of the various datasets utilized for each operation. Lastly, a description of the queries used for the performance test and specifics for the test is given.

3.1 Operations

3.1.1 V-Search using the Vesiro algorithm

As Vesiro intends to maintain the confidentiality of their search algorithm, the specific details of its implementation will not be elaborated upon here. It is important however to emphasize that executing a single search with V-Search does not offer substantial opportunities for parallelization and also requires overhead for the GPU in the form of copy the search term to the GPU for every search. The performance for a single search executed on the GPU does therefor not exceed the performance of a CPU. This is why single searches have not been the central focus of this test. Instead, the evaluation methodology adopted primarily involves quantifying the volume of searches conducted in batches that is required to surpass the throughput of searches accomplished by the CPU. This approach gives a more fair comparison as the advantage of the GPU is its parallelization capability.

It is worth mentioning that the search functions for both CPU and GPU has been optimized with OpenMP to ensure that the result reflects the full capability of the devices. For the CPU, OpenMP ensures that the CPU executes multiple searches

in parallel, depending on the number of cores available to the CPU. For the GPU OpenMP ensures that multiple kernels are created in parallel to avoid the sequential execution of kernels for low batch sizes.

3.1.2 Fuzzy search using Levenshtein distance

To calculate the Levenshtein distance between two strings, String1 and String2, a cost matrix is created of size (L1+1, L2+1) where L1 is the length of String1 and L2 the length of String2. Each value [x, y] of the matrix does then represent the Levenshtein distance between the first x number of characters of String1 and the first y number of characters of String2. The first row and column is then initialised to 0,1,2..., as the first row and column represents the insertion of characters to form String1 and String2. The rest of the cost matrix is then filled row after row by taking the minimum cost of each operation. Since [x-1, y] is the Levenshtein distance of the first x-1 characters of String1 and the first y characters of String2, the cost of the deletion operation becomes [x-1, y]+1. The cost of insertion is then similarly calculated with [x, y-1]+1. Finally the substitution cost is calculated by taking the value [x-1, y-1] plus the cost of substituting the x character of String1 with the y character of String2, which is 1 if they differ and 0 if they are equal.

As the Levenshtein distance computation offer little parallelisms due to its dependency on previously computed matrix values, OpenMP has not been used to parallelize the computation for CPU execution. Neither has it been used to parallelize the words searched as early testing showed significant decreased performance. GPU execution has however parallelized the words searched by dividing the search space for each kernel thread.

3.1.3 Join using Hash Join

Two functions are used to implement the hash join operation for which there are two implementations of each, one for CPU and one for GPU. The first function is the build function, which constructs the hash table from the build table using the join element as the key for the hash table. The join element and the index of the row is then saved in the hash table. To manage collision cases and insertion time, the hash table's size is set to double the size of the build table and when a collision occurs, the colliding records is placed in the next available slot in the hash table.

The second function, the probe function, enables the probe table to perform lookups in the hash table to identify matches. To manage the collision avoidance mechanic, it looks up the position in the hash table given by using the join attribute as key and initiates a loop that makes comparisons between the join attributes until either an accurate match or an empty slot is found. If a match is found, the index of the build table saved in the hash table is then saved in a new array on an index matching the row from the probe table.

After these two functions has been executed, an array with corresponding indexes for each table is returned. The array is then used to combine data from the corresponding rows of both tables on the CPU. The reason for combining the rows from

the tables on the CPU is that it enables us to only transferring the columns of the join element for both tables to the GPU instead of both whole tables, decreasing the amount of data transferring overhead significantly.

As both tables are stored as arrays of bytes, a struct with the number of columns, number of rows, the size of each row, index to the join key, and an array of indexes to the columns is required for accurate indexing. The declaration of the struct can be seen below.

```
struct TableStruct {
    uint32_t num_columns;
    uint32_t num_rows;
    uint32_t row_size;
    uint32_t join_index;
    uint32_t* column_indexes;
};
```

3.2 Hardware and services used

Throughout the development phase two dedicated workstations were utilized, each equipped with Intel i5 CPUs and NVIDIA Ampere series GPUs. In the subsequent testing phase, additional hardware resources with diverse configurations was utilized to contribute to a deeper understanding of performance variations across different setups. For access to additional hardware AWS was used. By leveraging AWS services, a more comprehensive analysis of scalability and stress-testing on each implemented operation was made possible. The combined use of dedicated workstations and AWS during the development and testing phases ensured efficient development and extensive assessment of performance, scalability, and stress resilience across different hardware configurations.

Three AWS hardware instances of different compute capabilities were chosen to examine the scalability of the operations and the impact of different hardware configurations. Detailed specifications of all three hardware instances is described in Table 3.1. The choice of performing all test on EC2 also ensures that the result is comparable between the different machines.

	Instance 1	Instance 2	Instance 3
CPU	Intel Xeon E5-2686 v4	Intel Xeon Cascade Lake P-8259L	AMD EPYC 7R32
Cores	16	32	64
RAM	122 GB	128 GB	256 GB
GPU	NVIDIA Tesla M60	NVIDIA T4 Tensor Core	NVIDIA A10G Tensor Core
VRAM	8 GB	16 GB	24 GB

Table 3.1: Hardware specifications

3.3 Test data

3.3.1 V-Search and Fuzzy Search

The test data utilized for the V-Search and fuzzy search operations consists of randomly generated words ranging from 3 to 20 characters in length. The objective was to cover a variety of scenarios and accurately assess the performance of GPU-accelerated operations compared to CPU-based operations. To simulate different database sizes, lists with varying numbers of rows were created, starting from one thousand rows and increasing by a factor of 10 up to 100 million. Notably, for the V-Search evaluation, list of size one thousand and 10 thousand were not used, due to the design of the test queries used.

This diverse test data enabled a comprehensive evaluation of the scalability, efficiency, and limitations of the operations. This approach allowed for a thorough examination of the performance characteristics and limitations of GPU-accelerated operations, as well as a comparative analysis against CPU-based approaches.

3.3.2 Join

The join operation deals with situations that need to bring together data from different tables. Consequently, a testing environment reflective of conventional SQL data is necessitated. Datasets designed specifically for SQL benchmarks, embodying a range of data similar to those found in practical SQL environments, are highly appropriate for this purpose, satisfying our testing requirements effectively. For this reason the dataset from TPC-H (Transaction Processing Performance Council Benchmark H) benchmark suite was chosen. This dataset consist of a number of data tables considered to represent industry use cases and is widely used in industry [25]. It is also a dataset that is possible to scale which makes it well suited for testing different data sizes to find a performance breaking point between CPU and GPU. Most of the table sizes are defined by a scaling factor that when used to generate the dataset corresponds to the number of GB of the whole dataset. Meaning that a scale factor of 1 generates tables that sum up to 1 GB of data. In this project two tables are utilized: the customer table and the orders table. The size of the customer table is determined by multiplying the scale factor by 150,000 (number of rows in the table), while the size of the orders table is determined by multiplying the scale factor by 1.5 million. These two tables relate through the primary key named CUSTKEY.

3.4 Selected test queries

3.4.1 V-Search and Fuzzy Search

All test queries, regardless of the operation type, are first categorized based on the different file sizes ranging between one thousand to 100 million rows of data. Then they are divided into four different categories based on the length of the words being

searched for. These four categories include:

1. Short Terms: Words with a length between 3 and 8 characters.
2. Middle Terms: Words with a length between 8 and 15 characters.
3. Long Terms: Words with a length between 15 and 20 characters.
4. Random Terms: Words with varying lengths between 3 and 20 characters.

For the operation V-Search search, each query involved searching for 10,000 words. However, it was quickly realized that searching for one word at a time was not beneficial for the GPU. The 10,000 words were therefore searched for in different batch sizes, namely: 1, 5, 10, 50, 100, 500, 1000, 5000, 10000. This enabled a continued exploration of the overall efficiency and performance to GPU-accelerate V-Search search.

Conversely, the queries for the operation fuzzy search consisted of 25 words in each query, as opposed to the 10,000 words used in the V-Search search. These 25 words were distributed across the same word length categories (short, middle, long, and random), ensuring a comprehensive evaluation of the operation on both CPU and GPU.

3.4.2 Join

While the TPC-H benchmark suite includes not only test datasets but also a variety of queries for performance measurement, these queries were not utilized in the course of this thesis. This decision stems from the fact that a fully operational DBMS was not developed throughout the thesis work. Despite the join operation being fully functional, the benchmark queries incorporate numerous other directives such as aggregate functions and sorting, which were not the central focus of this study.

The queries formulated for evaluating the join operation involve an inner join between two tables, namely the customer and orders tables. In this setup, the customer table consistently serves as the build table, since its possession of the unique primary key. Conversely, the orders table assumes the role of the probe table. The queries were categorized into five distinct groups based on the relative scale between the tables. These categories are denoted as: 100:1, 10:1, 1:1, 1:10, and 1:100. The numbering corresponds to the size relationship between the build table and the probe table, where the sizes of the tables are measured in terms of the number of rows they contain. By employing this categorization scheme, the evaluation aims to investigate the how the size of the build and probe table impact performance and were each phase should be run, whether if it is on the GPU or the CPU.

4

Results

In this chapter, we present the findings from the testing of the three operations. This chapter solely discloses the test results, a comprehensive evaluation of these outcomes will be provided in the subsequent chapter. First the result from the V-Search operation is presented, followed by the result from the fuzzy search operation, and lastly the result from the join operation is presented. It's important to mention that the complete set of results is rather extensive, and for this reason, it's fully outlined in the appendix. In this chapter, we present selected portions of the full results, intending to provide a representative overview of the dataset without overwhelming the reader.

4.1 V-search

In this section the result from the performance test of the V-Search operation is presented for both CPU and GPU. Initially, the outcomes derived from CPU tests is presented, followed by those from the GPU tests, and finally a comparison is between all computing devices is presented.

Performance test results for each CPU on Instance 1, 2 and, 3 are presented in Table 4.1. The column furthest to the left displays the size of the database tested, whereas the topmost row indicates the dataset utilized. As observed from the tables, the term length had a greater impact on performance than the size of the database.

