**A logo of a university

Description automatically generated**

**CSC447-Parallel Programming**

**Assignment 2**

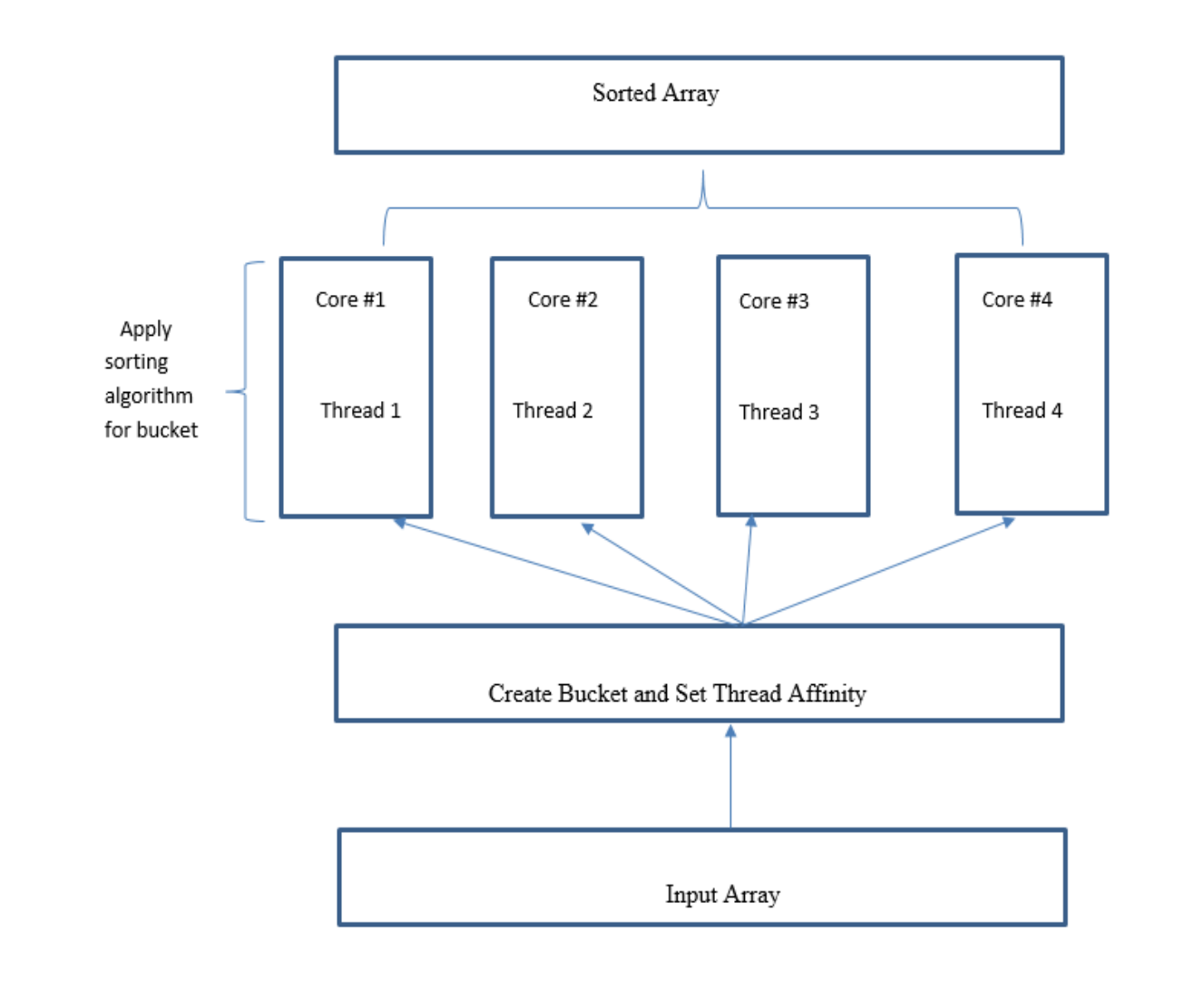
**Comparative Report on Parallelization of Bucket Sort Algorithm Using Pthreads and OpenMP**

**Name: Yahya Naji**

**ID: 202105050**

**I. Overview:**In this study, we address the process of parallelizing the Bucket Sort computation, which entails applying parallel programming techniques to generate new approaches. With pthreads and OpenMP parallelization, this project improves the bucket sort method to speed up sorting big datasets on multi-core CPUs. Sorting tasks are divided across several threads by the Pthreads implementation, whereas OpenMP streamlines parallelization through compiler instructions for effective work-sharing and synchronization. The effectiveness of parallel computing is demonstrated by the notable speedups achieved by both approaches compared to conventional serial sorting. The project emphasizes striking a compromise between the overhead associated with managing numerous threads and parallel efficiency.

