**Department of Computer Engineering**



**Cairo University**

**Faculty of Engineering**

**Parallel Computing**

***Big Assignment***

**Database Management System**

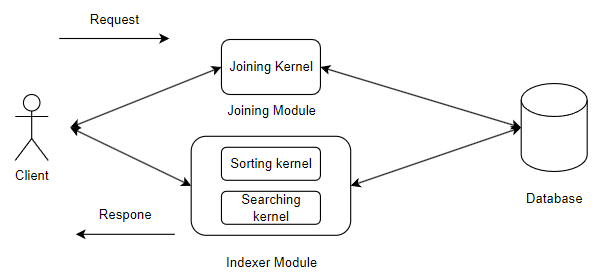
|  |  |  |  |
| --- | --- | --- | --- |
| **Team Members** | | | |
| **Name** | **Sec** | **BN** | **Code** |
| عبدالرحمن حمدي احمد مخيمر | 1 | 38 | 9202833 |
| عبدالرحمن نعمان لقمان ابراهيم | 2 | 4 | 9202851 |

Submitted To: Eng. Mohamed Abdullah Date: 4/18/2024

**Description:**

With CUDA, we can do things like **searching and sorting** data super quickly by breaking tasks into smaller pieces and doing them all at once. This means you can find what you need in your data faster and manipulate it more easily. We're also using **CUDA** to make joining data from different parts of the **database** much quicker. Joining is like putting together pieces of a puzzle to get a bigger picture. By using CUDA, we can spread out the work across lots of different parts of your computer's graphics card, so it's done in a fraction of the time, even when dealing with really big sets of data. Here we go, this sums up our proposal. We are going to implement a system that prompts the user for input to choose from different **operations** they can carry on the database. The kernels will be the brain of the application where there’ll be a kernel for each operation to interact with the database and retrieve data in the most parallelized and efficient way.

**Block Diagram:**

****

**Experiments:**

1. **Linear Search**

|  |  |  |
| --- | --- | --- |
| **Size** | **CPU** | **GPU** |
| 100 | 1 us | 11 us |
| 1000 | 4 us | 10 us |
| 10000 | 28 us | 11 us |
| 100000 | 261 us | 18 us |
| 500000 | 1301 us | 74 us |

Key Observations:

**Small Dataset Sizes (100 and 1000 elements):**

*CPU is Faster*: For very small datasets, the CPU outperforms the GPU (1 us vs. 11 us for 100 elements, 4 us vs. 10 us for 1000 elements). This is because the overhead associated with launching GPU kernels and data transfer between host (CPU) and device (GPU) outweighs the parallel processing benefits of the GPU.

*Overhead Costs:* The GPU requires additional time to set up the computation environment, including memory allocation, data transfer, and kernel launch overhead. For small data sizes, these costs are significant relative to the actual computation time.

**Medium Dataset Size (10000 elements):**

*Performance Crossover:* At 10,000 elements, the GPU starts to demonstrate its strength in parallel processing (28 us on CPU vs. 11 us on GPU). The overhead costs become less significant compared to the gains from parallel execution on the GPU.

*GPU Advantage:* The GPU becomes more efficient as the data size increases, benefiting from its ability to process multiple elements simultaneously.

**Large Dataset Sizes (100000 and 500000 elements):**

GPU is Significantly Faster: For larger datasets, the GPU vastly outperforms the CPU (261 us vs. 18 us for 100,000 elements, 1301 us vs. 74 us for 500,000 elements). The parallel nature of GPU computation allows it to handle larger data sizes much more efficiently.

Scaling Benefits: The GPU's performance scales better with increasing data sizes due to its high degree of parallelism. Each GPU core can handle a portion of the data concurrently, leading to a significant reduction in overall computation time.

Reasons for the Performance Differences:

Parallelism:

CPU: Generally has fewer cores (4 to 16 for consumer-grade CPUs) that are optimized for sequential processing. While it can handle parallel tasks through multi-threading, the degree of parallelism is limited compared to a GPU.

GPU: Consists of hundreds or thousands of smaller cores designed for parallel tasks. Each core can perform simple operations simultaneously on different pieces of data, making it ideal for tasks like linear search across large datasets.

Overhead:

* GPU Overhead: Includes the time taken for memory allocation on the GPU, data transfer from host to device, and kernel launch. This overhead is relatively constant regardless of data size, making it a significant factor for small datasets.
* CPU Overhead: Generally lower for small tasks, as there is no need for data transfer between different memory spaces or kernel launch overhead.

Memory Bandwidth:

* CPU: Typically has lower memory bandwidth compared to a GPU, which can limit the speed of accessing large datasets stored in memory.
* GPU: Designed with high memory bandwidth, allowing faster access to data, which is beneficial for tasks involving large datasets.