	Random Terms	**Short Terms**	**Middle Terms**	**Long Terms**
100K	2.14	0.92	2.03	3.29
1M	2.49	0.99	2.29	3.80
10M	2.14	0.82	1.92	3.13
100M	2.64	1.07	2.30	3.76

(a) Instance 1

	Random Terms	**Short Terms**	**Middle Terms**	**Long Terms**
100K	2.26	0.88	2.18	3.57
1M	2.31	0.91	2.17	3.56
10M	2.39	0.81	2.19	3.61
100M	2.52	0.84	2.24	3.67

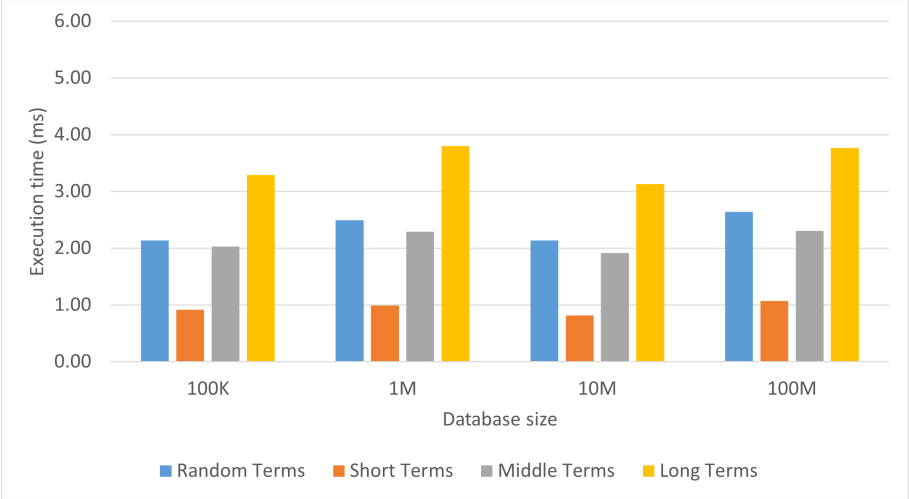
(b) Instance 2

	Random Terms	Short Terms	Middle Terms	Long Terms
100K	2.35	0.81	2.22	3.74
1M	2.40	0.86	2.21	3.73
10M	2.47	0.84	2.20	3.74
100M	2.56	0.87	2.20	3.76

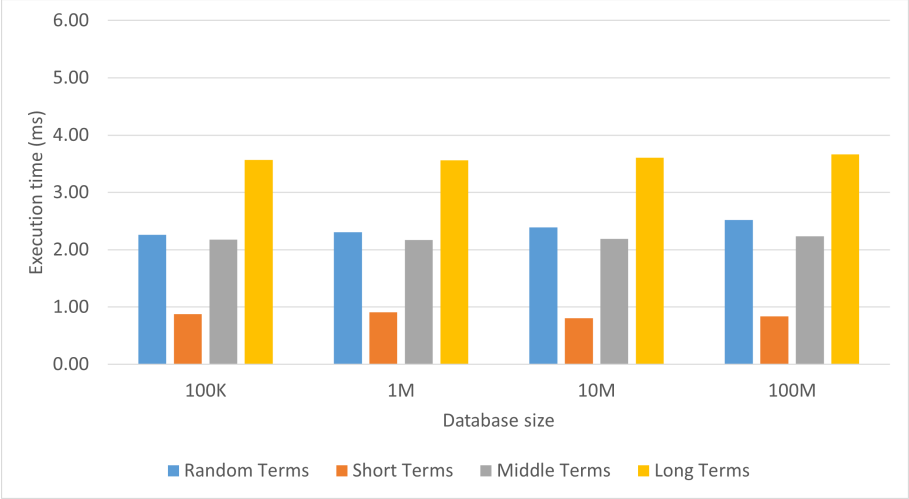
(c) Instance 3

Table 4.1: Term length comparison between database sizes (CPU)

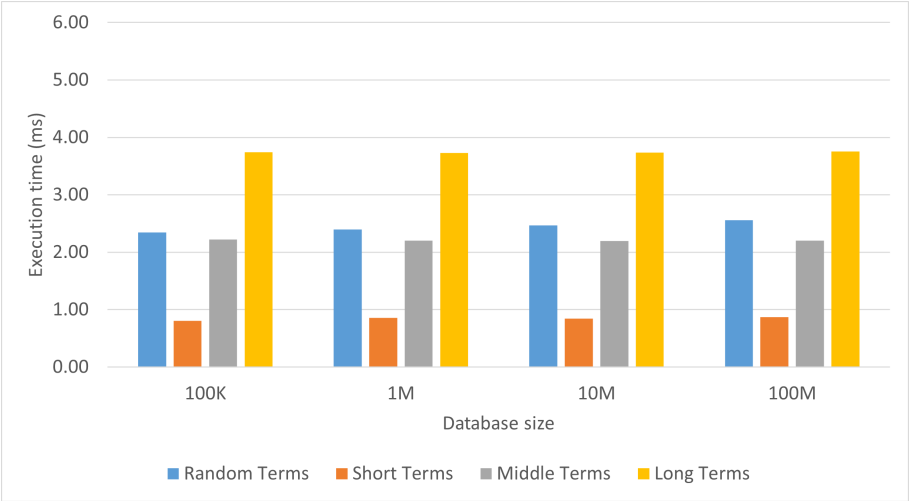
For a more intuitive comprehension of the results, data visualization is provided in Figure 4.1. In these diagrams, database sizes are grouped, and the datasets are denoted by color. The charts reveal a consistent performance discrepancy across the datasets for every instance. Notably, each instance demonstrates signs of great scalability.



(a) Instance 1



(b) Instance 2



(c) Instance 3

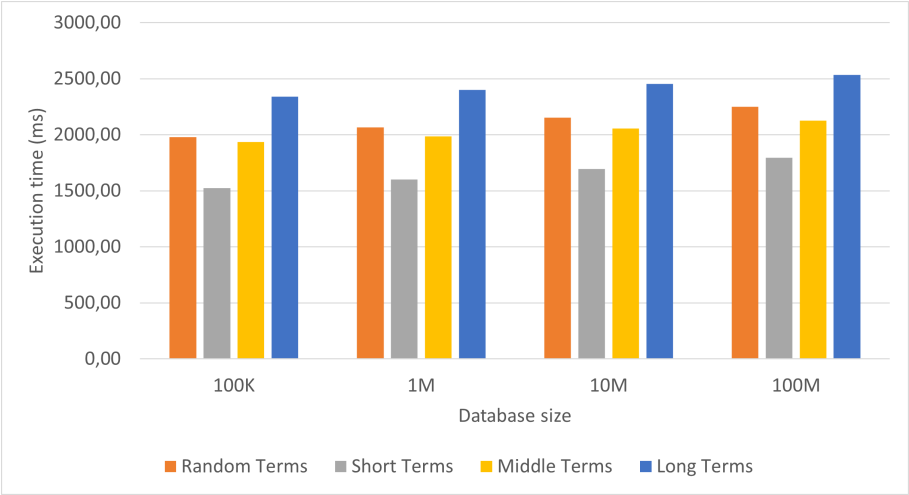
Figure 4.1: Term length comparison for V-Search (CPU)

Given the GPU's memory is separate from the CPU's, it requires a transfer of the database to the GPU prior to any operation execution. Table 4.2 presents the duration required to transfer databases of varied sizes. Since this transfer time considerably surpasses the time taken for the CPU to execute the operation, the strategy of fully transferring the database and then conducting the operation is impractical from a performance perspective. However, assuming the database has been pre-transferred to the GPU's memory, effectively creating an in-memory database, performance could potentially be enhanced through GPU utilization.

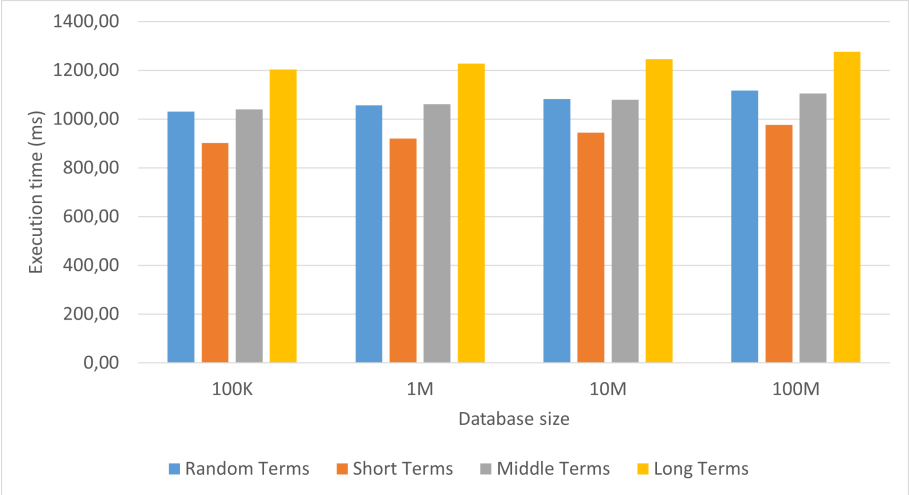
	Instance 1	**Instance 2**	**Instance 3**
100K	142.51	152.49	136.23
1M	146.76	159.36	134.99
10M	190.67	231.45	167.36
100M	648.18	1122.92	617.35

Table 4.2: Database transfer times in ms, (CPU to GPU)

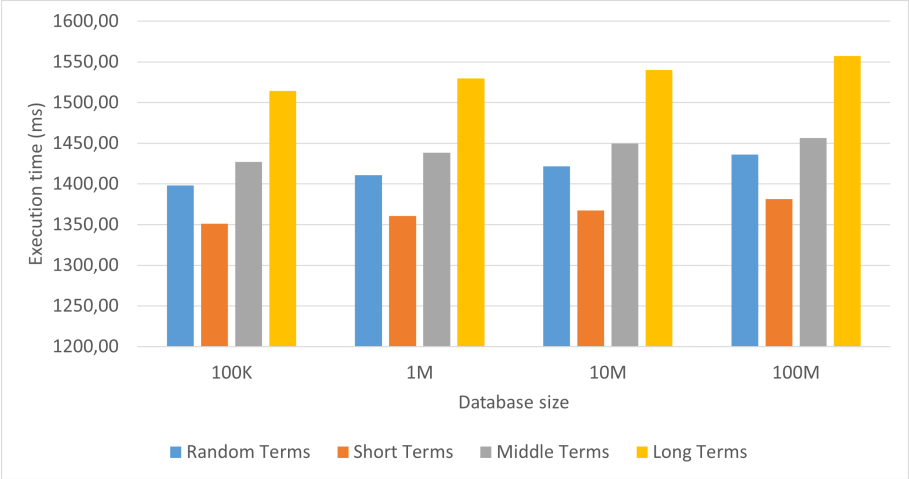
To illustrate the impact term length had on performance when run on the GPU, we present the execution times for each instance in Figure 4.2. Consequently, the GPU:s exhibit a similar performance discrepancy as the CPU:s, between the four datasets, for every instance. In contrast to the significant disparity observed in execution time between short terms and long terms tasks on the CPU, reaching up to a threefold difference, the GPU demonstrates a comparatively modest increase of up to 1.5 times.



