**Experiment No. – 02**

**Aim:** To study and implement a hashing-based sorting algorithm using CUDA C++.

**Objective:** To gain insights into parallel sorting techniques, leverage the power of GPUs for sorting large data sets efficiently, and contribute to the development of high-performance GPU-accelerated sorting solutions.

**Theory:**

Hashing is a fundamental concept in computer science that involves mapping data of arbitrary size to fixed-size values, typically integers. A hash function is used to perform this mapping, taking an input (or "key") and producing a hash value or hash code. The hash function should have the following properties:

* ***Deterministic***: For a given input, the hash function should always produce the same hash value.
* ***Fast Computation***: The hash function should be computationally efficient to calculate the hash value.
* ***Uniform Distribution***: The hash function should distribute the hash values uniformly across the range of possible hash codes, reducing the likelihood of collisions (different inputs producing the same hash value).

Hashing is widely used in various algorithms and data structures for different purposes, including:

* **Hash Tables**: Hash tables use hashing to provide efficient key-value pair storage and retrieval. The hash function maps keys to indices in an array, allowing constant-time average case lookup, insertion, and deletion.
* ***Hash Sets***: Similar to hash tables, hash sets store unique elements using a hash function to map elements to indices. They support efficient membership testing, insertion, and deletion operations.
* ***Hash-based Sorting***: Hashing can be employed in sorting algorithms, as seen in hashing-based sorting algorithms like Counting Sort, Radix Sort, and Bucket Sort. Hashing is used to distribute elements into different buckets or slots before sorting them within each bucket.
* ***Data Integrity and Verification***: Hash functions are widely used for data integrity checks and verification purposes. For example, cryptographic hash functions ensure data integrity by producing a fixed-size hash value that uniquely represents the input data.
* ***Hash-based Data Structures***: Various data structures, such as Bloom filters and hash-based indexing structures, utilize hashing to provide efficient data access and storage.

Hashing plays a crucial role in algorithm design by enabling efficient data organization, retrieval, and verification. It helps reduce search complexity and improve performance by leveraging the properties of hash functions to distribute and organize data effectively.

Hash-based sorting algorithms are efficient sorting algorithms that utilize hash functions to distribute elements into different buckets or slots. These algorithms typically involve three main steps: hashing, collision resolution, and output generation.

***Hashing***:

* Hash functions map elements to unique keys or indices in a hash table or an array.
* The hash function should distribute elements uniformly across the hash table or array to minimize collisions and optimize performance.
* Common hash functions include division hashing (using modular arithmetic), multiplication hashing, and universal hashing.

***Collision Resolution***:

* Collisions occur when two or more elements are mapped to the same hash table index or slot.
* Various collision resolution techniques can be employed to handle collisions effectively.
* Chaining is a common approach where each index or slot contains a linked list or an array to store multiple elements with the same hash value.
* Other techniques include open addressing (probing neighboring indices until an empty slot is found) and cuckoo hashing (rehashing elements to alternative locations).

***Output Generation***:

* Once the elements are distributed and collisions are resolved, the sorted order is determined based on the hash table or array contents.
* The elements can be extracted from the hash table or array in a specific order (e.g., ascending or descending) to generate the sorted output.

Hashing-based sorting algorithms are primarily used for sorting large datasets efficiently. When implemented using CUDA C++, these algorithms can take advantage of parallel processing capabilities offered by GPUs to further improve performance.

CUDA C++ Implementation of Counting Sort

One example of a hashing-based sorting algorithm is the Counting Sort algorithm. It works by counting the occurrences of each distinct element in the input array and then using these counts to determine the sorted order of the elements.

Here is a high-level overview of how you can implement Counting Sort using CUDA C++:

1. Allocate a global histogram on the GPU.
2. Partition the input array into chunks and distribute them among CUDA threads.
3. Each thread processes its assigned chunk and updates the global histogram accordingly.
4. Synchronize the threads to ensure all computations are complete.
5. Use the global histogram to determine the sorted order of the elements. Each thread updates the output array based on the global histogram.
6. Synchronize the threads to ensure all computations are complete.

This implementation of Counting Sort takes advantage of the parallel processing capabilities of GPUs to improve performance. By distributing the work among CUDA threads.

**Conclusion:**   It can be concluded that implementing a hashing-based sorting algorithm using CUDA can significantly improve sorting performance compared to traditional CPU-based sorting algorithms.