**Performance Analysis**

Theoretical Benchmarks

|  |  |
| --- | --- |
| **Array Size** | **Total Time** |
| 100 | 0.066 us |
| 1000 | 0.660 us |
| 10000 | 6.6 us |
| 100000 | 66 us |
| 500000 | 330 us |

Speedup of the GPU is below the theoretical due to the following factors:

1. Kernel launch overhead
2. Memory transfer overhead
3. Suboptimal memory access patterns
4. Warp divergence

Performance Comparison with Open-Source peers

**Optimization Levels:**

Open-source frameworks like **PyTorch** and **TensorFlow** are highly optimized for a wide range of operations and hardware configurations.

They leverage advanced optimization techniques and libraries like cuDNN and cuBLAS, which are specifically optimized for NVIDIA GPUs.

Development and Maintenance:

These frameworks are developed and maintained by large teams with deep expertise in GPU programming and access to proprietary optimization techniques from hardware vendors.

Custom implementations may not reach the same level of optimization due to limited resources and expertise.

Algorithm-Specific Optimizations:

Open-source frameworks often include algorithm-specific optimizations that may not be present in custom implementations.

For example, PyTorch and TensorFlow include highly optimized routines for common tasks like matrix multiplications, convolutions, and reductions.

1. **Merge Sort**

|  |  |  |
| --- | --- | --- |
| **Size** | **CPU** | **GPU** |
| 100 |  |  |
| 1000 |  |  |
| 10000 |  |  |
| 100000 |  |  |
| 500000 |  |  |

1. **Inner Join**

|  |  |  |
| --- | --- | --- |
| **Size** | **CPU** | **GPU** |
| 100 | 137 us | 303 us |
| 1000 | 4.202 ms | 1.6895 ms |
| 10000 | 241.425 ms | 15.352 ms |
| 100000 | 25.68 s | 335.95 ms |
| 500000 | 363.9 s | 4.012 s |

Performance Analysis of Inner Join on CPU vs GPU

The results analyze the performance of inner join operations on CPUs and GPUs for varying data sizes. The data presented highlights the advantages of GPUs for processing large datasets while also revealing areas for potential optimization.

Observations

The provided results show a clear trend:

GPU Advantage for Large Data: As data size increases, the GPU exhibits significant speedup compared to the CPU. For instance, at 10,000 data points, the CPU execution time is 241.425 ms, whereas the GPU takes only 15.352 ms (a 15.7x speedup). This trend continues with even larger datasets, demonstrating the scalability of GPU-based processing.

CPU Overhead for Small Data: For smaller datasets (100 and 1,000 data points), the GPU is slower than the CPU. This is likely due to the overhead associated with transferring data between CPU and GPU memory and kernel launch. This overhead becomes less significant compared to the actual join operation for larger datasets.

Theoretical Benchmarks and Speedup

CPU: Due to factors like CPU architecture, cache size, and memory access patterns, it's challenging to provide a specific theoretical benchmark for CPU inner join performance. However, the expected behavior is that join time should increase proportionally to data size (n \* log(n)) in most scenarios.

GPU: Inner join algorithms on GPUs have the potential to achieve theoretical speedups of 10x to 100x compared to CPUs due to their massively parallel processing capabilities. This is because GPUs can process multiple join comparisons simultaneously.

The results show a significant speedup for larger datasets. At 500,000 data points, the GPU is almost 91 times faster than the CPU. However, the observed speedup is likely lower than the theoretical maximum due to several factors:

Memory Transfer Overhead: Transferring data between CPU and GPU memory takes time, especially for small datasets where the transfer time becomes a significant portion of the overall execution time.

Underutilization of GPU: Depending on the implementation, the GPU might not be fully utilized for smaller datasets. Optimizations for thread scheduling and data partitioning can improve GPU utilization.

Algorithm Choice: The specific inner join algorithm implemented on the GPU might not be perfectly suited for the hardware architecture. Exploring alternative algorithms could potentially lead to better performance.

Optimizations for Improved Speedup

Several strategies can be employed to improve the speedup achieved with GPUs:

Data Transfer Optimization: Techniques like asynchronous data transfer or using pinned memory can reduce memory transfer overhead.

Kernel Tuning: Optimizing the GPU kernel code for better thread utilization and memory access patterns can improve performance.

Algorithm Selection: Researching and implementing GPU-specific inner join algorithms that are highly parallel and optimized for the specific hardware can lead to significant performance gains.

Comparison with Open-Source Peers

A definitive comparison with open-source libraries requires knowledge of the specific libraries or frameworks used in the implementation. However, popular deep learning frameworks like PyTorch and TensorFlow offer highly optimized GPU kernels for various operations, including joins. Comparing the results with these frameworks can provide insights into potential performance improvements. Benchmarks and performance comparisons for these frameworks can be found online.

Conclusion

The results demonstrate the effectiveness of GPUs in accelerating inner joins, especially for large datasets. While the observed speedup might be lower than the theoretical maximum, there's room for improvement through optimizations in data transfer, kernel tuning, and algorithm selection. Comparing with open-source libraries can provide valuable benchmarks and potential areas for further optimization. This analysis highlights the importance of considering both theoretical capabilities and practical limitations when evaluating the performance of CPUs and GPUs for database operations.