(a) Instance 1



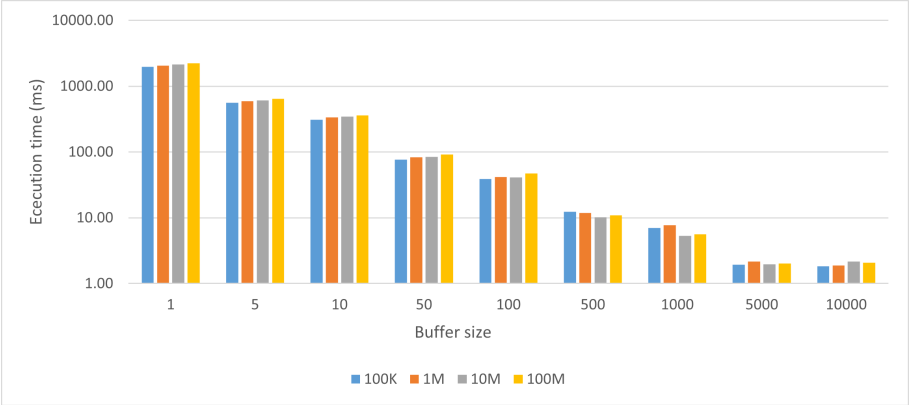
(b) Instance 2



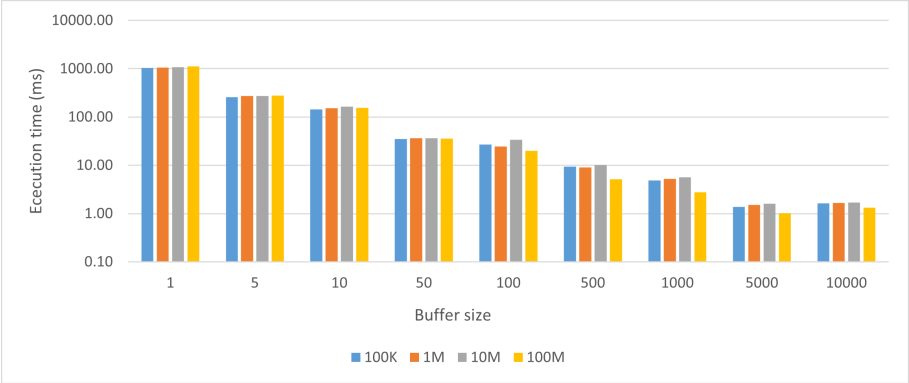
(c) Instance 3

Figure 4.2: Term length comparison for V-Search (GPU)

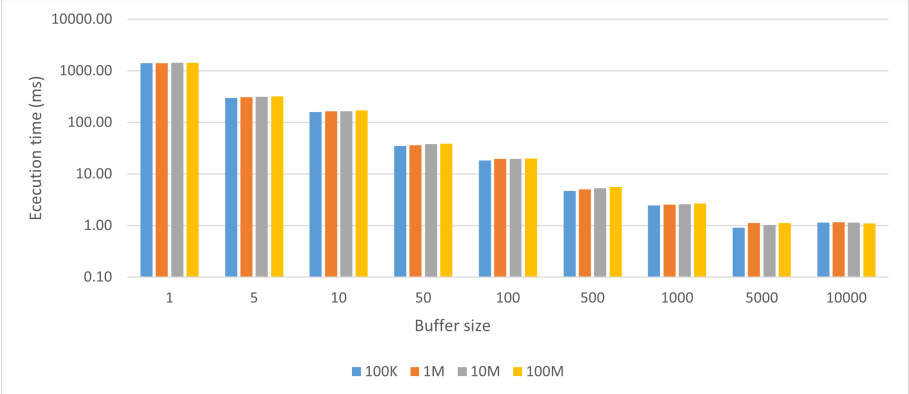
The execution performance of the GPU is notably affected by the selection of a buffer size for grouping searches. In Figure 4.3, the execution time of a workload comprising 10,000 randomly selected terms is depicted, considering various buffer sizes. It is observed that as the buffer size increases, thereby reducing the number of created kernels and associated overhead, a significant decrease in execution time is achieved. Interestingly, the impact of the database size on the execution time is found to have a lesser impact in comparison to the buffer size.



(a) Instance 1



(b) Instance 2



(c) Instance 3

Figure 4.3: Database size comparison for V-Search (GPU)

As evident from the previous results tables, when buffer searches are performed, the GPU's execution time can surpass the CPU's. This performance pattern is particularly notable with a dataset of entirely random terms and a database size of 100M, as depicted in Figure 4.4. The graph demonstrates that the GPU starts outperforming the CPU when the buffer size is of a magnitude 1000, though the exact breakpoint largely depends on the specific CPU and GPU under comparison.

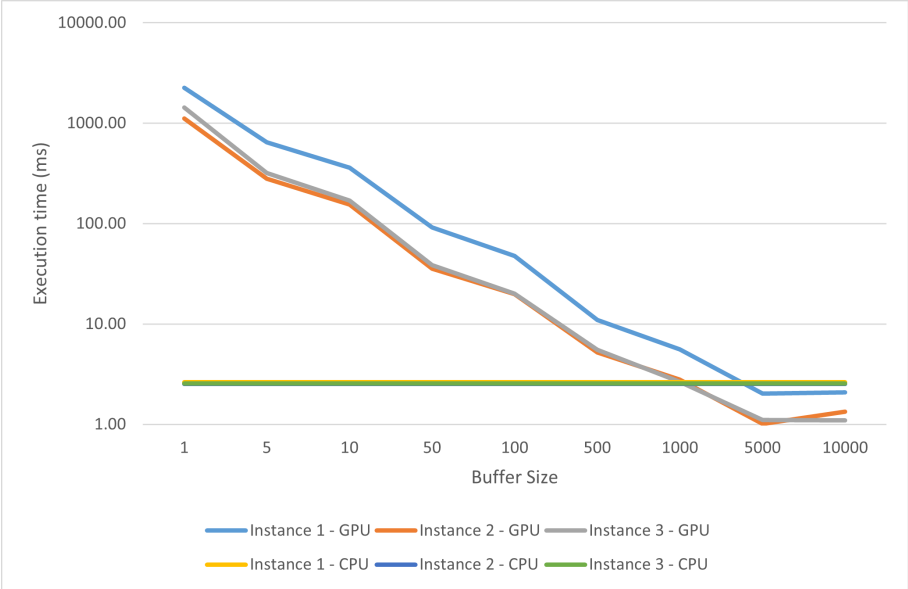


Figure 4.4: CPU and GPU comparison over buffer sizes for random terms

4.2 Fuzzy search

This section highlights the results of the performance evaluation for the fuzzy search operation. Because of the substantial execution times, not all dataset sizes were utilized for conducting the CPU-based fuzzy search tests. Only the 1K, 10K, and 100K dataset sizes were utilized. However, for GPU-based testing, all dataset sizes were employed as it demonstrated significantly lower execution times, particularly for larger databases.

On the CPU side, the fuzzy search operation results exhibited considerable variability based on the dataset size and length of terms used, which can be seen in Tables 4.3. For smaller datasets, the CPU demonstrated a robust performance, successfully managing the operation with reasonable speed. This can be attributed to the low overhead involved in initiating the computation process for CPU. As the dataset size increased, the operation's performance showed a proportional decrease, scaling linearly.

Given that the computational requirements for calculating the Levenshtein distance between two strings are contingent on their character count, the substantial disparity in execution times across different test datasets is to be expected.

	Random Terms	Short Terms	Middle Terms	Long Terms
1K	25,83	11,47	21,00	34,55
10K	229,18	117,06	211,16	342,80
100K	2341,63	1170,09	2135,09	3529,76

(a) Instance 1

	Random Terms	Short Terms	Middle Terms	Long Terms
1K	21,16	9,60	17,50	28,72
10K	190,99	98,64	176,23	285,48
100K	1952,95	986,56	1780,96	2936,19

(b) Instance 2

	Random Terms	Short Terms	Middle Terms	Long Terms
1K	26,67	12,33	22,18	36,24
10K	241,85	125,98	222,97	359,55
100K	2474,57	1261,90	2261,35	3700,77

(c) Instance 3

Table 4.3: Execution times of fuzzy search in ms (CPU)

In contrast to the CPU result, the GPU demonstrated a significantly better performance advantage for larger datasets. As can be seen in Table 4.4, the execution times scales quite well, far better than the linear scaling from the CPU. However, for smaller datasets, the GPU's performance was not as impressive. The overhead involved in transferring data between the host and the device, combined with the initialization of the GPU kernels, resulted in a delay that was noticeable for smaller operations.

	Random Terms	Short Terms	Middle Terms	Long Terms	Transfer Time
1K	3,59	1,63	2,93	4,76	146,35
10K	20,04	12,33	18,93	27,59	146,21
100K	49,84	44,06	49,70	55,55	147,57
1M	355,79	302,13	352,58	415,34	152,32
10M	2906,49	2514,35	2782,73	3336,39	166,69

(a) Instance 1

	Random Terms	Short Terms	Middle Terms	Long Terms	Transfer Time
1K	2,40	1,19	1,99	3,10	162,18
10K	13,02	7,99	12,19	17,96	151,05
100K	31,72	28,20	30,55	37,57	153,43
1M	223,19	183,27	209,63	271,65	156,82
10M	2071,13	1701,49	1937,32	2499,67	182,24

(b) Instance 2

	Random Terms	Short Terms	Middle Terms	Long Terms	Transfer Time
1K	2,26	1,08	1,86	2,99	151,89
10K	12,52	7,73	11,80	17,60	135,99
100K	29,65	26,94	28,17	34,45	139,34
1M	167,05	158,32	168,58	173,29	141,36
10M	1297,18	1299,37	1266,30	1343,33	155,64

(c) Instance 3

Table 4.4: Execution times of fuzzy search in ms (GPU)

However, the discrepancy in execution times across the different test sets is notably diminished for the GPU, especially with respect to the larger datasets. This could be attributed to the GPU's architecture, which processes multiple threads in warps. The execution times of a thread within a warp is influenced not solely by the length of the strings pertaining to that specific thread, but also by the length of the strings associated with the other threads in the same warp.

Figure 4.5 presents a comparison of execution times for both CPU and GPU across all three instances. The visual representation clearly demonstrates that for smaller datasets, utilizing the CPU for searches yields superior results. However, when dealing with larger datasets, leveraging the GPU becomes more beneficial.

Figure 4.5: Execution times for fuzzy search using random dataset

Across all instances, a similar performance tipping point is observed - the GPU starts to outperform the CPU just as the database size approaches 10,000 entries. Additionally, the figure illustrates that the performance gap between the individual instances is substantially less significant than the performance divergence between the CPU and GPU. This highlights the crucial role of device choice in determining overall performance, as opposed to the specific instance used.

4.3 Join

This section outlines the performance analysis results for the join operation. Performance testing was conducted using various scaling ratios between the tables, alongside different sizes corresponding to these ratios. The scaling factors for the tables are as follows: 100:1, 10:1, 1:1, 1:10, and 1:100. In this context, the table with a scaling factor of 1 signifies the table whose row count matches the number provided in the left column. Due to memory constraints for both CPU and GPU, it was not possible to test all table sizes across every CPU and GPU. Attempting to circumvent this limitation by subdividing the operation could potentially impact performance. Hence, the decision was made to exclusively test scenarios where all data could fit into memory.

A substantial portion of the join operation's execution time is devoted to data transfer, necessitated by the merging of the tables following the indexing process. As this segment of the operation is uniformly executed on the CPU for both CPU and GPU executions, it introduces a constant factor into our execution time measurement. Therefore, to ensure a fair and more precise comparison of execution times across devices, this data transfer time will be excluded from the subsequent presentation of results. By focusing solely on device-specific execution times, we can gain a more accurate understanding of each device's performance. The time spent on transferring data is however important to know and is therefor presented for all table combinations in Table 4.5.

Table Size	Scale 100-1	Scale 10-1	Scale 1-1	Scale 1-10	Scale 1-100
15K	3,27	3,07	3,03	48,35	413,54
75K	17,16	17,25	17,16	253,37	2072,69
150K	47,25	47,05	47,11	531,45	4219,88
750K	252,63	255,81	253,08	3078,34	4372,95
1500K		541,22	529,74	6305,86	
7500K		3031,99	3059,62	7002,70	
15000K			6235,89		
75000K			7011,38		

(a) Instance 1

Table Size	Scale 100-1	Scale 10-1	Scale 1-1	Scale 1-10	Scale 1-100
15K	2,48	2,45	2,39	34,40	311,40
75K	16,03	14,75	15,28	182,18	1567,26
150K	34,64	34,25	34,33	374,91	3228,70
750K	187,00	181,52	181,09	2207,86	3345,00
1500K		380,12	376,44	4661,34	
7500K		2244,98	2234,07	5155,41	
15000K			4547,73		
75000K			5130,48		

(b) Instance 2

Table Size	Scale 100-1	Scale 10-1	Scale 1-1	Scale 1-10	Scale 1-100
15K	1,40	1,39	1,37	16,73	162,67
75K	8,59	9,10	8,50	114,03	834,07
150K	16,68	16,75	16,50	240,15	1712,46
750K	113,72	113,92	114,31	1308,11	1889,41
1500K		233,81	233,89	2916,95	
7500K		1305,02	1311,40	3544,66	
15000K			2880,65		
75000K			3496,31		

(c) Instance 3

Table 4.5: Transfer time to construct join table in ms (GPU)

The execution times for CPU execution for all scaling combinations and possible sizes are detailed in Table 4.6 and illustrated in Figure 4.6. Both the table and figure show that the execution times are influenced by the size of the tables and their scaling ratios. As expected, larger tables correlate with extended execution times. Interestingly, execution times also rise when the build table exceeds the size of the probe table.

