

Programming Assignment  
Section A

CIS6007: Parallel and Distributed Systems

15 November 2022

<https://github.com/arioanindito/parallel-1>

**Prepared by**

|  |  |
| --- | --- |
| Name: | Ario Anindito |
| Student ID: | st20250662 |
| Programme: | BSc (Hons)  Software Engineering |

**TASK 1: IMPLEMENT A PARALLEL VERSION OF THE BUBBLE SORT SORTING ALGORITHM**

**Parallel Bubble Sort Using Threads**

The bubble sort sorting algorithm is a sorting algorithm for a given array. The way it works is by comparing one element to another and swap. When the initial element is greater than the next element in comparison, we swap the elements. The algorithm stops once no swap occurs. In the case of multi-threads, all phases can occur simultaneously, which is known as a parallel implementation using threads.

**Evaluation Of The Task**

1. The problem can be parallelized.
2. The problem can be partitioned by splitting the array into a small sub-list.
3. Yes, the communications are needed.
4. No, there is no data dependencies.
5. Yes, the synchronization is needed.
6. Yes, the load balancing will be a concern.

**Structure Of the Program**

1. The main program is start from declare the number of threads, running the stopwatch from system diagnostic, and call the parallelSorting method. After all the process finished, the stopwatch will stop and give the elapsed milliseconds.

Text

Description automatically generated

1. Inside the parallelSorthing method, the first algorithm that will be executed is making a sublist from the array, split according to the number of threads that are running.

Text

Description automatically generated

With this way, we have a group of sublist that will be sorted by each running threads simultaneously.

1. Afterwards, we start the threads and call the bubbleSort method to perform the sorting algorithm. After all the sublist are sorted, we join them back.

Text

Description automatically generated

1. The bubble sort algorithm itself is quite simple, by traversing all the array elements and check if current element greater than next element.

Text

Description automatically generated

If the current element is greater than next element, we swap the elements. Lastly, using the WriteLine function we print the performance diagnostic.

**The Results**

Text

Description automatically generated

Based on the results above, the **1 thread** performance takes **102,541 milliseconds**, the **2 threads** performance takes **26,123 milliseconds**, the **3 threads** performance takes **13,102 milliseconds**, the **4 threads** performance takes **9,214 milliseconds**, and **6 threads** performance takes **6,508 milliseconds**. In order to see the details how each group-threads perform; we can use Stopwatch in System.Diagnostics to see how long it takes to finished in milliseconds. As it shown.

In a conclusion, not only this problem can be parallelized using a different range of threads, but also the more threads are involved the faster the process is.

In addition to implement a parallel version of the bubble sort sorting algorithm, there is also another approach which we can divide sorting of the unsorted into two phases, which are odd phase and even phase.

When it comes to the odd phase, we compare the element at index 0 with the element at index 1, and so on. On the other hand, for the even phase we compare index 1 element with index 2 element, and so on. We compare all pairs of elements in the array asynchronously in both odd phase and even phase.

**TASK 2: A PARALLEL ALGORITHM FOR A VERY LARGE COLLECTION OF INVENTORIES**

**Structure Of the Program**

1. The first configuration of the code is by implementing the element of an array (100,000 elements), the number of the threads, and make the dictionary for the specific number of items that we will get from search algorithm from the list.

Text

Description automatically generated

1. The implementation for question and barcode generator.

Text

Description automatically generated

1. Implementation for the sub list of the give list.

Text

Description automatically generated

1. The implementation of finding barcodes algorithm.

Text

Description automatically generated

From all of those lines of code above, we can execute the search algorithm for given list parallelly hence it will make the runtime faster. By using the threads, we can make the sub list within the given list and execute the algorithm from each threads to sub list.

**The Results**

Table

Description automatically generated with medium confidence

From the results above, we can conclude that the problem can be executed parallelly. Although the process time are vary, it can be caused by the memory allocation when the program starts.

**Evaluation Of The Task**

1. The problem can be parallelized.
2. The problem can be partitioned by splitting the array into a small sub-list.
3. Yes, the communications are needed.
4. No, there is no data dependencies.
5. Yes, the synchronization is needed.
6. Yes, the load balancing will be a concern.