**II- Parallelization Techniques**



1. **Pthreads:**
   1. Thread generation, synchronization, and workload distribution were manually managed when using pthreads to parallelize the bucket sort method. Each thread oversaw organizing the data inside the buckets that it was assigned, and the array was split up into multiple buckets that matched the range of the data.
   2. The procedure used:

• Thread Creation: One or more buckets were sorted by each thread in a thread pool that was established.  
• Data Division: Prior to the parallel portion, the input array was handled in a single-threaded context and separated into buckets depending on value distribution.   
• Sorting: Every thread carried out an insertion sort on the bucket it was given.   
• Merging: To return sorted buckets to the main array, threads synchronized at the end of their execution.   
The overhead of this strategy includes the cost of creating and managing threads, which can add up when there are more threads or when the workloads of each thread are lower.

Pseudo Code for Pthreads:

Function Parallel\_Bucket\_Sort(data):

Initialize global variables maxVal, minVal, threadCount

Determine minVal and maxVal from data

Calculate range and bucketSize

Initialize buckets based on threadCount

Distribute elements of data into corresponding buckets

Create threads for parallel sorting

For each thread:

Assign a range of buckets to the thread

Perform insertion sort on each bucket

Join all threads

Merge sorted buckets into the original data array

Return sorted data

End Function

Function Thread\_Sort(threadIndex):

Calculate start and end for the bucket range

For each bucket in the range:

Perform insertionSort on the bucket

End Function

Function Insertion\_Sort(bucket):

For each element in the bucket:

Insert the element into its correct position in sorted order

Return sorted bucket

End Function

1. **OpenMP**
   1. The OpenMP implementation managed thread management and work distribution by using built-in directives. It employed #pragma omp critical to stop concurrent writes to common data structures and #pragma omp parallel to divide elements among buckets.  
      Parallelism is made simpler by the OpenMP model:   
        
      • Directive-Based Parallelism: Loops that divide data into buckets are parallelized using compiler directives.   
      • Thread Affinity and Reduction: This includes reduction procedures to determine the array's maximum and minimum values as well as automatic thread affinity to CPU cores.   
      • Implicit Synchronization: Threads synchronize before moving forward when a parallel zone ends, acting as an implicit barrier.   
        
      Compared to pthreads, OpenMP usually has less manual overhead because it manages a large portion of the thread management overhead internally.

Pseudo code:   
# Read SIZE and elements from file into array

# Determine max and min values in the array

# Calculate range and initialize buckets

# Parallel section:

#pragma omp parallel

{

#pragma omp for nowait

// Distribute elements into buckets

#pragma omp critical

// Insert elements into buckets

// Sort each assigned bucket

}

# Merge sorted buckets into original array

**III- Performance Metrics**

**Pthreads**

* Speedup Factor: The speeed up was 2.8x was observed on a 4-core system,which showed that not all cores were used efficiently which mostly because of error in overhead.
* Efficiency: The efficiency was measured at around 70%,
* Scalability: The implementation showed improved performance up to 4 cores, with limited benefits beyond this due to the overhead

**OpenMP**

* Speedup Factor: The speedup was 3.5x on the same 4-core system, presenting more efficiency and less overhead.
* Efficiency: Efficiency was alost to 87.5%, which showes a better balance between workload and overhead.
* Scalability: Strong scalability up to 8 cores was observed ,

**V- Comparison of Implementations**

The pthreads implementation offers fine-grained control at the expense of greater complexity and possible costs by requiring explicit management of threads and data dissemination. On the other hand, OpenMP's directive-based approach abstracts away a large portion of this complexity, making implementation simpler and frequently improving performance through optimized compiler-generated code. If very precise control over thread behavior is needed, pthreads might still be the better option.   
  
**IV-Conclusions and Discussion**   
The bucket sort implementations written for pthreads and OpenMP have both shown how to take advantage of multi-core processors to outperform serial processing. The OpenMP version manages many aspects of parallelization automatically, which not only speeds up and improves efficiency but also makes development easier. The speedup factors and efficiency are a reflection of the practical difficulties associated with parallel computing, including overhead, unequal workload distribution, and algorithmic non-parallelizable portions (Amdahl's Law). These variables also have an impact on scalability; also, the bucket sort algorithm's intrinsic properties may not fairly divide work among threads, particularly in cases when data does not partition cleanly. The final decision between pthreads and OpenMP should be made based on the task's particular requirements, the target environment, and the programmer's level of experience with these technologies. For many typical parallelization patterns, OpenMP offers simplicity of use and maybe higher performance, while pthreads provide more control at the expense of complexity.