Table Size	Scale 100-1	Scale 10-1	Scale 1-1	Scale 1-10	Scale 1-100
15K	20,88	3,78	1,77	3,45	15,34
75K	133,35	11,52	3,28	8,95	75,33
150K	282,56	21,72	4,95	16,29	155,17
750K	358,48	138,07	17,68	82,21	183,27
1500K		297,50	33,92	177,40	
7500K		444,61	215,33	358,98	
15000K			471,64		
75000K			583,96		

(a) Instance 1

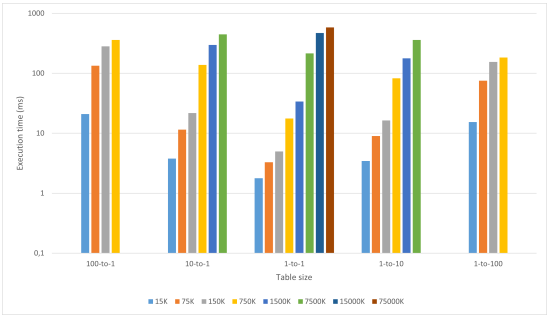
Table Size	Scale 100-1	Scale 10-1	Scale 1-1	Scale 1-10	Scale 1-100
15K	13,04	3,32	2,28	2,97	8,45
75K	64,29	7,42	3,05	5,84	36,76
150K	128,84	12,29	3,96	9,11	77,33
750K	161,60	64,55	10,11	41,83	91,48
1500K		137,63	18,16	86,00	
7500K		216,86	106,25	167,41	
15000K			230,66		
75000K			286,74		

(b) Instance 2

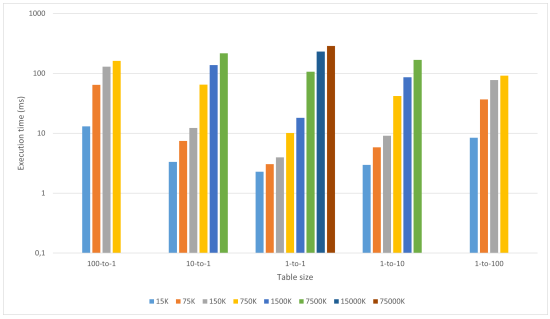
Table Size	Scale 100-1	Scale 10-1	Scale 1-1	Scale 1-10	Scale 1-100
15K	12,08	6,45	5,92	6,21	8,56
75K	35,54	9,11	6,29	7,73	21,63
150K	66,52	12,45	6,85	9,59	36,36
750K	82,37	37,26	10,55	24,50	41,73
1500K		70,05	15,24	48,17	
7500K		110,46	59,50	93,49	
15000K			121,81		
75000K			144,95		

(c) Instance 3

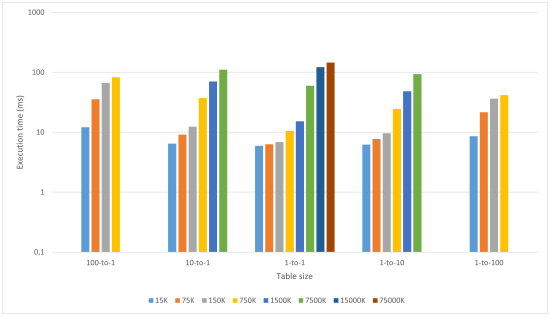
Table 4.6: Execution times in ms for join (CPU)



(a) Instance 1



(b) Instance 2



(c) Instance 3

Figure 4.6: Execution times for join (CPU)

For a more detailed understanding of which phase of the join operation consumes the most time, execution times are broken down for each phase in the 1:1 proportion case in Table 4.7. It's evident from the table that the build phase considerably exceed the probe phase in execution time.

Table Size	Build Phase	Probe Phase
15K	1,61	0,16
75K	2,63	0,65
150K	3,67	1,28
750K	11,00	6,68
1500K	20,24	13,68
7500K	134,80	80,53
15000K	278,15	193,49
75000K	350,07	233,89

(a) Instance 1

Table Size	Build Phase	Probe Phase
15K	2,17	0,11
75K	2,66	0,39
150K	3,22	0,74
750K	7,04	3,07
1500K	11,71	6,45
7500K	62,41	43,84
15000K	134,74	95,92
75000K	167,37	119,37

(b) Instance 2

Table Size	Build Phase	Probe Phase
15K	5,69	0,22
75K	5,88	0,41
150K	6,25	0,60
750K	8,68	1,87
1500K	11,75	3,50
7500K	35,15	24,34
15000K	66,59	55,22
75000K	81,18	63,76

(c) Instance 3

Table 4.7: Build and Probe phase comparison in ms (CPU)

The comprehensive results detailing the total execution times on the GPU, mirroring the tests conducted on the CPU, are compiled in Table 4.8 and shown in Figure 4.7. In contrast to the CPU test observations, the execution times on the GPU is not clearly larger when the probe table size exceed the build table size. In fact the execution times does not seem to follow a clear pattern between the different table ratios.

Table Size	Scale 100-1	Scale 10-1	Scale 1-1	Scale 1-10	Scale 1-100
15K	12,72	2,13	0,76	2,51	17,29
75K	58,12	7,12	1,87	7,68	79,43
150K	108,79	13,48	2,83	13,85	152,52
750K	150,75	61,97	9,67	63,32	162,44
1500K		120,32	19,36	123,94	
7500K		181,04	92,17	179,98	
15000K			190,54		
75000K			229,69		

(a) Instance 1

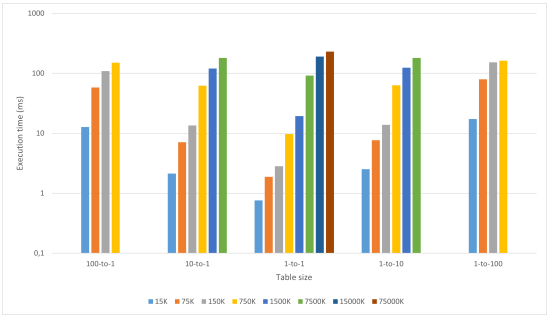
	Scale 100-1	Scale 10-1	Scale 1-1	Scale 1-10	Scale 1-100
15K	7,90	1,26	0,60	1,41	7,75
75K	37,13	4,37	1,31	4,31	36,28
150K	73,55	8,40	2,07	7,54	71,35
750K	92,71	39,18	6,61	35,01	77,93
1500K		78,31	12,82	72,35	
7500K		119,60	64,86	105,12	
15000K			132,76		
75000K			153,31		

(b) Instance 2

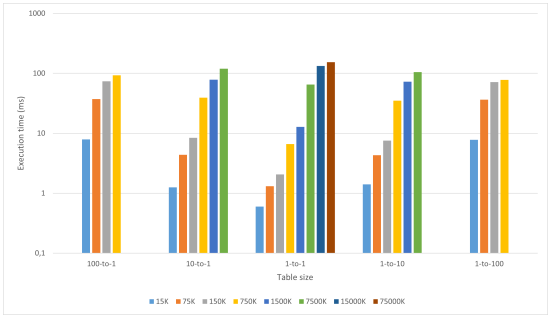
	Scale 100-1	Scale 10-1	Scale 1-1	Scale 1-10	Scale 1-100
15K	2,83	0,83	0,48	0,98	3,27
75K	11,57	1,93	0,89	2,10	14,71
150K	22,22	3,15	1,25	3,31	26,75
750K	28,28	12,49	2,81	14,40	28,72
1500K		24,19	4,92	27,99	
7500K		40,00	24,24	38,68	
15000K			47,43		
75000K			54,21		

(c) Instance 3

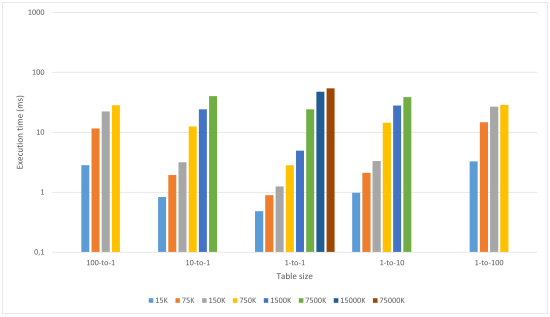
Table 4.8: Execution times in ms for join (GPU)



(a) Instance 1



(b) Instance 2



(c) Instance 3

Figure 4.7: Execution times for join (GPU)

The time taken for the overhead of transferring required data to and from the GPU before and after the execution is heavily impacted by the size of the tables. This impact could be the reason for the more scattered execution times seen on the GPU. For execution on CPU, larger tables means simply larger compute complexity which can often scale in an efficient way. For the GPU however, larger tables means larger transferring times. For a more detailed view, the execution times for the different phases plus the required initial overhead for the 1:1 proportion case can be seen in 4.9.

Table Size	Build Phase	Probe Phase
15K	0,55	0,14
75K	1,14	0,45
150K	1,57	0,71
750K	5,91	1,94
1500K	12,09	3,81
7500K	57,21	17,72
15000K	114,66	35,75
75000K	142,01	45,48

(a) Instance 1

Table Size	Build Phase	Probe Phase
15K	0,45	0,09
75K	0,67	0,34
150K	1,12	0,42
750K	3,62	1,31
1500K	7,30	2,33
7500K	35,94	12,32
15000K	71,60	24,91
75000K	88,44	26,84

(b) Instance 2

Table Size	Build Phase	Probe Phase
15K	0,39	0,05
75K	0,52	0,21
150K	0,72	0,24
750K	1,38	0,64
1500K	2,42	0,94
7500K	10,91	4,64
15000K	21,33	9,17
75000K	26,44	10,13

(c) Instance 3

Table 4.9: Build and Probe phase comparison in ms (GPU)

As seen in the table, the build phase takes significantly longer time to execute. As the size of the index array increase with the size of the probe table however, longer transfer time is needed to transfer the index array back which could contribute to longer execution times for the operation as the probe table increases in size. The amount of overhead to transfer the join attributes of the build and probe table is also dependant on the size of the tables which have an effect to equalize the execution times between the different scaling ratios.

When comparing Table 4.6 and Table 4.8 we can see that execution times for the GPU to execute is almost always better than the CPU for the same instance. When comparing different instances however there is no evidence for the GPU being superior in performance.

As the join operation is executed in two separate phases, it is possible for a heterogeneous execution plan. This could be done by executing the build phase on the GPU and the probe phase on the CPU. This does however comes with a bit of overhead for the GPU since the hash table created in the build phase has to be transferred back to the host after execution. Likewise if the opposite execution is chosen, executing the build phase on CPU and the probe phase on GPU, the index array of matching rows needs to be transferred back to the host after execution. The time taken for each phase including all required overhead can be seen in Table 4.10 and is the time to consider when considering a heterogeneous execution.

Table Size	Build Phase	Probe Phase
15K	0.78	0.21
75K	1.98	0.73
150K	3.08	1.26
750K	12.67	3.76
1500K	25.40	7.27
7500K	138.81	34.96
15000K	273.24	75.88
75000K	334.77	87.67

(a) Instance 1

Table Size	Build Phase	Probe Phase
15K	0.69	0.15
75K	1.51	0.64
150K	2.55	0.95
750K	10.26	2.99
1500K	21.01	5.52
7500K	109.64	28.93
15000K	219.65	61.16
75000K	270.26	64.88

(b) Instance 2

Table Size	Build Phase	Probe Phase
15K	0.52	0.09
75K	0.96	0.37
150K	1.42	0.53
750K	4.49	1.43
1500K	9.46	2.50
7500K	44.51	13.33
15000K	88.12	26.10
75000K	108.08	27.78

(c) Instance 3

Table 4.10: Build and Probe phase comparison in ms, including overhead (GPU)

5

Discussion

In this chapter, we discuss the experiments conducted, their results, and the implications of these findings. By implementing the aforementioned operations on both CPU and GPU platforms, we have attempted to quantify and compare their performances to provide a better understanding of their capabilities. The objective of our analysis is not to establish the absolute superiority of one processing unit over the other, but rather to gain insights into the situational advantages and drawbacks of each. The goal is to understand the scenarios where CPUs might be more efficient and others where the use of GPUs can be beneficial. The impact different factors have on the performance is also discussed and some general guidelines for the creation of an execution plan is given. We also discuss the possibility to combine the computational power of each device for a single operation.

5.1 V-Search

For the V-Search operation, we explore several factors that significantly impact the performance. These factors include the length of the word being searched for, the batch size used during execution, and the size of the database. Figure 4.1 and 4.2 illustrates the pronounced effect of word length on the CPU's performance, whereas the impact is less distinct on the GPU.

Particularly on the CPU's, where it at worst takes three times longer to search for a long word compared to a short one. It also demonstrates a linear relationship between word length and execution time, suggesting a time complexity of $\mathcal{O}(n)$. Which is also demonstrated by comparing the time is takes for random and middle term length. On the GPU, the difference between various word lengths is not as substantial as on the CPU, but a similar trend can still be observed, resulting in a similar time complexity.

Furthermore, when executing the operation in batches on the GPU, we observe a significant performance impact. Figure 4.4 reveals that all instances eventually outperform the CPUs. However, there is a trade-off associated with batch processing. As this approach increase the throughput of searches performed, depending on the use case of the operation, it could increase the latency, as earlier searches is returned at the same time as later searches in the search batch. Additionally, the transfer overhead between the CPU and GPU increases as the batch size grows, introducing

additional considerations when determining the efficiency of running the operation on the GPU.

The size of the database must also be taken into account for the GPU. Although the result indicate that the execution of the operation scales remarkably well on both CPU and GPU, with minimal differences observed, the entire database must still fit into memory. The amount of memory available on GPU is often less than that is available for the CPU, resulting in a limit of the scalability offered by GPU. When considering an in-memory database it is also crucial to consider the frequency of updates to the database as this then needs to be applied to the GPU memory as well.

5.2 Fuzzy search

The findings discussed in Chapter 4 indicate that the superiority of CPUs or GPUs in conducting fuzzy search is context-dependent, and primarily influenced by the size of the database. For the hardware examined in this study, a breakpoint at a database size of around 10,000 entries seemed to dictate the CPU and GPU performance.

In circumstances where the database size was less than 10,000, the CPU outpaced the GPU. This can be attributed to the overhead requirement of the GPU before execution, which resulted in a slower operation. Conversely, for database sizes exceeding 10,000, the GPU demonstrated superior execution, leveraging its parallel structure to handle large volumes of data more effectively.

The length of the search terms also plays a critical role in influencing the performance of the CPU and GPU. The impact is more pronounced for CPUs due to the complexity involved in computing the Levenshtein distance between two terms, which directly correlates with the length of the terms. In contrast, the performance impact of varying term lengths on the GPU is less noticeable. Given the GPU's capability to execute multiple threads simultaneously in warps, the effect of term length becomes more diluted since the performance is also dependant on the length of the terms of the other threads in the warp. As the database size expands, the GPU concurrently executes an increasing number of threads in parallel. This parallel processing inherently minimizes the influence of term length on performance as the database size escalates.

Moving on to the impact of hardware capabilities, the findings from Chapter 4 reveal that CPU performance did not correspond directly to the compute power of the instances. Interestingly, Instance 2, not the most powerful, achieved the highest performance among the three instances. Given that OpenMP was not employed for fuzzy search due to its underwhelming performance, the operation is significantly influenced by memory constraints. Instance 2's superior performance, in spite of its less powerful computing capacity, can be attributed to more efficient cache and memory management. This is particularly noteworthy considering this CPU had the lowest clock speed among the three instances. Additionally, the increased number of cores in Instance 3 did not offer any advantage in this context since the operation was executed with a single core.

Unlike CPU execution, GPU execution displays a straightforward correlation between performance and compute power - the higher the compute power, the better the performance, as exemplified by the most powerful GPU instance delivering the fastest execution time. This relationship further illustrates the GPU's superior scaling efficiency over the CPU.

The flexibility inherent in GPU operations, due to the ease of subdividing the search space, also facilitates enhanced scalability. This capacity permits the leveraging of multiple devices, including various GPUs or CPUs, to maximize performance potential. Alternatively, a heterogeneous execution strategy can be adopted to optimize the utilization of all available hardware resources.

5.3 Join

In terms of overall performance, our findings suggest that neither the CPU nor the GPU can be declared universally better for executing the join operation. The efficacy of each hardware device largely depends on the context and specific parameters of the operation. For instance, while the GPU generally outperforms the CPU for the same instances, when comparing the result between instances the result differ extensively.

One of the key impacting factors is the overhead associated with transferring data to and from the GPU both before and after the execution. The efficient way of only transferring the join columns of the tables does however offer increased efficiency as quite a lot more data would otherwise be required to be transferred back and forth to the GPU.

Another influential factor affecting performance is the size of the tables and the scaling ratio between them. It's intuitively understood that larger tables will require longer execution times. However, when assessing performance, it's crucial to consider the ratio between the tables, as this factor greatly impacts the duration required to complete different phases of operation. In situations where one table is significantly larger than the other, the optimal strategy is to leverage the device that exhibits superior performance in the most time-consuming phase. This approach ensures efficiency by capitalizing on the strengths of the specific hardware in dealing with the most challenging aspects of the task.

The duration required to carry out the indexing process, whether it is executed on a CPU or GPU, is frequently outstripped by the length of time needed to merge the tables. Memory becomes a pivotal element at this juncture, directly impacting the performance metrics of the entire operation. However, the contribution of this table merging operation to the overall performance comparison between the CPU and GPU should not be overstated. The reason for this is that the time needed for this part of the process remains consistent, irrespective of whether the task is being carried out on a CPU or a GPU. Therefore, while the operation is a constant factor in our performance equation, it doesn't necessarily make a substantial difference when determining which processing unit delivers optimal performance. It's also important to remember that the context in which the operation is utilized, some context might

not require the creation of a new table and could instead only require indexes to retrieve the data from the rows. In this case the time taken to create a new table can be ignored.

The performance difference between the build and probe phases of the hash join operation was also notable. The GPU showed more significant improvement during the probe phase due to its parallel processing capabilities. As for the build phase however there is not a great advantage of utilizing the GPU due to high overhead costs. When considering only the execution times for the build phase the GPU shows good results but the overhead is quite large.

Given the performance discrepancies between CPUs and GPUs in different phases of the join operation, it is worth considering a hybrid approach. In this model, the build phase could be executed on the CPU to minimize the effects of GPU memory latency, and the probe phase could be run on the GPU to take advantage of its parallel processing capabilities. However, the efficiency of such an approach would depend on several factors for each hardware configuration, including the overhead of transferring data between the CPU and GPU, and this presents an interesting avenue for future research.

In conclusion, while GPUs can offer significant performance improvements for join operations, there are several factors, including data transfer overhead and the specific characteristics of the operation phases, which can impact their efficiency. The findings suggest that a nuanced approach, potentially involving a hybrid model, may be beneficial for optimizing the performance of the operation depending on the hardware configuration.

6

Conclusion

The primary objective of this thesis was to assess the potential of GPU-accelerated computing for database operations, with a particular focus on the three selected operations and an analysis of their performance and scalability. This objective has been successfully achieved. The insight derived from this investigation suggests a definite potential for GPU acceleration in these operations, albeit not universally across all scenarios.

To gain a performance increase by employing GPU-utilization for V-Search, the operation is required to be performed on an in-memory database and in significant batches. This is due to the considerable overhead incurred during data transfer to and from the GPU before and after operation execution. Similarly, for the fuzzy search to be optimally accelerated by the GPU, a significant search space size is necessary so that the computational aspect of the operation can offset the initial overhead associated with GPU usage. In the case of an in-memory database, the required search space size could be considerably smaller, as the overhead would reduce significantly. For the join operation, data transfer overhead to the GPU can be minimized by only transferring the necessary join columns from both tables, thereby making the parallel architecture of the GPU more beneficial compared to the CPU, even considering the overhead of data transfer before execution and the subsequent return of the index array.

While GPUs can provide performance enhancements for these operations, their effectiveness is always constrained by the data transfer overhead required and the degree of parallelism in the operation. These factors may preclude many database operations from experiencing any performance increase with GPU usage. In terms of scalability, GPUs generally perform commendably due to their parallel architecture. However, their performance is limited by their memory, which is less flexible to configure compared to conventional memory.

6.1 Future work

While this thesis provides substantial insights into the potential of GPU accelerated database operations, several avenues for future research still exists.

6.1.1 Wider Range of Database Operations

While the scope of this thesis was confined to the operations string search, fuzzy search, and join, future research could incorporate a broader array of operations like sorting and graph traversal. These operations could potentially benefit from GPU acceleration, thereby expanding our understanding of GPU-accelerated database operations further.

6.1.2 Implementation evaluation

As the operations in this thesis were executed using a single implementation, an evaluation of various implementations could provide valuable insights. Such an assessment could aid in enhancing performance benefits by enabling more optimized use of the GPU.

6.1.3 Execution planning

For optimal utilization of GPU acceleration in a Database Management System (DBMS), a well-defined execution plan is essential. Therefore, further research is needed to determine more accurately the conditions under which it's advantageous to use the CPU versus the GPU for an operation. These conditions would depend on several factors such as workload, data complexity, and specific database and hardware configurations.

Bibliography

[1] Statista, *Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2020, with forecasts from 2021 to 2025*, 2021. [Online]. Available: https://www.statista.com/statistics/871513/worldwide-data-created/ (visited on 03/09/2023).

[2] P. Du, R. Weber, P. Luszczek, S. Tomov, G. Peterson, and J. Dongarra, "From cuda to opencl: Towards a performance-portable solution for multi-platform gpu programming," *Parallel Computing*, vol. 38, no. 8, pp. 391–407, 2012, APPLICATION ACCELERATORS IN HPC, ISSN: 0167-8191. DOI: https://doi.org/10.1016/j.parco.2011.10.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167819111001335.

[3] HEAVY.AI, *Heavydb*. [Online]. Available: https://www.heavy.ai/ (visited on 03/01/2023).

[4] SQream, *Sqreamdb*. [Online]. Available: https://sqream.com/product/sqreamdb/ (visited on 03/01/2023).

[5] brytlyt, *Brytlytdb*. [Online]. Available: https://brytlyt.io/database/ (visited on 03/01/2023).

[6] H. Inc, *Pg-strom*. [Online]. Available: https://en.heterodb.com/ (visited on 03/01/2023).

[7] D. Dojchinovski, M. Gusev, and V. Zdraveski, "Efficiently running sql queries on gpu," in *2018 26th Telecommunications Forum (TELFOR)*, 2018, pp. 1–4. DOI: 10.1109/TELFOR.2018.8611821.

[8] P. Bakkum and K. Skadron, "Accelerating sql database operations on a gpu with cuda," in *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, ser. GPGPU-3, Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 2010, pp. 94–103, ISBN: 9781605589350. DOI: 10.1145/1735688.1735706. [Online]. Available: https://doi.org/10.1145/1735688.1735706.

[9] B. Grandhi, S. Chickerur, and M. S. Patil, "Performance analysis of mysql, apache spark on cpu and gpu," in *2018 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, 2018, pp. 1494–1499. DOI: 10.1109/RTEICT42901.2018.9012459.

[10] S. S. M. Al-Dabbagh and Y. M. Abdal, "Parallel hybrid string matching algorithm using cuda api function," in *2021 International Conference on Computing and Communications Applications and Technologies (I3CAT)*, 2021, pp. 66–70. DOI: 10.1109/I3CAT53310.2021.9629415.

[11] Y. Mitani, F. Ino, and K. Hagihara, "Parallelizing exact and approximate string matching via inclusive scan on a gpu," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 7, pp. 1989–2002, 2017. DOI: 10.1109/TPDS.2016.2645222.

[12] C.-H. Lin and C.-C. Huang, "High-performance parallel location-aware algorithms for approximate string matching on gpus," in *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*, 2015, pp. 570–575. DOI: 10.1109/ICPADS.2015.77.

[13] K. Wang, K. Zhang, Y. Yuan, *et al.*, "Concurrent analytical query processing with gpus," *Proc. VLDB Endow.*, vol. 7, no. 11, pp. 1011–1022, Jul. 2014, ISSN: 2150-8097. DOI: 10.14778/2732967.2732976. [Online]. Available: https://doi.org/10.14778/2732967.2732976.

[14] B. He, M. Lu, K. Yang, *et al.*, "Relational query coprocessing on graphics processors," *ACM Trans. Database Syst.*, vol. 34, no. 4, Dec. 2009, ISSN: 0362-5915. DOI: 10.1145/1620585.1620588. [Online]. Available: https://doi.org/10.1145/1620585.1620588.

[15] A. Burnes, "Nvidia ampere ga102 gpu architecture," NVIDIA Corporation, whitepaper, Sep. 2020.

[16] "Cuda c++ programming guide, release 12.1," NVIDIA Corporation & Affiliates, whitepaper, Apr. 2023.

[17] "Openmp application programming interface specification, version 5.2," OpenMP, whitepaper, Nov. 2021.

[18] A. W. Services, *Amazon ec2*. [Online]. Available: https://aws.amazon.com/ec2/ (visited on 07/03/2023).

[19] V. I. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions and Reversals," *Soviet Physics Doklady*, vol. 10, p. 707, Feb. 1966.

[20] K. Balhaf, M. A. Alsmirat, M. Al-Ayyoub, Y. Jararweh, and M. A. Shehab, "Accelerating levenshtein and damerau edit distance algorithms using gpu with unified memory," in *2017 8th International Conference on Information and Communication Systems (ICICS)*, 2017, pp. 7–11. DOI: 10.1109/IACS.2017.7921937.

[21] S. Soroushnia, M. Daneshtalab, T. Pahikkala, and J. Plosila, "Parallel implementation of fuzzified pattern matching algorithm on gpu," in *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, 2015, pp. 341–344. DOI: 10.1109/PDP.2015.75.

[22] K. Balhaf, M. A. Shehab, W. T. Al-Sarayrah, M. Al-Ayyoub, M. Al-Saleh, and Y. Jararweh, "Using gpus to speed-up levenshtein edit distance computation," in *2016 7th International Conference on Information and Communication Systems (ICICS)*, 2016, pp. 80–84. DOI: 10.1109/IACS.2016.7476090.

[23] N. Devarajan, S. Navneeth, and S. Mohanavalli, "Gpu accelerated relational hash join operation," in *2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2013, pp. 891–896. DOI: 10.1109/ICACCI.2013.6637294.

[24] R. Rui, H. Li, and Y.-C. Tu, "Join algorithms on gpus: A revisit after seven years," in *2015 IEEE International Conference on Big Data (Big Data)*, 2015, pp. 2541–2550. DOI: 10.1109/BigData.2015.7364051.

[25] T. P. P. C. (TPC), “Tpc benchmark h standard specification revision 3.0.1,” TPC, whitepaper, 2022.

A

Appendix 1 - V-Search

In this first part of the appendix is the complete set of result data generated for the V-Search operation. Executions times for all four test datasets with different term lengths is presented for both CPU and GPU with the different database sizes. For GPU the time taken to transfer the batches is also included.

A.1 Instance 1

	Random Terms	**Short Terms**	**Middle Terms**	**Long Terms**
100K	2,14	0,92	2,03	3,29
1M	2,49	0,99	2,29	3,80
10M	2,14	0,82	1,92	3,13
100M	2,64	1,07	2,30	3,76

Table A.1: V-Search Execution Times in ms (Instance 1 - CPU)

	Random Terms	**Short Terms**	**Middle Terms**	**Long Terms**	**Transfer Time**
1	1980.34	1524.60	1934.65	2341.74	144.21
5	561.44	381.79	489.48	586.06	142.11
10	311.38	208.58	269.99	324.35	142.23
50	76.68	50.70	66.18	80.41	143.32
100	39.00	25.94	34.40	41.86	142.71
500	12.41	7.77	10.46	13.50	142.51
1000	7.05	4.71	6.64	7.70	141.54
5000	1.94	1.23	1.55	2.23	142.09
10000	1.83	1.40	1.67	1.90	141.89

Table A.2: V-Search Execution Times (ms) with 100K Database (Instance 1 - GPU)

	Random Terms	Short Terms	Middle Terms	Long Terms	Transfer Time
1	2066.62	1602.60	1985.05	2401.35	148.00
5	596.55	406.85	525.74	618.33	146.67
10	337.68	227.78	292.24	348.46	146.68
50	83.05	55.60	73.36	88.47	146.02
100	41.97	29.55	37.50	45.25	146.80
500	11.91	7.77	10.83	12.43	146.36
1000	7.68	4.81	6.19	7.80	146.89
5000	2.16	1.39	1.87	2.34	146.49
10000	1.88	1.45	1.72	1.97	146.93

Table A.3: V-Search Execution Times (ms) with 1M Database (Instance 1 - GPU)

	Random Terms	Short Terms	Middle Terms	Long Terms	Transfer Time
1	2154.57	1694.93	2056.23	2454.74	192.38
5	610.13	429.43	542.38	640.28	189.88
10	345.05	240.03	301.66	354.41	189.65
50	84.40	58.71	75.75	88.63	188.93
100	41.42	30.55	39.04	44.31	189.60
500	10.13	6.85	8.87	10.82	189.26
1000	5.29	3.53	4.93	5.80	189.79
5000	1.96	1.45	1.82	2.36	192.06
10000	2.17	1.63	1.96	2.29	194.48

Table A.4: V-Search Execution Times (ms) with 10M Database (Instance 1 - GPU)

	Random Terms	Short Terms	Middle Terms	Long Terms	Transfer Time
1	2249.07	1793.69	2126.41	2535.60	645.11
5	646.05	455.79	560.58	663.29	652.76
10	361.49	247.99	318.09	372.26	649.96
50	91.66	60.27	79.45	92.77	643.68
100	47.57	33.94	42.52	52.55	647.98
500	10.96	7.46	9.37	12.62	649.80
1000	5.59	3.80	5.01	6.63	645.35
5000	2.03	1.21	1.84	2.85	649.18
10000	2.08	1.39	1.93	2.87	649.82

Table A.5: V-Search Execution Times (ms) with 100M Database (Instance 1 - GPU)

A.2 Instance 2

	Random Terms	Short Terms	Middle Terms	Long Terms
100K	2.26	0.88	2.18	3.57
1M	2.31	0.91	2.17	3.56
10M	2.39	0.81	2.19	3.61
100M	2.52	0.84	2.24	3.67

Table A.6: V-Search Execution Times in ms (Instance 2 - CPU)

	Random Terms	**Short Terms**	**Middle Terms**	**Long Terms**	**Transfer Time**
1	1030.28	901.59	1040.43	1202.63	170.84
5	259.93	195.10	233.34	273.89	152.10
10	143.82	101.83	125.85	148.15	149.73
50	34.85	24.79	31.61	36.60	148.64
100	27.04	18.48	24.12	27.19	149.76
500	9.39	5.55	7.45	9.16	150.50
1000	4.82	3.11	3.88	5.13	150.05
5000	1.38	0.99	1.08	1.22	150.49
10000	1.62	1.21	1.43	1.66	150.31

Table A.7: V-Search Execution Times (ms) with 100K Database (Instance 2 - GPU)

	Random Terms	**Short Terms**	**Middle Terms**	**Long Terms**	**Transfer Time**
1	1056.46	920.86	1061.41	1227.69	161.69
5	270.64	202.42	248.68	288.43	161.68
10	151.16	108.02	133.80	155.09	159.28
50	36.58	25.96	34.41	38.52	158.36
100	24.36	18.45	23.42	28.57	158.79
500	9.05	5.87	8.56	10.31	158.67
1000	5.23	3.76	3.81	5.84	158.39
5000	1.52	1.22	1.10	1.30	159.31
10000	1.65	1.28	1.52	1.72	158.07

Table A.8: V-Search Execution Times (ms) with 10M Database (Instance 2 - GPU)

	Random Terms	**Short Terms**	**Middle Terms**	**Long Terms**	**Transfer Time**
1	1081.86	944.54	1079.36	1245.39	236.16
5	273.51	210.19	252.05	290.30	231.92
10	163.41	111.74	136.25	154.53	230.63
50	36.68	33.02	37.62	42.36	228.56
100	33.92	23.38	29.02	30.97	228.96
500	10.08	5.91	9.59	10.73	232.66
1000	5.63	3.71	4.58	6.11	234.39
5000	1.61	1.11	1.29	1.67	230.70
10000	1.69	1.28	1.54	1.76	229.09

Table A.9: V-Search Execution Times (ms) with 10M Database (Instance 2 - GPU)

	Random Terms	Short Terms	Middle Terms	Long Terms	Transfer Time
1	1117.46	975.93	1105.01	1276.58	1122.40
5	279.80	217.81	262.93	299.44	1121.74
10	154.79	115.36	140.03	161.43	1123.48
50	35.62	26.62	32.01	37.42	1122.37
100	19.84	14.42	17.40	20.50	1122.18
500	5.18	3.58	5.10	5.55	1122.60
1000	2.77	2.18	2.42	2.84	1123.68
5000	1.01	0.80	0.83	1.10	1124.44
10000	1.34	0.98	1.29	1.39	1123.36

Table A.10: V-Search Execution Times (ms) with 100M Database (Instance 2 - GPU)

A.3 Instance 3

	Random Terms	Short Terms	Middle Terms	Long Terms
100K	2.35	0.81	2.22	3.74
1M	2.40	0.86	2.21	3.73
10M	2.47	0.84	2.20	3.74
100M	2.56	0.87	2.20	3.76

Table A.11: V-Search Execution Times in ms (Instance 3 - CPU)

	Random Terms	Short Terms	Middle Terms	Long Terms	Transfer Time
1	1398.16	1351.12	1426.88	1514.30	168.16
5	296.58	265.97	289.79	317.97	135.61
10	157.90	135.20	148.47	163.80	134.82
50	34.68	28.00	31.12	36.26	133.32
100	18.10	14.19	17.20	19.11	132.60
500	4.65	3.51	4.04	4.58	137.45
1000	2.44	2.06	2.39	2.45	128.41
5000	0.91	1.04	0.80	0.92	128.87
10000	1.13	0.91	0.98	1.07	126.85

Table A.12: V-Search Execution Times (ms) with 100K Database (Instance 3 - GPU)

	Random Terms	Short Terms	Middle Terms	Long Terms	Transfer Time
1	1410.88	1360.83	1438.60	1529.82	144.49
5	306.99	274.59	300.17	326.37	136.32
10	163.41	138.07	154.19	173.97	132.70
50	36.05	28.80	34.74	39.33	132.82
100	19.71	15.06	17.97	20.43	132.09
500	5.04	3.67	4.56	5.19	134.13
1000	2.54	2.02	2.10	2.94	136.52
5000	1.11	0.84	0.76	0.93	133.74
10000	1.15	0.99	1.01	1.11	132.10

Table A.13: V-Search Execution Times (ms) with 1M Database (Instance 3 - GPU)

	Random Terms	**Short Terms**	**Middle Terms**	**Long Terms**	**Transfer Time**
1	1421.75	1367.24	1449.48	1540.10	193.64
5	310.70	274.97	302.43	331.80	160.19
10	163.52	141.36	156.54	174.74	175.73
50	37.89	29.90	34.47	39.83	178.01
100	19.63	15.13	19.06	22.64	165.55
500	5.24	3.73	4.22	5.42	157.08
1000	2.58	1.94	2.45	2.42	159.38
5000	1.03	0.76	0.76	0.95	158.41
10000	1.14	0.81	1.02	1.15	158.21

Table A.14: V-Search Execution Times (ms) with 10M Database (Instance 3 - GPU)

	Random Terms	**Short Terms**	**Middle Terms**	**Long Terms**	**Transfer Time**
1	1436.32	1381.56	1456.60	1557.12	618.18
5	317.57	277.61	306.24	333.07	616.68
10	169.98	140.99	157.43	177.74	623.45
50	38.54	30.20	36.56	41.36	611.15
100	19.99	15.83	18.77	22.21	610.66
500	5.53	3.81	4.58	5.41	617.99
1000	2.66	2.25	2.59	2.78	622.65
5000	1.11	0.72	0.89	1.02	613.74
10000	1.10	0.82	1.04	1.16	621.68

Table A.15: V-Search Execution Times (ms) with 100M Database (Instance 3 - GPU)

B

Appendix - Fuzzy Search

In this second part of the appendix is the complete set of result data generated for the fuzzy search operation on both CPU and GPU. Times are given for all tested combinations of database sizes and test datasets, including the time taken return the result for GPU.

B.1 Instance 1

	Random Terms	**Short Terms**	**Middle Terms**	**Long Terms**
1K	25.83	11.47	21.00	34.55
10K	229.18	117.06	211.16	342.80
100K	2341.63	1170.09	2135.09	3529.76

Table B.1: Fuzzy Search Execution Times in ms (Instance 1 - CPU)

	Random Terms	**Short Terms**	**Middle Terms**	**Long Terms**	**Transfer Time**
1K	3.59	1.63	2.93	4.76	146.35
10K	20.04	12.33	18.93	27.59	146.21
100K	49.84	44.06	49.70	55.55	147.57
1M	355.79	302.13	352.58	415.34	152.32
10M	2906.49	2514.35	2782.73	3336.39	166.69

Table B.2: Fuzzy Search Execution Times in ms (Instance 1 - GPU)

B.2 Instance 2

	Random Terms	**Short Terms**	**Middle Terms**	**Long Terms**
1K	21.16	9.60	17.50	28.72
10K	190.99	98.64	176.23	285.48
100K	1952.95	986.56	1780.96	2936.19

Table B.3: Fuzzy Search Execution Times in ms (Instance 2 - CPU)

	Random Terms	Short Terms	Middle Terms	Long Terms	Transfer Time
1K	2.40	1.19	1.99	3.10	162.18
10K	13.02	7.99	12.19	17.96	151.05
100K	31.72	28.20	30.55	37.57	153.43
1M	223.19	183.27	209.63	271.65	156.82
10M	2071.13	1701.49	1937.32	2499.67	182.24

Table B.4: Fuzzy Search Execution Times in ms (Instance 2 - GPU)

B.3 Instance 3

	Random Terms	Short Terms	Middle Terms	Long Terms
1K	26.67	12.33	22.18	36.24
10K	241.85	125.98	222.97	359.55
100K	2474.57	1261.90	2261.35	3700.77

Table B.5: Fuzzy Search Execution Times in ms (Instance 3 - CPU)

	Random Terms	Short Terms	Middle Terms	Long Terms	Transfer Time
1K	2.26	1.08	1.86	2.99	151.89
10K	12.52	7.73	11.80	17.60	135.99
100K	29.65	26.94	28.17	34.45	139.34
1M	167.05	158.32	168.58	173.29	141.36
10M	1297.18	1299.37	1266.30	1343.33	155.64

Table B.6: Fuzzy Search Execution Times in ms (Instance 3 - GPU)

C

Appendix - Join

In this last part of the appendix is the complete set of result data generated for the join operation for both CPU and GPU. For the CPU the times given is the time taken to execute the build phase, probe phase, the sum of theses two phases, and the time taken to construct a result table. For the GPU more times are given and include the time taken for the required overhead for the build phase, overhead for the probe phase, execution time of the build phase, time taken to transfer the hash table back to memory after the build phase, execution time of the probe phase, the sum of build and probe phase, time taken to return the index array after the probe phase, and the time taken to construct a result table.

C.1 Instance 1

	Build	**Probe**	**Execution**	**Transfer**
15K	20.69	0.19	20.88	3.50
75K	132.55	0.80	133.35	18.37
15K	280.94	1.62	282.56	48.97
75K	351.38	7.10	358.48	256.90

Table C.1: Join Execution Times (ms) for 100:1 ratio (Instance 1 - CPU)

	Build	**Probe**	**Execution**	**Transfer**
15K	3.61	0.16	3.78	3.22
75K	10.84	0.68	11.52	18.30
15K	20.34	1.39	21.72	49.06
75K	131.10	6.96	138.07	260.93
15K	283.25	14.25	297.50	545.16
75K	365.71	78.90	444.61	3044.50

Table C.2: Join Execution Times (ms) for 10:1 ratio (Instance 1 - CPU)

	Build	Probe	Execution	Transfer
15K	1.61	0.16	1.77	3.21
75K	2.63	0.65	3.28	18.19
15K	3.67	1.28	4.95	48.79
75K	11.00	6.68	17.68	258.45
15K	20.24	13.68	33.92	533.23
75K	134.80	80.53	215.33	3082.59
15K	278.15	193.49	471.64	6197.75
75K	350.07	233.89	583.96	6964.57

Table C.3: Join Execution Times (ms) for 1:1 ratio (Instance 1 - CPU)

	Build	Probe	Execution	Transfer
15K	1.75	1.70	3.45	50.25
75K	2.67	6.29	8.95	259.22
15K	3.69	12.60	16.29	535.04
75K	11.03	71.17	82.21	3087.65
15K	20.76	156.64	177.40	6366.25
75K	137.39	221.58	358.98	6959.92

Table C.4: Join Execution Times (ms) for 1:10 ratio (Instance 1 - CPU)

	Build	Probe	Execution	Transfer
15K	1.72	13.62	15.34	418.86
75K	2.68	72.65	75.33	2074.27
15K	3.71	151.46	155.17	4267.82
75K	12.80	170.47	183.27	4379.52

Table C.5: Join Execution Times (ms) for 1:100 ratio (Instance 1 - CPU)

	Build Overhead	Probe Overhead	Build Execution	Hash Table	Probe Execution	Total Execution	Index overhead	Result Table Creation
15K	1.49	0.41	10.41	13.60	0.32	10.74	0.09	3.05
75K	4.94	0.50	50.85	80.37	1.51	52.36	0.31	15.95
150K	9.39	0.64	95.82	156.30	2.35	98.16	0.60	45.53
750K	11.56	0.97	131.85	193.18	4.54	136.38	1.84	248.36

Table C.6: Join Execution Times (ms) for 1:100 ratio (Instance 1 - GPU)

	Build Overhead	Probe Overhead	Build Execution	Hash Table	Probe Execution	Total Execution	Index overhead	Result Table Creation
15K	0.67	0.24	1.08	1.54	0.07	1.15	0.07	2.91
75K	1.02	0.43	5.11	6.77	0.26	5.37	0.30	16.20
150K	1.48	0.47	10.45	13.47	0.49	10.94	0.60	45.05
750K	5.03	0.82	51.91	80.69	2.35	54.26	1.85	250.69
1500K	9.38	1.36	101.33	158.87	4.74	106.07	3.51	537.28
7500K	11.45	4.60	131.02	192.76	17.28	148.30	16.70	3019.48

Table C.7: Join Execution Times (ms) for 1:10 ratio (Instance 1 - GPU)

	Build Overhead	Probe Overhead	Build Execution	Hash Table	Probe Execution	Total Execution	Index overhead	Result Table Creation
15K	0.29	0.03	0.26	0.23	0.11	0.38	0.07	2.85
75K	0.63	0.26	0.52	0.84	0.18	0.70	0.28	16.13
150K	0.69	0.46	0.88	1.51	0.25	1.13	0.55	45.42
750K	1.06	0.75	4.85	6.76	1.19	6.04	1.81	247.72
1500K	1.50	1.21	10.59	13.31	2.60	13.19	3.46	526.25
7500K	5.18	4.61	52.03	81.60	13.11	65.14	17.24	3036.66
15000K	9.50	8.75	105.16	158.58	27.00	132.17	40.13	6274.02
75000K	11.76	9.33	130.25	192.76	36.15	166.40	42.19	7058.19

Table C.8: Join Execution Times (ms) for 1:1 ratio (Instance 1 - GPU)

	Build Overhead	Probe Overhead	Build Execution	Hash Table	Probe Execution	Total Execution	Index overhead	Result Table Creation
15K	0.41	0.30	0.33	0.26	0.90	1.23	0.58	46.45
75K	0.70	0.75	0.67	0.87	3.73	4.40	1.83	247.52
150K	0.75	1.19	1.13	1.53	7.31	8.44	3.48	527.87
750K	1.12	4.56	5.17	6.86	35.10	40.27	17.37	3069.03
1500K	1.76	8.55	9.93	13.63	63.99	73.92	39.72	6245.47
7500K	5.25	9.24	52.32	80.46	70.93	123.25	42.24	7045.47

Table C.9: Join Execution Times (ms) for 10:1 ratio (Instance 1 - GPU)

	Build Overhead	Probe Overhead	Build Execution	Hash Table	Probe Execution	Total Execution	Index overhead	Result Table Creation
15K	0.49	1.21	0.33	0.26	11.77	12.10	3.49	408.22
75K	0.72	4.51	0.66	0.89	56.61	57.27	16.94	2071.11
150K	0.90	8.75	1.04	1.77	101.88	102.92	39.95	4171.93
750K	1.32	8.99	5.03	6.94	105.44	110.47	41.67	4366.38

Table C.10: Join Execution Times (ms) for 100:1 ratio (Instance 1 - GPU)

C.2 Instance 2

	Build	Probe	Execution	Transfer
1	12.89	0.15	13.04	2.61
2	63.79	0.50	64.29	16.84
3	127.92	0.92	128.84	36.04
4	158.09	3.51	161.60	189.34

Table C.11: Join Execution Times (ms) for 100:1 ratio (Instance 2 - CPU)

	Build	Probe	Execution	Transfer
1	3.19	0.13	3.32	2.49
2	7.01	0.41	7.42	15.60
3	11.45	0.84	12.29	35.99
4	61.09	3.46	64.55	183.97
5	130.20	7.43	137.63	382.71
6	172.67	44.19	216.86	2225.94

Table C.12: Join Execution Times (ms) for 10:1 ratio (Instance 2 - CPU)

	Build	Probe	Execution	Transfer
1	2.17	0.11	2.28	2.49
2	2.66	0.39	3.05	15.84
3	3.22	0.74	3.96	35.86
4	7.04	3.07	10.11	183.10
5	11.71	6.45	18.16	379.68
6	62.41	43.84	106.25	2239.11
7	134.74	95.92	230.66	4554.18
8	167.37	119.37	286.74	5253.17

Table C.13: Join Execution Times (ms) for 1:1 ratio (Instance 2 - CPU)

	Build	Probe	Execution	Transfer
1	2.21	0.76	2.97	35.94
2	2.69	3.15	5.84	185.37
3	3.22	5.89	9.11	378.51
4	7.07	34.76	41.83	2211.74
5	11.29	74.71	86.00	4746.26
6	60.62	106.79	167.41	5125.87

Table C.14: Join Execution Times (ms) for 1:10 ratio (Instance 2 - CPU)

	Build	Probe	Execution	Transfer
1	2.20	6.25	8.45	314.26
2	2.68	34.09	36.76	1576.15
3	3.29	74.04	77.33	3287.99
4	7.06	84.42	91.48	3345.96

Table C.15: Join Execution Times (ms) for 1:100 ratio (Instance 2 - CPU)

	Build Overhead	Probe Overhead	Build Execution	Hash Table	Probe Execution	Total Execution	Index overhead	Result Table Creation
15K	1.86	0.35	5.50	14.27	0.11	5.61	0.08	2.35
75K	7.05	0.45	28.85	74.12	0.48	29.33	0.29	15.21
150K	13.45	0.56	58.01	148.19	0.95	58.96	0.58	33.24
750K	16.66	1.14	71.62	187.25	1.51	73.13	1.78	184.67

Table C.16: Join Execution Times (ms) for 100:1 ratio (Instance 2 - GPU)

	Build Overhead	Probe Overhead	Build Execution	Hash Table	Probe Execution	Total Execution	Index overhead	Result Table Creation
15K	0.65	0.04	0.47	1.43	0.03	0.50	0.07	2.41
75K	1.17	0.39	2.44	6.64	0.09	2.53	0.27	13.90
150K	1.76	0.47	5.49	13.46	0.16	5.64	0.53	32.51
750K	6.76	1.02	28.84	73.31	0.88	29.72	1.68	179.08
1500K	13.61	1.71	58.02	147.88	1.79	59.81	3.18	377.54
7500K	16.79	6.79	71.72	182.25	7.01	78.73	17.29	2264.03

Table C.17: Join Execution Times (ms) for 10:1 ratio (Instance 2 - GPU)

	Build Overhead	Probe Overhead	Build Execution	Hash Table	Probe Execution	Total Execution	Index overhead	Result Table Creation
15K	0.26	0.03	0.19	0.24	0.05	0.24	0.07	2.28
75K	0.39	0.27	0.28	0.84	0.07	0.35	0.30	14.72
150K	0.65	0.32	0.47	1.43	0.10	0.56	0.53	32.81
750K	1.17	0.96	2.45	6.65	0.35	2.80	1.68	179.08
1500K	1.79	1.55	5.51	13.71	0.78	6.29	3.19	373.20
7500K	7.10	6.65	28.83	73.71	5.66	34.50	16.61	2229.02
15000K	13.61	13.02	57.99	148.05	11.88	69.87	36.25	4541.29
75000K	16.79	13.63	71.65	181.82	13.21	84.86	38.03	5007.79

Table C.18: Join Execution Times (ms) for 1:1 ratio (Instance 2 - GPU)

	Build Overhead	Probe Overhead	Build Execution	Hash Table	Probe Execution	Total Execution	Index overhead	Result Table Creation
15K	0.32	0.14	0.19	0.25	0.20	0.39	0.55	32.86
75K	0.45	0.99	0.28	0.81	0.89	1.17	1.70	178.98
150K	0.66	1.54	0.46	1.40	1.67	2.13	3.21	371.31
750K	1.23	6.48	2.44	6.58	7.93	10.37	16.93	2203.98
1500K	2.02	12.78	5.49	13.91	15.97	21.46	36.09	4576.42
7500K	7.20	13.45	28.84	73.56	17.58	46.42	38.05	5184.95

Table C.19: Join Execution Times (ms) for 1:10 ratio (Instance 2 - GPU)

	Build Overhead	Probe Overhead	Build Execution	Hash Table	Probe Execution	Total Execution	Index overhead	Result Table Creation
15K	0.38	1.57	0.19	0.23	2.32	2.51	3.28	308.55
75K	0.45	6.41	0.28	0.83	11.63	11.91	17.51	1558.36
150K	0.71	12.79	0.46	1.52	21.02	21.47	36.38	3169.41
750K	1.44	13.38	2.45	6.84	22.62	25.07	38.04	3344.05

Table C.20: Join Execution Times (ms) for 1:100 ratio (Instance 2 - GPU)

C.3 Instance 3

	Build	Probe	Execution	Transfer
1	11.85	0.23	12.08	1.52
2	35.04	0.50	35.54	8.91
3	65.78	0.74	66.52	16.97
4	80.21	2.16	82.37	119.45

Table C.21: Join Execution Times (ms) for 100:1 ratio (Instance 3 - CPU)

	Build	Probe	Execution	Transfer
1	6.28	0.17	6.45	1.43
2	8.65	0.46	9.11	8.82
3	11.76	0.68	12.45	17.10
4	35.05	2.21	37.26	118.97
5	65.84	4.21	70.05	235.82
6	84.10	26.36	110.46	1309.42

Table C.22: Join Execution Times (ms) for 10:1 ratio (Instance 3 - CPU)

	Build	Probe	Execution	Transfer
1	5.69	0.22	5.92	1.46
2	5.88	0.41	6.29	8.74
3	6.25	0.60	6.85	16.79
4	8.68	1.87	10.55	120.27
5	11.75	3.50	15.24	233.14
6	35.15	24.34	59.50	1318.18
7	66.59	55.22	121.81	2856.98
8	81.18	63.76	144.95	3505.59

Table C.23: Join Execution Times (ms) for 1:1 ratio (Instance 3 - CPU)

	Build	Probe	Execution	Transfer
1	5.70	0.50	6.21	17.09
2	6.11	1.62	7.73	119.83
3	6.38	3.21	9.59	242.43
4	8.75	15.74	24.49	1311.86
5	11.85	36.32	48.17	2899.40
6	35.33	58.16	93.49	3575.67

Table C.24: Join Execution Times (ms) for 1:10 ratio (Instance 3 - CPU)

	Build	Probe	Execution	Transfer
1	5.67	2.89	8.56	163.26
2	5.96	15.67	21.63	838.31
3	6.27	30.09	36.36	1718.61
4	8.86	32.87	41.73	1914.97

Table C.25: Join Execution Times (ms) for 1:100 ratio (Instance 3 - CPU)

	Build Overhead	Probe Overhead	Build Execution	Hash Table	Probe Execution	Total Execution	Index overhead	Result Table Creation
15K	1.08	0.30	1.36	7.04	0.04	1.40	0.05	1.27
75K	3.23	0.37	7.69	33.62	0.11	7.80	0.17	8.27
150K	5.81	0.41	15.47	66.51	0.20	15.67	0.33	16.39
750K	7.24	0.59	19.25	81.66	0.33	19.58	0.86	107.98

Table C.26: Join Execution Times (ms) for 100:1 ratio (Instance 3 - CPU)

	Build Overhead	Probe Overhead	Build Execution	Hash Table	Probe Execution	Total Execution	Index overhead	Result Table Creation
15K	0.60	0.03	0.13	0.73	0.02	0.15	0.04	1.35
75K	0.82	0.33	0.60	3.32	0.03	0.63	0.16	9.38
150K	1.05	0.36	1.37	7.56	0.05	1.41	0.32	16.40
750K	3.20	0.57	7.69	33.38	0.21	7.90	0.82	108.87
1500K	5.80	0.83	15.47	66.55	0.42	15.89	1.68	231.79
7500K	7.24	2.93	19.19	81.33	1.96	21.16	8.68	1300.63

Table C.27: Join Execution Times (ms) for 10:1 ratio (Instance 3 - CPU)

	Build Overhead	Probe Overhead	Build Execution	Hash Table	Probe Execution	Total Execution	Index overhead	Result Table Creation
15K	0.31	0.02	0.08	0.13	0.03	0.11	0.04	1.29
75K	0.43	0.18	0.09	0.44	0.03	0.12	0.16	8.27
150K	0.59	0.21	0.13	0.71	0.03	0.16	0.29	16.20
750K	0.78	0.56	0.60	3.11	0.08	0.68	0.79	108.35
1500K	1.05	0.74	1.37	7.04	0.20	1.57	1.56	234.64
7500K	3.24	2.89	7.67	33.60	1.76	9.43	8.69	1304.62
15000K	5.85	5.31	15.48	66.80	3.86	19.33	16.94	2904.31
75000K	7.22	5.57	19.22	81.64	4.57	23.78	17.65	3487.03

Table C.28: Join Execution Times (ms) for 1:1 ratio (Instance 3 - CPU)

	Build Overhead	Probe Overhead	Build Execution	Hash Table	Probe Execution	Total Execution	Index overhead	Result Table Creation
15K	0.48	0.06	0.08	0.14	0.05	0.12	0.31	16.37
75K	0.49	0.53	0.09	0.43	0.18	0.27	0.81	108.22
150K	0.60	0.79	0.13	0.68	0.31	0.44	1.47	237.88
750K	0.78	2.88	0.60	3.00	1.52	2.12	8.63	1304.36
1500K	1.05	5.29	1.36	6.98	3.34	4.70	16.95	2934.50
7500K	3.24	5.66	7.69	33.45	4.46	12.15	17.63	3513.65

Table C.29: Join Execution Times (ms) for 1:10 ratio (Instance 3 - CPU)

	Build Overhead	Probe Overhead	Build Execution	Hash Table	Probe Execution	Total Execution	Index overhead	Result Table Creation
15K	0.45	0.75	0.08	0.14	0.41	0.49	1.59	162.08
75K	0.52	2.82	0.09	0.51	1.88	1.97	9.40	829.82
150K	0.62	5.40	0.13	0.70	3.53	3.66	17.07	1706.31
750K	0.84	5.51	0.60	3.07	4.20	4.79	17.58	1863.84

Table C.30: Join Execution Times (ms) for 1:100 ratio (Instance 3 - CPU)